Component-based Timed Hazard Analysis of Self-healing Systems

Claudia Priesterjahn
Heinz Nixdorf Institute and Department of Computer Science, Software Engineering Group, University of Paderborn
Warburger Str. 100
Paderborn, Germany
cpr@uni-paderborn.de

Dominik Steenken
Research Group Specification and Modelling of Software Systems, Department of Computer Science, University of Paderborn
Warburger Str. 100
Paderborn, Germany
dominik@uni-paderborn.de

Matthias Tichy
Organic Computing, University of Augsburg
Universitätsstr. 6a
Augsburg, Germany
tichy@informatik.uni-augsburg.de

ABSTRACT

Today, self-healing is increasingly used in embedded real-time systems, that are applied in safety-critical environments, to reduce hazards. These systems implement self-healing by reconfiguration, i.e., the exchange of system components during run-time that aims at stopping or removing failures. This reaction is subject to hard real-time constraints because reacting too late does not yield the intended effects. Consequently, it is necessary to analyze the propagation of failures over time and also take into account how the propagation of failures is changed by the reconfiguration. Current approaches do not analyze the propagation times of failures and the changes of structural reconfiguration on the failure propagation.

We enhance our hazard analysis approach by extending our failure propagation models by propagation times and taking the system's real-time reconfiguration behavior into account. This allows to analyze how a reconfiguration with certain duration changes the failure propagation of a real-time system and thus whether it is able to prevent a hazard. We show the feasibility of our approach by an example case study from the RailCab project.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – Reliability, Formal methods

General Terms

Algorithms, Reliability

Keywords

Hazard Analysis, Real-time, Reconfiguration, System Safety

ASAS'11, September 4, 2011, Szeged, Hungary.

1. INTRODUCTION

The value creation in today's technical systems is mostly driven by embedded software. Self-* techniques are examples of innovative functionality which is realized by embedded software. This has become a major trend in engineering complex systems (cf. [4]). Self-* postulates that systems adapt autonomously to changes in the system itself, e.g., error occurrences, or the environment. We regard structural reconfiguration as a special case of adaptation that is implemented by adapting the system structure, i.e., creating, deleting or exchanging system components during run-time.

We consider embedded real-time systems that interact with the real world, where they are often employed in safety-critical contexts. Even in the case that the system does not contain any design errors, hazardous situations may be caused by random errors that happen, e.g., due to the wear of physical components. Consequently, these systems have to be analyzed with respect to potential hazards.

Self-healing systems, as an example of self-* systems, perform hazard reduction as defined by Leveson [14]. In our case, they try to reduce the probability of hazards by stopping the propagation of a failure. Because of the real-time aspects, the hazard analysis of such systems also must consider the time properties of the system , i.e., the propagation times of failures as well as the duration of the reconfiguration. This also takes into account that failures propagate while the reconfiguration is executed.

Current automated hazard analysis approaches [1, 5, 9, 11, 15, 22] do not take the propagation times of failures and the changes of structural reconfiguration on the failure propagation into account. They are thus not suitable for self-healing systems.

In previous works [9], we already presented a component-based hazard analysis approach for reconfigurable systems. That approach analyzes the failure propagation of all reachable configurations of a technical system. It determines the combinations of errors that lead to hazards and computes the likelihood that the hazard occurs for each configuration. But so far it does not take time properties of the failure propagation or the duration of the reconfiguration into account.

In this paper, we present a component-based hazard analysis approach which specifically targets reconfigurable real-

time systems that employ self-healing. For this, we introduce an extension of this approach that

(1) uses a specification language for timed structural reconfigurations,

(2) provides a specification language for time properties of the failure propagation,

(3) determines the propagation times of errors and failures through the part of the system architecture which is affected by a reconfiguration and,

(4) checks whether a reconfiguration is fast enough to stop the propagation of a failure and, thus, reduces a hazard.

In the next section, we present our running example. We then present the foundations of our approach, particularly the system structure and the reconfiguration in Sect. 3. The modeling of timed failure propagation and its formalization is presented in Sect. 4. The timed hazard analysis approach follows in Sect. 5. Section 6 contains a discussion of related work. We conclude and give an outlook on future work in Sect. 7.

2. RUNNING EXAMPLE

Our running example is adapted from the RailCab – a rail vehicle that is developed in the RailCab project[1] at the University of Paderborn. Here, we model a simplified subsystem of the RailCab that controls the speed and the active suspension. This subsystem is a safety-critical part of the RailCab. A wrong speed or an erroneous extension of the suspension of the structure can lead to a wrong position of the RailCabs and to harmful accidents like derailment or collision. By this example, we will illustrate our timed hazard analysis approach presented in this work.

The architecture of this subsystem is shown in Figure 1. We model the system architecture with UML deployment diagrams [10]. In order to denote the propagation of hardware failures, we visually add additional hardware ports. A more detailed description of deployment diagrams will be given in Sect. 3

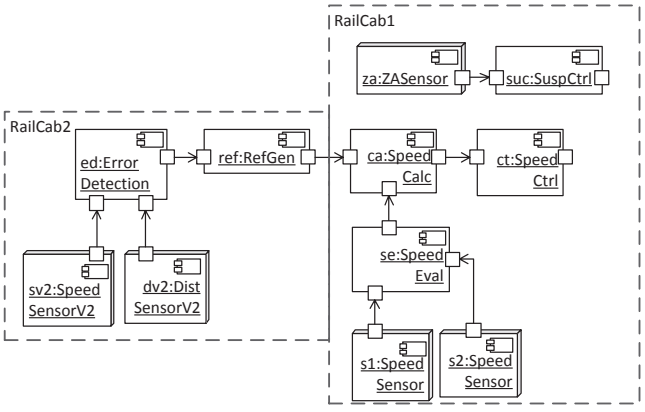


Figure 1: Start Configuration of the Application Example

The speed control subsystem consists of two parts that are deployed on two different RailCabs. The speed sensors s1:SpeedSensor and s2:SpeedSensor (3D-boxes), the evaluation of the speed sensors se:SpeedEval (rectangle), the speed calculation ca:SpeedCalc, and the speed controller ct:SpeedCtrl are deployed on the RailCab of which we want to analyze hazards, namely "RailCab1". The speed sensor sv2:SpeedSensorV2, the distance sensor dv2:DistSensorV2, the error detection ed:ErrorDetection, and the ref:RefGen are deployed on "RailCab2".s1:SpeedSensor and s2:SpeedSensor measure the speed of RailCab1. se:SpeedEval compares the two values of s1:SpeedSensor and s2:SpeedSensor and forwards a speed value to ca:SpeedCalc if the values do not differ too much. ca:SpeedCalc uses this value to calculate a target speed and forwards it to sc:SpeedControl which controls the vehicle's speed. ca:SpeedCalc additionally uses reference data from RailCab2 that is provided by ref:RefGen. sv2:SpeedSensorV2 and dv2:DistSensorV2 measure the speed of RailCab2 and the distance between RailCab1 and RailCab2. The error detection ed:ErrorDetection detects errors of the sensors, e.g. using model-based fault diagnosis [20]. If no error was detected, ref:RefGen generates speed reference data and forwards it to RailCab1. In case of an erroneous value, ed:ErrorDetection triggers a reconfiguration that disconnects ref:RefGen from ca:SpeedCalc, in order to prevent the failure from causing a hazard.

For simplicity, we assume, that the error is detected immediately when an error enters ed:ErrorDetection and the reconfiguration is started without delay. This assumption does not affect the hazard analysis, because the delay between the entrance of the failure into the component and the start of the reconfiguration is not part of the failure propagation but part of the system behavior. As this paper is about hazard analysis, we omit this part.

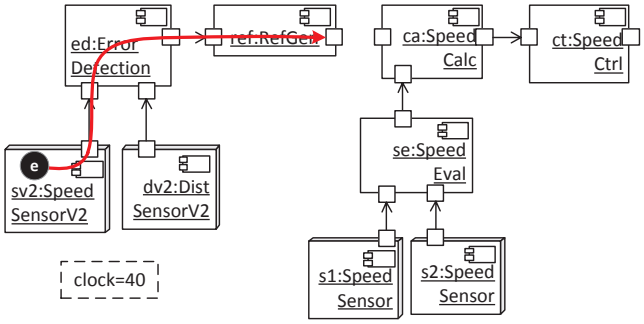
The suspension subsystem consists of the sensor za:ZASensor that measures the deflection of the RailCab's structure along the Z-Axis and a control suc:SuspCtrl that computes the values to control the active suspension. In the remainder of this paper, the suspension subsystem will only be used to illustrate the hazard definition. For the illustration of the failure propagation, we will concentrate on the speed control subsystem.

It is a hazardous situation if the structure of the RailCab has a wrong position as this could lead to a derailment. The structure of the RailCab1 has a wrong position if ct:SpeedCtrl or suc:SuspCtrl emit wrong values. A value failure of ct:SpeedCtrl results in a too strong or too weak acceleration which causes the structure to have a wrong position. A wrong value of suc:SuspCtrl causes a wrong position by a wrong extension of the suspension.

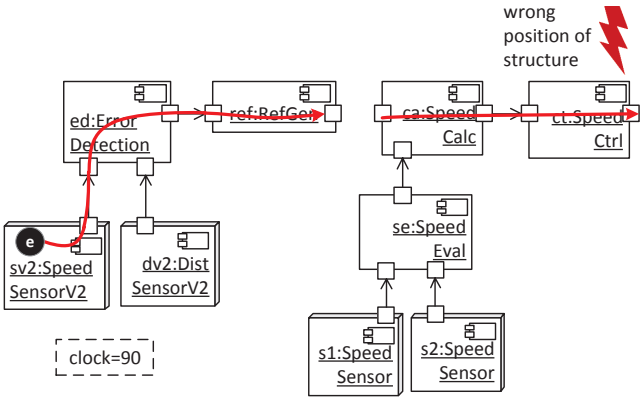
In order to prevent this hazard, the system reacts by removing the connector between ref:RefGen and ca:SpeedCalc. Figure 2 shows the effects of this reconfiguration for different points of time. In both figures, we show the propagation of an error in s2:SpeedSensorV2 which could propagate to ct:SpeedCtrl which then would then output a wrong value.

In Figure 2(a) the reconfiguration is executed at a clock value of 40 time units. We see that the failure has not been propagated to the components ca:SpeedCalc and ct:SpeedCtrl as the reconfiguration has changed the structure before this happens. In Figure 2(b), we consider the case that the reconfiguration is executed at a clock value of 90 time units. We now see that even though the reconfiguration has been executed, the failure has already propagated to the component sc:SpeedCtrl and caused the hazard.

[1] http://www.railcab.de/en



(a) Reconfigured at a Clock Value of 40 Time Units



(b) Reconfigured at a Clock Value of 90 Time Units

Figure 2: Failure Propagation during Reconfiguration

To prevent the latter case, our hazard analysis takes time properties into account. The basic idea here is that the propagation of an error can be stopped by reconfiguring the component structure into another configuration in which the error does not cause a hazard. For that to be possible, the reconfiguration has to be faster than the propagation of this error.

For this, our timed hazard analysis computes the time required by the failure to propagate through that part of the configuration which is changed by the reconfiguration. This time is compared to the duration of the reconfiguration.

3. MODELING THE SYSTEM STRUCTURE

In this section we present our models that specify the system architecture and the reconfiguration.

We define the system architecture using UML deployment diagrams. Deployment diagrams consist of component instances – a modular system unit either hardware or software drawn by rectangles. Each component instance has a number of ports, illustrated by squares at the edges of the component instances, that specify communication interfaces. Connectors connect component instances by their ports. Connectors are represented by arrows which specify the direction of messages that are exchanged.

Regarding the specification of the system architecture, we distinguish between component types and component instances. Component types define a component that can be used to derive a number of component instances which are used to build the system architecture. Each component type can be instantiated multiple times.

A configuration is a concrete assembly of component instances which we illustrate by UML deployment diagrams. Figure 1 shows a configuration of our example system. Due to reconfiguration a system has several possible configurations.

In order to present our analysis approach in a formal manner, we will define the necessary elements of our approach formally. We start with the definition of the system architecture, i.e, configurations.

Definition 3.1 (Configuration)
We define the component specification $s = (K, P, \pi)$ *of a system with* K *the set of* component types, P *the set of* port types, *and* $\pi : K \to 2^P$.

We define a configuration *of a system by* $c = (\overline{K}, \overline{P}, t, L)$ *with the set of* component instances $\overline{K}$, *the set of* port instances $\overline{P}$, *a pair of* typing functions $t = \left(t_K : \overline{K} \to K, t_P : \overline{P} \to P\right)$, *and a set of connectors* $L \subseteq \overline{P} \times \overline{P}$. *The set of all configurations of a system* s *is denoted by* C

By $k.p$, *we denote port type* p *of component type* k. *The same notation holds for component instances.*

Note that ports are never instantiated on their own. They are rather instantiated as adjuncts of their respective component type.

For the definition of our structural reconfiguration we use *durative graph transformation rules* (DGTR) [6]. Reconfiguration means changing the set of component instances and their interconnections at run-time. Thus, a reconfiguration transforms one configuration into another configuration. A DGTR is a classic graph transformation rule (cf. [6,19]) extended by a time interval that specifies the minimum and maximum time needed to execute the DGTR.

We will now give a high-level explanation of the DGTR we employ. Any rule consists of two graphs, a *left hand side*(LHS) and a *right hand side*(RHS). In our case, each of the graphs represents a subconfiguration [21]. In order to apply the rule to a given configuration, we must first find a *matching* of the LHS within the configuration, i.e. we need to find a part of the configuration that is isomorphic to the LHS. Then, we remove the matched part and replace it by the RHS, redirecting incoming and outgoing connections to the rest of the configuration as the isomorphism dictates.

In order to achieve a succinct notation for a transformation rule, we merge both the LHS and the RHS into one graph, annotating the parts occuring in the LHS but not in the RHS with the stereotype `<<destroy>>` and elements occurring in the RHS but not in the LHS by `<<create>>`. These stereotypes are implicitly carried over to the subobjects of an element, e.g. the deletion of a component results in the deletion of the component's ports, too.

In addition to the LHS and RHS, each rule has a time constraint, called *duration*, that specifies the minimum and maximum units of time it takes to apply the rule to a configuration during run-time.

The DGTR is executed by first matching the LHS to the configuration. Then objects are destroyed and afterwards objects are created. The duration of the DGTR only specifies how long it takes to execute all these operations. Consequently, we do not know, which operation is executed at

which point in time. Therefore, we assume for our hazard analysis approach that all changes of the DGTR are applied at the same time, namely at the point in time when the execution of the DGTR is finished. This is a pessimistic assumption, because when we assume that the reconfiguration is executed at the latest point of time possible, the failures can propagated the farthest possible through the system.

In the remainder, we refer to a DGTR by the tuple $r = (LHS, RHS, d)$ with the duration $d = [d_{min}, d_{max}]$. The application of a DGTR is denoted by $c \overset{r}{\Rightarrow} \hat{c}$ that transforms a configuration c into a subsequent configuration $\hat{c}$. A matching $m(r, c) = c'$ is a subconfiguration c' of a configuration c that is isomorphic to the LHS of the DGTR r.

Figure 3 shows the DGTR for our example. The rule specifies that the connector between ca:SpeedCalc and ref:RefGen and its corresponding ports are deleted. Note that this is a very simple example due to limited space. Of course, it is possible to specify the reconfiguration of complex configurations (cf. [6]).

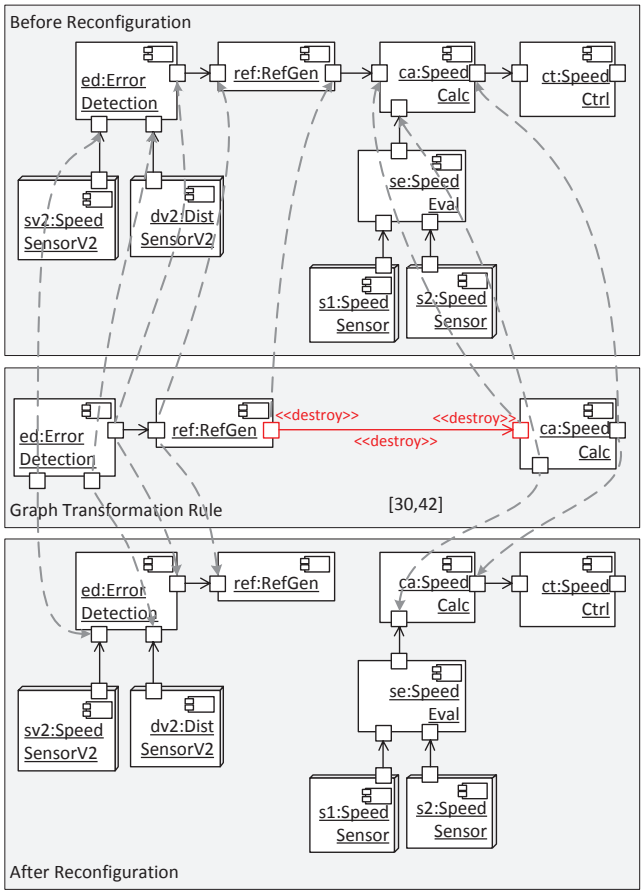


Figure 3: Application of a DGTR on a configuration.

Figure 3 also shows the matching of the ports that is part of the reconfiguration rule. The time constraint in Figure 3 further specifies that at least 30 and at most 42 time units elapse from the time when the application of the rule is initiated and the last change in the configuration that completes the rule. We assume that this information is known by the system developer.

In our approach DGTRs are defined for component types. This means, they are specific to the system but not specific to a configuration of component instances. Further, we did not yet consider the system consistency during reconfiguration.

Timing Assumptions.

As said, we assume that errors are detected immediately when they enter an error detection component. Each DGTR has a duration that specifies the minimum and maximum time needed to execute the associated reconfiguration. Further, we assume that each DGTR has its own clock that starts with zero, when the execution of the DGTR is started. This clock also measures the time during our analysis approach.

4. MODELING TIMED FAILURE PROPAGATION

Having defined the system structure, we model the system's failure propagation, giving particular attention to its time properties. For this, we follow the terminology of Laprie [2] by associating *failures* – the external visible deviation from the correct behavior – to the ports where the component instances interact with their environment. *Errors* – the manifestation of a *fault* in the state of a component – are restricted to the internal of the component.

Failures and errors are typed using an (extensible) failure classification like the one from Fenelon et al. [7]. We distinguish the general error and failure classes *service* and *value*. A value error specifies that a value deviates from a correct value, e.g., an erroneous value in the memory of a component. A service error specifies that no value at all is present, e.g., a component crashed.

We use timed failure propagation graphs (TFPGs) to relate an outgoing failure to a set of combinations of incoming failures, errors and outgoing failures of embedded components. Timing annotations enable the calculation of the propagation times of failures.

For a system, we first specify the error and failure variables for each component type and then we specify the TFPG for each component type manually. For a component specification $s = (K, P, \pi)$, failure and error variables are named according to the following schema: $f^d_{k.p,ft}$ and $e_{k,ft}$ for $k \in K$, $p \in P$, $ft \in \{\text{value}, \text{service}\}$, and $d \in \{\text{i}, \text{o}\}$. ft is the set of error and failure classes. i and o specify the direction of failures – i stands for incoming and o for outgoing.

For a configuration, we instantiate error and failure variables by instantiating component types. The notation, described above, holds for component instances and port instances analogously. This instantiation makes all error and failure variable instances unique.

Definition 4.1 (Timed Failure Propagation Graph) *We define* $I = \{[\Delta t_{min}, \Delta t_{max}] \mid \Delta t_{min}, \Delta t_{max} \in \mathbb{Q}_{\geq 0}$ *(we do not define* $\Delta t_{min}, \Delta t_{max} \in \mathbb{R}_{\geq 0}$ *to be able to map the TFPG to a time petri net in Definition 4.3),* $\Delta t_{min} \leq \Delta t_{max}\}$ *as the set of* propagation time intervals, $\mathcal{E}$ *as the set of* error variables, $\mathcal{F}$ *as the set of* failure variables, *and* $O = \{\&, \geq 1\}$ *as the set of* operators.

We then define the timed failure propagation graph *(TFPG)* $G = (N, E, f_s, f_t, l, \iota, \eta)$ *as a labeled graph (cf. [19]) over* $(\{\mathcal{E} \cup \mathcal{F} \cup O\}, I)$ *where*

- N *is the set of* nodes,
- $E \subseteq N \times N$ *is the set of* edges,
- $f_s, f_t : E \to N$ *are the* source *and* target *functions,*
- $l : N \to \{\mathcal{E} \cup \mathcal{F} \cup O\}$ *is the* node labeling *function, and*
- $\iota : E \to I$ *is the* edge labeling *function.*
- $\eta : N \to \{\text{active}, \text{inactive}\}$

We define $\delta^+(n) = |\{e \in E \mid f_s(e) = n\}|$ *as the* out-degree *and* $\delta^-(n) = |\{e \in E \mid f_t(e) = n\}|$ *as the* in-degree *of a node* $n \in N$. *Let* $N_0 = \{n \in N \mid \delta^-(n) = 0\}$. *Then* $\forall n \in N_0 : l(n) \in \mathcal{E}$ *and* $\forall n \in N \setminus N_0 : l(n) \in \mathcal{F} \cup O$ *hold. This means, all nodes with in-degree zero are labeled with error variables. All other nodes are labeled with either a failure variable or a logical operator* ≥ 1 *or* &.

The TFPG of a configuration is built by connecting the TFPGs of all component instances by the connectors. In order to analyze the delays of inter-component communication, the developer has to assign propagation time intervals to the connectors. Figure 4 shows a configuration of the speed control subsystem from Figure 1 with its TFPG. Of course, the annotated time values are fictitious.

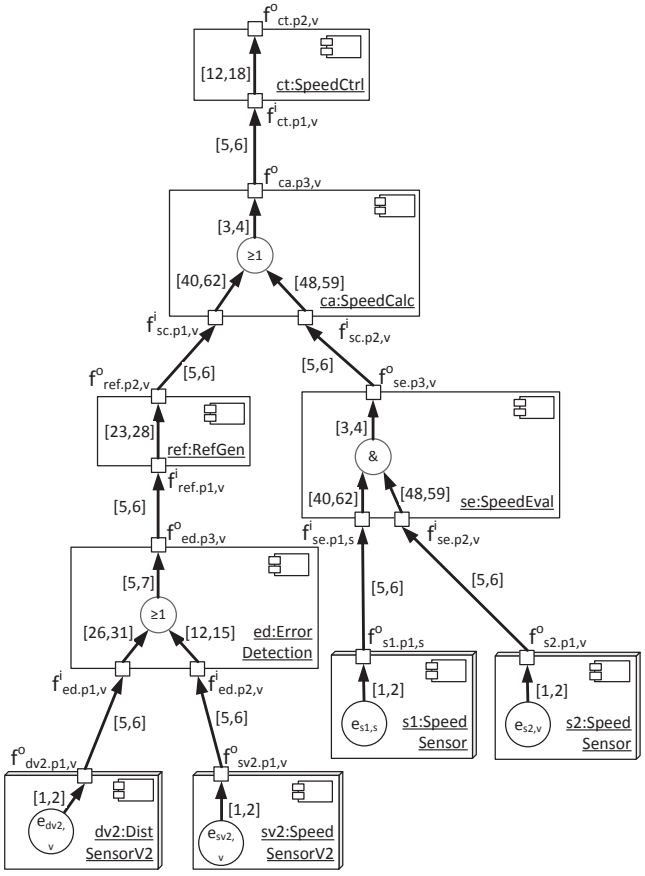


Figure 4: Configuration with TFPGs

When a failure propagates through the system over time, we model this fact by the activation of the error and failure variables of the TFPG. In order to express which error or failure variable has been activated, we set the associated error or failure variable to "active". When the error or failure propagates further through the system, the subsequent failure variables are set to active and the error or failure variable is set to "inactive". This is repeated until the failure variable at the top of the TFPG is reached.

To model this flow, we add a formal semantics in form of a time petri net [3] to our TFPG. Time petri nets are marked petri nets [18] with a time extension. In this paper we consider time petri nets in which a transition is labeled with a time interval. Each transition has a clock that measures the time until the transition has been activated. The transition can only fire if the clock has a value that is within the transition's time interval.

We now give the formal definition of a time petri net as presented in [3]. We assume the semantics as they have been defined in [3].

Definition 4.2 (Time Petri Net (TPN))
[3] A timed petri net *(TPN)* $\mathcal{T}$ *is a tuple* $(P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ *where*

- $P = \{p_1, p_2, ..., p_{|P|}\}$ *is a finite* set of places,
- $T = \{t_1, t_2, ..., t_{|T|}\}$ *is a finite* set of transitions,
- ${}^\bullet(.) \in (\mathbb{N}^{|P|})^{|T|}$ *is the* backward incidence mapping,
- $(.)^\bullet \in (\mathbb{N}^{|P|})^{|T|}$ *is the* forward incidence mapping,
- $M_0 \in \mathbb{N}_0^{|P|}$ *is the initial marking,*
- $\alpha \in (\mathbb{Q}_{\geq 0})^{|T|}$ *and* $\beta \in (\mathbb{Q}_{\geq 0} \cup \{\infty\})^{|T|}$ *are the* earliest *and the* latest firing time mappings.

The morphism defined below allows us to map a TFPG to a time petri net.

Definition 4.3 (Morphism from TFPG to TPN)
We define a graph morphism $\mu : G \mapsto \mathcal{T}$ *from a TFPG* $G = (N, E, f_s, f_t, l, \iota, \eta)$ *to a TPN* $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ *as a tuple* $\mu = (\mu_N, \mu_E, \mu_P)$ *where*

- $\mu_N : N \to P$ *(bijective),*
- $\mu_E : E \to T$ *(bijective),*
- $\mu_I : I \to (\mathbb{Q}_{\geq 0}^{|T|}, \mathbb{Q}_{\geq 0}^{|T|})$ *(bijective) is the mapping from the set of propagation intervals to the earliest and latest firing time mappings.*

with

- *For all* $t \in T$ *the backward incidence mapping is described by* ${}^\bullet(t) = (v_1, ..., v_{|P|})$, *where*

$$v_i = \begin{cases} x & \mu_N^{-1}(p_i) = f_t(\mu_E^{-1}(t)) \\ 0 & else \end{cases}$$

with

$$x = \begin{cases} \delta^-(f_s(\mu_E^{-1}(t))) & l(f_s(\mu_E^{-1}(t))) = \& \\ 1 & else \end{cases}$$

This means that the backward incidence mapping has only one element that is not zero. This element is the

transition following a node labeled with &, and it is set to the in-degree of this &-node.

- *For all $t \in T$ the forward incidence mapping is described by $(t)^\bullet = (b_1, ..., b_{|P|})$, where*

$$b_i = \begin{cases} 1 & \mu_N^{-1}(p_i) = f_s(\mu_E^{-1}(t)) \\ 0 & else \end{cases}$$

This means that the forward incidence mapping has only one element that is one.

- $M_0 = (m_1, ..., m_{|P|})$, *where*

$$m_i = \begin{cases} 1 & \eta(\mu^{-1}(p_i)) = \text{active} \\ 0 & else \end{cases}$$

- $\alpha = (\alpha_1, ..., \alpha_{|T|})$ *where* $\alpha_i = \min(\iota(\mu_E^{-1}(t_i)))$
- $\beta = (\beta_1, ..., \beta_{|T|})$ *where* $\beta_i = \max(\iota(\mu_E^{-1}(t_i)))$

The weight of all edges of the TPN is one. The only exception are edges that originate from &-nodes. They are weighted with the in-degree of the &-node so that the transition after the &-node can only fire if all transitions before the &-node have fired. This represents the fact that an &-node in a TFPG is only satisfied if all predecessor error and failure variables have occurred.

Figure 5 shows the TPN of the TFPG of Figure 4. For better understanding, the labels of the nodes of the TFPG are shown at the corresponding places of the TPN. These labels can be referenced by the reverse application of the morphism and then applying the labeling function of the TFPG: $l(\mu^{-1}(P))$. Note, that only the edge originating from the node labeled with "&" has a weight of two. All other edges are not labeled with weights which means, their weight is one.

In order to make statements about error and failure variables being active or inactive at a point of time, we define the state of a TFPG. As a basis for this, we give the definition of a state of TPNs as used in [3].

Definition 4.4 (State of a TPN, State of a TFPG)
A clock zone z over a a pair of clocks k_1 and k_2 is a set of clock interpretations represented by a conjunction of constraints on pairs of clocks (k_1, k_2): $k_1 - k_2 \sim d$ where $\sim \in \{<, \leq, =, \geq, >\}$ and $d \in \mathbb{Z}$ [3]. A state *$s = (m, z)$ of a TPN $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ is defined by a marking $m \in \mathbb{N}^{|P|}$ and a clock zone z [3].*

A state *$q = (A, Z)$ of a TFPG $G = (N, E, f_s, f_t, l, \iota)$ is defined by a set of active error or failure variables A and a set of clock zones Z. Let $\mathcal{T}$ be the underlying TPN of G and $\mathcal{T} = \mu(G)$ the corresponding morphism. Let $M = (m_1, ..., m_{|P|}) \in \mathbb{N}^{|P|}$ be a marking in $\mathcal{T}$. Then*

$$A = \left(\bigcup_{i=1}^{|P|} \{ l(\mu_N^{-1}(p_i)) \mid m_i > 0 \} \right) \cap (\mathcal{E} \cup \mathcal{F})$$

A collects all active error and failure variables represented by a token in the underlying TPN of the TFPG during the span specified by the clock zone.

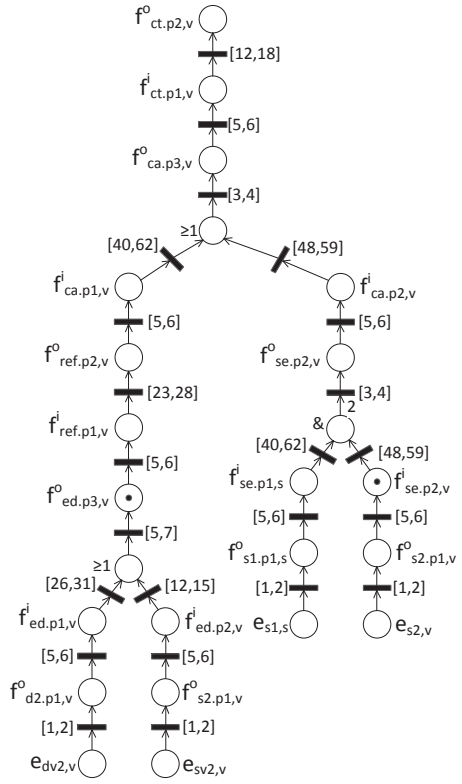


Figure 5: TPN of the TFGP of Figure 4 for the state $(A, [41, 42])$

Figure 5 illustrates the state $(A, [41, 42])$. A is one possible marking for the clock zone $[41, 42]$. It contains the elements that correspond to the places p with $l(\mu^{-1}(p))$ equals $f^o_{ed.p3,v}$ and $f^i_{se.p2,v}$. The places initially marked with a token were $e_{dv2,v}$ and $e_{s2,v}$.

Finally, we have to specify hazards. We use boolean formulas to specify the combinations of outgoing failures of a configuration that cause hazards.

Definition 4.5 (Hazard)
A hazard *h is defined by a boolean formula ψ over failure variables.*

The boolean formula ψ that defines the hazard is represented by a fault tree. The hazard is the top event. We construct the fault tree by applying manual fault tree analysis [12]. For the scope of our analysis, this fault tree does not need timing annotations. We are only interested in the propagation times of failures within a configuration. This fault tree is not part of the configuration. Timing annotations would be of use if one wanted to analyze time dependencies between the errors and failures in the system.

Figure 6 shows the fault tree for the hazard $h_{wrongPosStructure}$ that represents the hazard of a wrong position of the structure of RailCab1 from our running example (cf. Sec. 2). The cause for the hazard is the speed control or the suspension control emitting wrong values. This is represented by the failure variables $f^o_{ct.p2,v}$ and $f^o_{suc.p2,v}$ in Figure 6. The hazard is described by the formula $\psi = h_{wrongPosStructure} \Leftrightarrow f^o_{ct.p2,v} \vee f^o_{suc.p2,v}$.

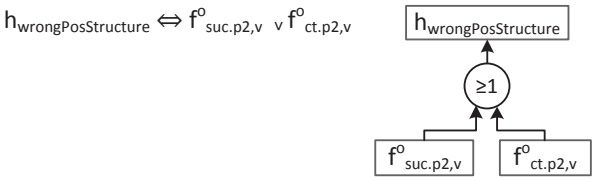


Figure 6: Formula and corresponding fault tree of the Hazard "wrong position of structure"

5. TIMED HAZARD ANALYSIS

Having modeled the failure propagation of our system, we can now perform the timed hazard analysis.

Given the configuration, its TFPG, a DGTR, and a matching, the analysis is done in four steps:

Step 1. Determine the minimal cut sets that lead to the hazard.

Step 2. Extract a subgraph of the configuration's TFPG that is affected by the reconfiguration.

Step 3. Analyze the subgraph of the configuration's TFPG that is affected by the reconfiguration.

Step 4. Analyze, if the reconfiguration was successful in reducing the hazard.

In the following, we present these steps in detail based on our running example.

Step 1 – Determine minimal cut sets

The boolean formula of a hazard (cf. Definition 4.5) contains all failures variables, that build the top-level failures of the TFPGs of the system (cf. $f^o_{ct.p2,v}$ in Figures 4 and 6). A TFPG defines the set of possible causes for such a top-level failure, i.e., sets of error variables that let this failure occur, namely *cut sets* [14].

To compute the cut sets, the TFPG (without propagation time intervals) can be transformed into a boolean formula φ. For this, all nodes of the TFPG with out-degree greater than one have to be divided into sub-nodes with out-degree one. Furthermore, all paths consisting solely of failure nodes are replaced by edges. The result is a syntax tree of φ that can be mapped to the corresponding boolean formula φ.

Definition 5.1 (Cut Set, Minimal Cut Set)
A cut set *s to a failure propagation formula φ of a TFPG G is a conjunction of literals over variables in φ such that $s \rightarrow \varphi$ is a tautology. It can be interpreted as an assignment of boolean values to a subset of the variables in φ that guarantees that φ evaluates to true. The set of all such cut sets for φ is denoted $S(\varphi)$.*

A cut set s for φ is a minimal cut set *iff no sub-term of s is a cut set for φ. The set of minimal cut sets for φ is denoted $S_m(\varphi)$.*

We compute the minimal cut sets that lead to the hazard using our approach previously presented in [9]. To keep the focus on the time aspects of our approach, we refer the interested reader to that publication for details. Cut sets of the failure propagation graph of Figure 4 are $\{e_{dv2,v}\}$, $\{e_{sv2,v}\}$ and $\{e_{s1,s}, e_{s2,v}\}$. They also represent the minimal cut sets.

Step 2 – Partition the TFPG

In Step 2, we identify the subgraph of the TFPG that is affected by the reconfiguration and which we will use in Step 3. For this, we partition the TFPG of the configuration into the two subgraphs *affected subgraph* and *remainder*.

Definition 5.2 (Partitioning of a TFPG)
Given are a configuration $c = (\overline{K}, \overline{P}, t, L)$, a DGTR $r = (LHS, RHS, d)$ and a matching $m(LHS) = (\overline{K}', \overline{P}', L')$ and $m(RHS) = (\overline{K}'', \overline{P}'', L'')$. Let $G = (N, E, f_s, f_t, l, \iota, \eta)$ be the TFPG of c and $h : \overline{N} \rightarrow \overline{K}$ be the mapping from the nodes to the corresponding component instances. The affected subgraph *G_A of G is the graph induced by the union over all paths through G that end at the border of the match. Formally, we define a set X of paths $x = n_1, \ldots, n_m$ where each $n_i \in N$ meets the following conditions:*

- *$l(n_1) \in \mathcal{E}$* *(path starts in an error)*
- *$h(n_m) \in \overline{K'} \dot{\cup} \overline{K''}$* *(path ends in an affected node)*
- *For all extensions of x, $h(n_i) \notin \overline{K'} \dot{\cup} \overline{K''}$ holds for $i > l$.* *(path is maximal)*

We then define G_A as the graph induced by X. We denote $G_R = G \setminus G_A$ the remainder.

The affected subgraph consists of all paths of the TFPG that start with an error variable and end in the part of the configuration that is altered by the DGTR, i.e., they end at a failure that is either part of the LHS or the RHS of the matching and which is the last node on this path that is altered by the DGTR. All other paths that contain only nodes that are not altered by the matching and end at the top-level failures of the TFPG build the *remainder*.

An example that illustrates the partitioning of the TFPG of Figure 4 is shown in Figure 7. The matching was introduced in Figure 3. The affected subgraph consists of the two paths $e_{dv2,v}, ..., f^i_{ca.p1,v}$ and $e_{sv2,v}, ..., f^i_{ca.p1,v}$. $f^i_{ca.p1,v}$ is the last node of this path, because $ca.p1$ is the last part of the configuration on these paths that is altered by the DGTR (cf. Figure 3).

The state of this subgraph is set to $(A, [0, 0])$. The set A of active error and failure variables is specified by the system developer. The clock zone is set to $[0, 0]$ because the clock, that we use for our analysis, starts with our analysis.

Step 3 – Analyze the Affected Subgraph

The DGTR influences the affected subgraph of the TFPG. It can remove error and failure variables or stop errors and failures from propagating further through the system. We now analyze the effect that the DGTR has on the occurrence of a hazard.

We start with the TFPG G of a configuration c. We compute the state (A, z) of the TFPG G for the time interval that is specified by the duration of the DGTR $r = (LHS, RHS, d)$ that is to be applied to reduce the probability of the hazard. The set A then contains all error and failure variables that may be active before the DGTR is executed completely. We take the state at the completion of the DGTR since we do not know exactly, when the individual operations in the DGTR will be performed. By regarding the end of the execution of the DGTR we take a pessimistic view by letting the failures propagate the farthest possible

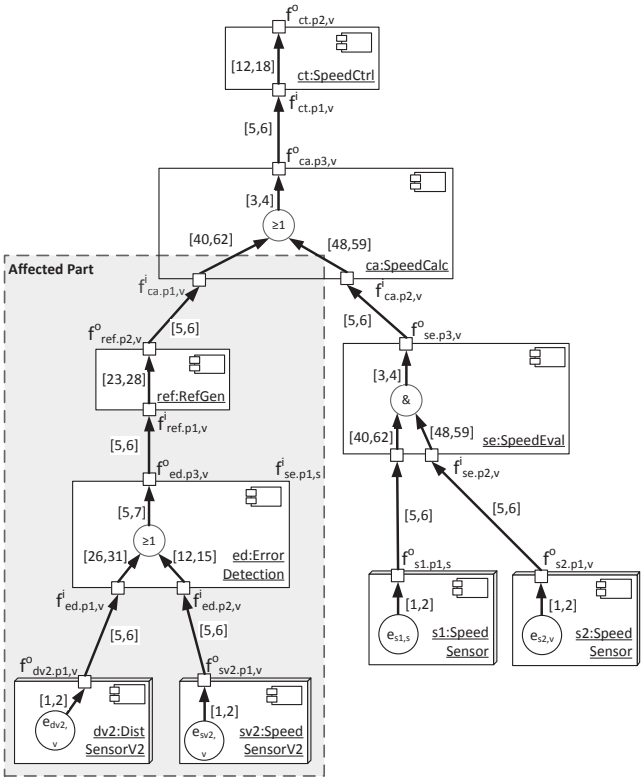


Figure 7: Decomposition of the Configuration

path. With this state of the TFPG we apply the DGTR, i.e., we change the configuration and at the same time the TFPG according to the DGTR. From this TFPG, we extract the active error and failure variables. These correspond to the errors and failures that remain in the system after the reconfiguration. This computation is made by Algorithm 1.

Algorithm 1 first creates the TPN of the affected subgraph that carries the marking of the active error and failure variables (Line 1). With the help of this TPN, it computes the state of the affected subgraph for the duration of the execution of the DGTR (Line 2). This step is explained in further detail in algorithm 2 below. The algorithm then applies the DGTR on the underlying configuration of the affected subgraph (Line 3) and builds the affected subgraph of the resulting configuration (Line 4) as we have described in Step 2. Afterwards the remaining active error and failure variables of the affected subgraph are collected in A'' in Line 5 and returned (Line 6).

Algorithm 2 computes the state of a TFPG by a given clock zone and the corresponding TPN. For this, it computes the set of reachable markings of the TPN for the clock zone z using the approach of [3] (Line 2). Then, it analyzes each marking and stores each error or failure variable, of which the corresponding place in the TPN contains a token in at least one marking (Lines 3-9). Then, A holds all error and failure variables that may be active during z.

Figure 8 shows different states of the TPNs of the affected subgraph during the analysis. Figure 8(a) shows the state of the TPN $\mathcal{T}$ of the affected subgraph of Figure 7 at the beginning of the reconfiguration. Failure variable $f^i_{ed.p1,v}$ has been activated because the failure has been recognized by

Algorithm 1 AnalyzeAffectedSubgraph

Input: $c = (\overline{K}, \overline{P}, t, L)$ a configuration,
$G_A = (N, E, f_s, f_t, l, \iota, \eta)$ the affected subgraph,
$A \subseteq \mathcal{E} \cup \mathcal{F}$ the active error and failure variables,
$r = (LHS, RHS, d)$ the DGTR
Output: A'' the active error and failure variables after the reconfiguration for the lifetime of the system

1: $\mathcal{T} = \mu(G_A)$ with $M_0 = (m_1, ..., m_{|P|})$ with

$$m_i = \begin{cases} 1 & (\mu^{-1}(p_i)) \in A \\ 0 & else \end{cases}$$

2: A' := ComputeTFPGState ($\mathcal{T}$,d)
3: $c \overset{r}{\Rightarrow} \hat{c}$
4: $\hat{G}_A = (\hat{N}, \hat{E}, \hat{f}_s, \hat{f}_t, \hat{l}, \hat{\iota})$:= buildAffectedSubgraph($\hat{c}$)
5: $A'' = \{x \in A' \mid x \in \hat{l}(\hat{N})\}$
6: return A''

Algorithm 2 ComputeTFPGState

Input: TPN $\mathcal{T}$, clock zone z
Output: A the set of active error and failure variables for the clock zone z

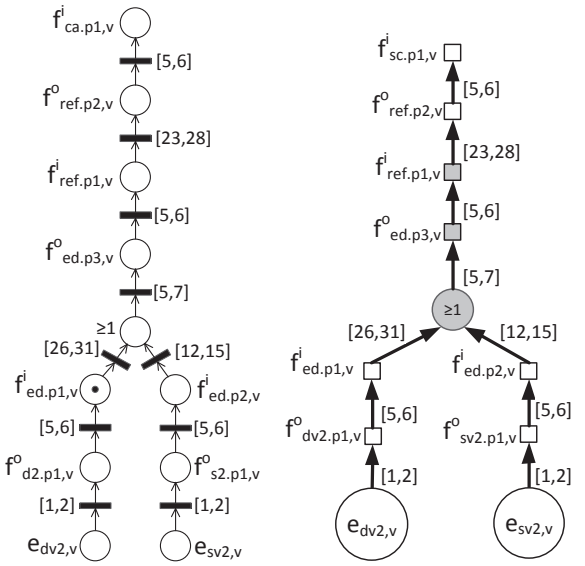
1: $A = \varnothing$
2: $\mathcal{M}$ = getReachableMarkings($\mathcal{T}$, z)
3: **for all** $M \in \mathcal{M}$ **do**
4: **for** $i = 1$ to $|P|$ **do**
5: **if** $m_i > 0$ **then**
6: $A = A \cup l(\mu^{-1}(p_i))$
7: **end if**
8: **end for**
9: **end for**
10: return A

the component instance ed:ErrorDetection. Figure 8(b) shows the TFPG marked with the active error and failure variables of the set A' that may be activated during the reconfiguration. Within the duration $d = [30, 42]$ of the DGTR shown in Figure 3 the failure variables $f^o_{ed.p3,v}$ and $f^i_{ref.p1,v}$ are reachable and may thus be activated in the real system.

Step 4 – Analyze the system after the reconfiguration

In the last step, we analyze if the remaining active error and failure variables still can cause the hazard that was to be reduced. For this, we take the active error and failure variables that remain in the system after the reconfiguration and that were the result of Algorithm 1, i.e., the set A''. We assign A'' the active error and failure variables to the TFPG $G = (N, E, f_s, f_t, l, \iota, \eta)$ of the configuration: $\eta(n) = active \Leftrightarrow l(n) \in A''$. We then create the TFPG G' that is induced by all paths of G that do not contain an active failure variable or that start at the active failure variable f and end at the top-level failure and that do not contain any other failure variable than f. For the resulting TFPG, we compute the minimal cut sets using our hazard analysis from [9]. If none of the minimal cut sets builds a subset of A'', the reconfiguration has been successful in reducing the probability of the hazard. Otherwise, the hazard can still occur.

For our example, we yield the cut sets $\{f^i_{sc.p1,v}\}$ and $\{e_{s1,s}, e_{s2,v}\}$ and the state $(\{f^o_{ref.p2,v}\}, [30, 42]$. $f^o_{ref.p2,v}$ is



(a) Marking of the TPN at the Start of the Reconfiguration

(b) State of the Affected Subgraph for the Duration of the DGTR

Figure 8: TPN and TFPG during Reconfiguration

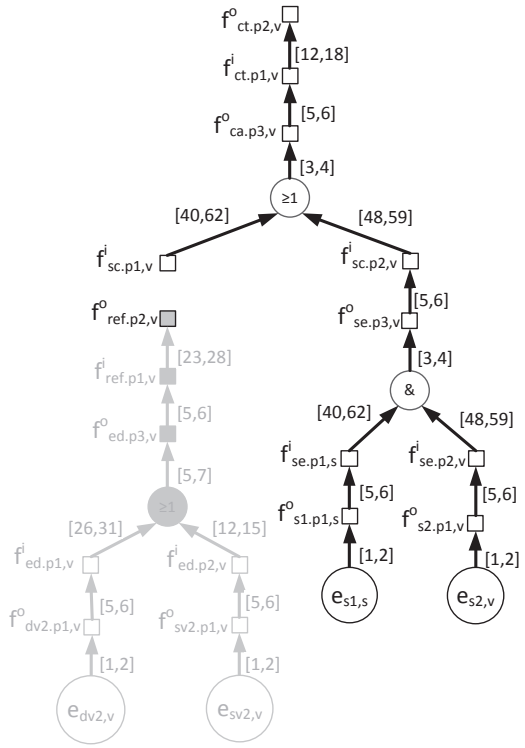


Figure 9: TFPG of the Configuration after the Reconfiguration

the only active failure variable in the TFGP and it does not contain any of the variables of the cut sets. Thus, our reconfiguration has been successful in reducing the hazard.

Cycles.

Cycles in the TFPG can also be handled by our analysis because we map our TFPG to a TPN. The reachability analysis of [3] can handle cycles in TPNs. Further, when computing cut sets, the component-based analysis of [9] can handle cycles, as well.

6. RELATED WORK

There already exist methods for the hazard analysis of technical systems [1, 5, 11, 13, 15–17, 22].

Approaches that apply model checking for hazard analysis, are the works of Güdemann et al. [16] and Colvin et al. [5]. Due to state explosion, the method of [16] is only applicable with bounded model checking and can thus not show the absence of flaws. In contrast, our approach can handle the whole system, as our failure propagation graph are an abstraction of the system behavior.

The approaches of Palshikar [17] and Walker et al. [22] take the temporal ordering of errors into account. In contrast to our approach, the analysis yields minimal cut sequences, i.e. sequences of events that are necessary to cause an event, but no concrete time values.

Approaches that do consider concrete time values are the works of Colvin et al. [5], Abdelwahed et al. [1], Güdemann et al. [16], Grunske et al. [11] and Magott et al. [15]. In [15] fault trees are used, that encodes temporal properties in gates. They compute temporal dependencies between errors whereas we are computing propagation times. The approach of [5] performs a timed Failure Modes and Effects Analysis on timed Behavior Trees. In [11] State Event Fault Trees (SEFT) – Fault Trees combined with state charts – are used. The SEFTs are transformed to Deterministic and Stochastic Petri Nets which contain probability distributions over time for their transitions. The method of [16] is a formal approach for the fault tree analysis by model checking. All these methods allow statements about temporal properties of hazards. Though, they do not analyze propagation times.

The approaches of [17], [22], [5], and [15] do not consider reconfiguration. Approaches that analyze reconfigurable systems are the approaches of [1], [16], and [11]. But they all do not take complex structural rule-based reconfiguration into account. The approach of [11] analyzes each configuration individually, but not the changes themselves. In [16] reconfiguration is specified by state changes and invariants that the system has to satisfy. The method of [1] uses mode variables on their models to switch part of their models on and off. Unlike all these approaches we specify graph transformation rules that define the reconfigurations that can be executed. These reconfiguration rules enable us to analyze failure propagation during the process of reconfiguration.

7. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach for the timed hazard analysis of systems that reconfigure in order to react to errors in the system, so called self-healing systems. In order to include time information into our hazard analysis, we extended our failure propagation graphs with information about propagation times. Together with the reconfigurations rules that specify the reaction of the system to errors we analyze, whether the system's reaction to an error is fast enough to avoid a hazard.

In the future, we want to develop an automatic generation of timed failure propagation graphs from real-time statecharts – the behavior models of the MechatronicUML approach [8].

To make the analysis fully applicable, we plan to also take the delay, that exists between the trigger and the execution of the reconfiguration, into account.

We also plan to add probability distributions over time to our propagation time intervals, as the number and the width of these interval affects the uncertainty of the analysis. With the help of probability distributions we can estimated, e.g., the most probable propagation times of failures.

As said, the systems we consider reconfigure during runtime. Hence not all system configurations are known at design time. We consequently plan to implement a hazard analysis that is applied during run-time in order to only execute reconfiguration that lead to safe configurations with respect to hazard probability.

8. ACKNOWLEDGMENTS

This work was developed in the course of the Special Research Initiative 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

The authors thank Steffen Priesterjahn, Steffen Becker, and Steffen Ziegert for proofreading and helpful comments.

9. REFERENCES

[1] S. Abdelwahed, G. Karsai, M. Nagabhushan, and S. C. Ofsthun. Practical implementation of diagnosis systems using timed failure propagation graph models. *IEEE Transactions on instrumentation and measurement*, 58(2):240–247, 2009.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, 2004.

[3] F. Cassez and O.-H. Roux. Structural translation from time petri nets to timed automata. *Electron. Notes Theor. Comput. Sci.*, 128:145–160, May 2005.

[4] B. H. C. Cheng et al. Software engineering for self-adaptive systems: A research roadmap. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[5] R. Colvin, L. Grunske, and K. Winter. Timed behavior trees for failure mode and effects analysis of time-critical systems. *J. Syst. Softw.*, 81:2163–2182, December 2008.

[6] T. Eckardt, C. Heinzemann, S. Henkler, M. Hirsch, C. Priesterjahn, and W. Schäfer. Modeling and verifying dynamic communication structures based on graph transformations. In *Computer Science – Research and Development*. Springer, 2011. accepted.

[7] P. Fenelon, J. A. McDermid, M. Nicolson, and D. J. Pumfrey. Towards integrated safety analysis and design. *ACM SIGAPP Applied Computing Review*, 2(1):21–32, 1994.

[8] H. Giese, S. Henkler, M. Hirsch, V. Roubin, and M. Tichy. Modeling techniques for software-intensive systems. In D. P. F. Tiako, editor, *Designing Software-Intensive Systems: Methods and Principles*, pages 21–58. Langston University, OK, 2008.

[9] H. Giese and M. Tichy. Component-based hazard analysis: Optimal designs, product lines, and online-reconfiguration. In *Proc. of the 25th International Conference on Computer Safety, Security and Reliability, Gdansk, Poland*, 2006.

[10] O. M. Group. Uml 2.2 superstructure specification, 2009. Document – formal/09-02-02.

[11] L. Grunske, B. Kaiser, and Y. Papadopoulos. Model-driven safety evaluation with state-event-based component failure annotations. In G. T. Heineman, I. Crnkovic, H. W. Schmidt, J. A. Stafford, C. Szyperski, and K. Wallnau, editors, *Component-Based Software Engineering*, volume 3489 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin / Heidelberg, 2005.

[12] International Electrotechnical Commission, Geneva, Switzerland. *International Standard IEC 61025. Fault Tree Analysis (FTA)*, 1990.

[13] B. Kaiser, C. Gramlich, and M. Förster. State/event fault trees–A safety analysis model for software-controlled systems. *Reliability Engineering & System Safety*, 92(11):1521–1537, 2007.

[14] N. G. Leveson. *Safeware: System Safety and Computers*. ACM, New York, NY, USA, 1995.

[15] J. Magott and P. Skrobanek. A method of analysis of fault trees with time dependencies. In *Proceedings of the 19th International Conference on Computer Safety, Reliability and Security*, SAFECOMP '00, pages 176–186, London, UK, 2000. Springer-Verlag.

[16] F. Ortmeier, W. Reif, and G. Schellhorn. Deductive cause-consequence analysis. In *Proceedings of the 16th IFAC World Congress*, 2006.

[17] G. K. Palshikar. Temporal fault trees. *Information and Software Technology*, 44(3):137 – 150, 2002".

[18] C. Reutenauer. *The mathematics of Petri nets*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[19] G. Rozenberg. *HANDBOOK of GRAPH GRAMMARS and COMPUTING by GRAPH TRANSFORMATION, Volume 1: Foundations*. World Scientific, 1997.

[20] S. Simani, C. Fantuzzi, and R. J. Patton. *Model-based Fault Diagnosis in Dynamic Systems Using Identification Techniques*. Springer Berlin / Heidelberg, 2002.

[21] M. Tichy, S. Henkler, J. Holtmann, and S. Oberthür. Component story diagrams: A transformation language for component structures in mechatronic systems. In *Postproc. of the 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems (OMER 4), Paderborn, Germany*. HNI Verlagsschriftenreihe, 2008.

[22] M. Walker, L. Bottaci, and Y. Papadopoulos. Compositional temporal fault tree analysis. In F. Saglietti and N. Oster, editors, *SAFECOMP*, volume 4680 of *Lecture Notes in Computer Science*, pages 106–119. Springer, 2007.