Observations on Formal Safety Analysis in Practice

Michaela Huhn[a], Stefan Milius[b]

[a]*Institut für Informatik, Technische Universität Clausthal, Clausthal-Zellerfeld, Germany*
[b]*Institut für Theoretische Informatik, Technische Universität Braunschweig, Germany*

Abstract

We report on the application of formal verification in the safety analysis of two level crossing controllers that were industrially designed using SCADE Suite. Although the theoretical grounds for formalizing safety analysis have been developed in recent years, we faced numerous and intense complexity problems even with these medium size industrial case studies. The complexity problems constricted formal verification and even remained after employing different heuristics based on abstraction and introducing environmental models. In addition, we found that the modeling style has a significant impact on the complexity of the verification tasks. We finally succeeded to formally classify all relevant fault combinations as either critical or uncritical by identifying a crucial, design-specific liveness property.

Keywords: model-based development, SCADE, formal safety analysis

1. Introduction

A key issue in the development of safety-critical systems is safety analysis, i. e., a thorough analysis how components may fail and cause a system hazard. From the safety analysis, the safety requirements for components are derived and the design is proven correct with respect to functional safety in the verification and validation phase.

For developing software for safety-critical systems, formal methods are considered an adequate means because they provide correctness results at a level of strict mathematical rigor. Standards like the IEC 61508 part 3 [1] and its railway-specific derivative CENELEC 50128 [2] highly recommend them for the software

Email addresses: `Michaela.Huhn@tu-clausthal.de` (Michaela Huhn), `mail@stefan-milius.eu` (Stefan Milius)

development according to higher safety integrity levels (SIL). Nevertheless, seamless usage of formal methods is still a future challenge for industries. Frequently heard arguments trying to explain industries' indecision towards the proliferation of formal methods are a lack of integration and coverage in the development process and poor scalability for larger designs.

In this paper we report on our experiences on a seamless, tool supported, formal approach that covers design, safety analysis and formal verification: We compare two medium-sized design variants of a level crossing controller. Both were developed using the SCADE Suite[1] by a manufacturer for rail automation equipment. The two models are different in their modeling style, namely one is state-based and built based on safe state machines [3], whereas the other one is strictly data-flow oriented. Both implement the same functionality and have exactly the same interfaces, but the models were originally thought as a comparative basis to evaluate model qualities relevant for application developers such as understandability, maintainability etc. This was a purely design-oriented comparison that we have extended to safety analysis and formal verification.

So the starting point for our investigation is the question whether the modeling style has an impact on formal safety analysis and verification. By using the built-in state-of-the-art SAT-based model checker of SCADE we also strive for a good showcase for seamless formal support of the safety process. A precondition for an authentic showcase is to take the original design models without tailoring them beforehand to the needs of some particular formal analysis technique at hand.

For formal safety analysis, we apply *Deductive Cause-Consequence Analysis* (DCCA) by Ortmeier et al. [4] as a formal generalization of the well accepted safety analysis techniques FMEA (*Failure Mode and Effect Analysis*) and FTA (*Fault Tree Analysis*). Moreover, two major safety requirements are imposed by the safety analysis that have to be verified to show functional correctness. However, we have to cope with severe complexity problems even with these medium sized designs that prevent us from completing both, the DCCA and the verification of safety requirements. We try several abstraction techniques to reduce complexity of the verification problems within the SCADE Design Verifier, but without significant effect. We illustrate our attempts on data abstraction, cone of influence, and symmetry reduction as well as decomposition as well as on variants of the environmental model.

Finally, we come up with a weakening of the main safety property, namely a

[1]SCADE is a product of Esterel Technologies, see `www.esterel-technologies.com`.

design-specific restricted liveness property, which we use to prove all combinations of up to two faults as either critical or uncritical for the state-based model. Unfortunately, none of the approaches lead to similar results for the dataflow model. However, to be able to generalize this result, a clear and *application-oriented* notion of model elements with major impact on the verification complexity has to be developed.

In [5], our original publication on the comparative safety analysis of the level crossing controllers, we had to omit the details of safety analysis due to lack of space. More importantly, now we enriched the fault model and we choose a very different approach to finally prove at least the state-based model correct: In [5], we transformed the state-based model into UPPAAL timed automata[2]. The model transformation between the two formal frameworks, namely SCADE and UPPAAL, is methodically an *abstraction of time*: The multiple steps performed by the SCADE model to await a timeout are handled as one transition in the UPPAAL verification framework.

It turns out that the model transformation used in [5] is methodically more elegant, but the systematic mechanic testing of all fault combinations using SCADE Design Verifier revealed a further critical combination of faults overseen before. However, none of the approaches should be recorded as a complete success of formal verification in safety analysis but as a mandate for further research on how to improve usability: Currently the search space spanned by the possible combinations of heuristics employed to reduce the complexity of the verification tasks is definitely too large for industrial application. At the moment the intermediate feedback on the verification progress does not allow the user to estimate on the prospects of finally achieving a result since no hints indicating performance bottlenecks are provided. Moreover, we found that for slight variations of the fault combinations the verification time varies between 15 seconds and more than 23 hours. In the light of the numerous abstractions, environmental models and variants of the correctness property we tested it is true serendipity that we awaited the successful verification of the latter fault combination.

The differences between [5] and this paper are as follows: (1) In [5] the transformation from SCADE *to* UPPAAL *is presented as a method and in the case study, but is omitted here (in [5] Sec. 3.3 and 3.4). (2) The safety analysis is detailed w.r.t. the fault combinations and the underlying fault models and presented*

[2]UPPAAL is an integrated tool for modeling and verification of real-time systems, see `www.uppaal.com`.

here (see Sec. 4.2). (3) Now we succeed to complete the formal safety analysis up to 2 faults on the basis of a weakening of the correctness property (see Sec. 4.3) by investigating a set of appropriate environmental models. Sec. 5 is adapted to the findings of both approaches.

The paper is structured as follows: In Sec. 2 we recall safety analysis using FMEA and DCCA as a formal approach to it and SCADE suite as a model-based development environment featuring formal verification. Liveness analysis and a number of heuristics we have tried in order to deal with the intrinsic complexity problems are described in Sec. 3. In Sec. 4, safety analysis and verification results of the two model variants of a level crossing controller are presented in detail. Sec. 5 concludes with lessons learned.

2. Safety Analysis and DCCA

The development of safety-critical systems and their software is regulated by standards. In the railway domain the CENELEC standards EN 50126, 50128, and 50129 [6, 2, 7] apply. Since software is intangible, it is commonly agreed that system failures caused by software malfunction stem from systematic errors that are introduced during the software development and are not recognized in the safety process. Consequently, software safety engineering aims at (1) a complete and consistent specification of functional and safety aspects for a software component, (2) the correct implementation of the specification and (3) providing evidence that the safety objectives are met.

2.1. Safety Analysis

In the architectural design phase, the intended functionality is modularized. For safety-critical systems, a *hazard analysis* is performed to identify potential failures, hazards, and the causal chains between them. Classical inductive methods for hazard analysis are *Failure Mode and Effect Analysis* (FMEA) and its extension *Failure Mode, Effects and Criticality Analysis* as standardized in IEC 60812 [8]: FMEA classifies failures according to the severity of their effects, the occurrence frequency, and the detection rate. Starting from a definition of the system and its boundaries, a functional viewpoint is taken and each subfunction is analyzed with respect to potential fault modes. For functionality implemented in software, fault identification is supported by generic fault types like omission, commission, untimely reaction and value fault [9]. Local effects can be directly deduced from the component faults. In order to determine the system level effects, fault propagation is analyzed based on the functional architecture by using a formal deduction method (see Section 2.4). A fault is called *critical* if it has severe effects and an unacceptably high occurrence frequency. Safety measures are taken to eliminate the faults or at least mitigate the effects, to decrease their occurrence probability, or to actively detect them. Findings and actions to be taken are usually summarized in an FMEA table. Often a Fault Tree Analysis (FTA) is conducted to complement FMEA. FTA proceeds deductively by starting from potential hazards and then investigates which combination of faults or operational modes in the components may cause them.

FMEA is a structured but semi-formal technique. In an iterative design process, an FMEA is conducted and refined with each iteration cycle or change in the design or operational constraints. As FMEA is an inductive method, common cause faults and their effects are analyzed in a separate step.

2.2. Safety-Oriented Software Design

For the development of safety-related software in the rail domain, the standard CENELEC EN 50128 [2] prescribes the activities for design, validation, and verification with their input and output artifacts. As faults due to software are traced back to systematic development errors, the standard recommends measures that are considered appropriate to avoid such errors or to reveal them in a validation or verification step. Hence, development activities have to be performed with an adequate degree of rigor such that functional correctness and fulfillment of safety requirements can be proven with substantial evidence.

The SCADE tool suite (Safety-Critical Application Development Environment) is a model-based development framework for safety-critical software that supports certification due to EN 50128 up to SIL 3 and 4. The SCADE modeling language is a synchronous and dataflow-oriented language based on LUSTRE [10], and was extended by safe state machines [3]. SCADE provides certified automated code generation and immediate simulation of the model. It supports testing, in particular model coverage is recorded, and the SCADE Design Verifier (SCADE DV) allows for SAT-based formal verification[3].

Here, SCADE was chosen as design framework for the level crossing control from our industrial partner. The reason was the seamless design flow from the requirements via the model executing on a hardware abstraction layer (HAL) to the C code generated for the real target processor. Back then, validation, verification and safety assessment were performed with traditional methods. With a broader

[3]SCADE uses the SAT solver developed by Prover Technologies, see `www.prover.com`

usage of SCADE, different modeling styles came up and also the desire to benefit from the Design Verifier.

2.3. Formal Verification Using SCADE *DV*

The behavior of a SCADE design model can be given as a transition system on which SAT-based model checking can be performed in order to verify reachability properties[4], see [11].

In contrast to other approaches, the SCADE DV does not offer a temporal logic but properties have to be modeled as *synchronous observers* [12] using the same language operators as for the design. Nevertheless, we will use CTL as a succinct notation for reachability properties to be verified, but the reader should keep in mind that they are encoded as observer nodes at the SCADE level (cf. Figures 1 and 4). Notice, however, that more general temporal properties, notably unbounded liveness, cannot be automatically verified using this SAT-based approach.

2.4. Deductive Cause Consequence Analysis

An important step in safety analysis concerns determining the causal relationship between component fault modes and hazards. There are various techniques that exploit formal methods, notably model checking, for identifying the desired cause-consequence relation [11, 13, 14]. Here we consider DCCA by Ortmeier et al. [4, 15], which is a formal approach to safety analysis that generalizes FTA and FMEA. In DCCA, a hazard H is specified as a state predicate. Primary component fault modes are modeled by adding simple fault automata that are triggered by a Boolean input indicating the occurrence of this particular fault. The immediate effect of a fault on a component is specified in a so-called fault node within the component model. The hazard is implemented as an observer node that takes signals from the system and evaluates them according to the *negation* of the hazard predicate H, i.e., returning false whenever the hazard occurs. Figure 1 gives a schematic picture of the integration of DCCA with SCADE.

A core notion of FTA is that of a *critical cut set* of faults for a hazard, i.e. a subset of primary faults for which some operational condition and occurrence sequence exist such that the hazard occurs without further influence. This was

[4]In the area of model checking, reachability properties are often called *safety properties*, because they express that the system always stays in a good or "safe" state. In contrast, for us a safety property is any constraint to ensure dependable behaviour.

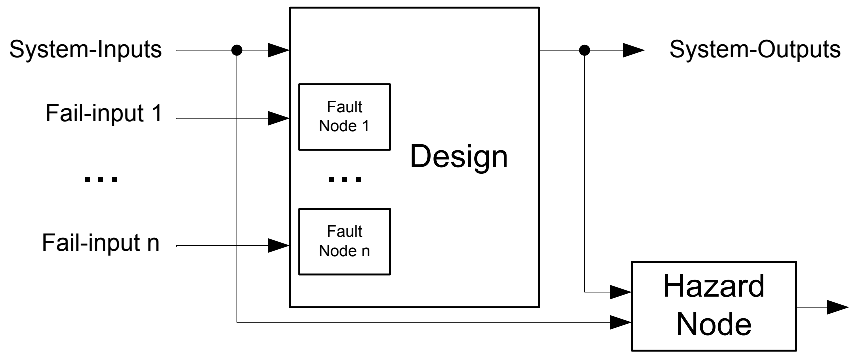

Figure 1: Integration of DCCA with SCADE

formalized in DCCA to the notion of a *critical set*. According to [15], critical sets can be formalized for the SCADE semantics as follows:

Definition 2.1. Let Γ be a subset of the set of primary component faults Δ of a system Sys. $\mathsf{Sys}\rangle_{\overline{\Gamma}}$ denotes that behavior of Sys where no fault from $\Delta \setminus \Gamma$ ever occurs. Now Γ is called *critical* for a hazard described by the state predicate H iff $\mathsf{Sys}\rangle_{\overline{\Gamma}} \not\models \mathbf{AG}\,\neg H$. A critical set Γ is called *minimal* iff no proper subset of Γ is critical.

If the analysis reveals a singleton as minimal critical set, this indicates that there exists a single-point-of-failure in the system, which has to be eliminated by further safety measures. A DCCA is called *complete* iff all minimal critical sets have been identified.

Theorem 2.2 (Minimal-Critical-Set Theorem [4, 15])**.** *If a complete DCCA has been conducted for a hazard H then for each minimal critical set, preventing* one *fault from occurrence will prevent the hazard H.*

In order to determine the minimal critical sets we use an iterative approach similar to [11]: Suppose that Δ is the set of all component fault modes identified during the FMEA and that H is a hazard. We iterate over all subsets $\Gamma \subseteq \Delta$ starting with $\Gamma = \emptyset$ and increasing Γ stepwise (singletons, doubletons etc.). We use the SCADE DV to prove whether Γ is critical for H: all trigger inputs for fault modes in Γ become system inputs, and the inputs of all other fault mode nodes are constantly false. Whenever SCADE DV returns a counterexample, i. e. a situation in which the hazard node expressing $\neg H$ is false, we have identified a minimal critical set Γ, and due to the monotonicity of criticality no super set of Γ needs to be considered.

3. Liveness and Abstraction Techniques

In this section we describe the general steps we took in our case study. Here we assume that we are given a design model of the system Sys in a formalism providing support for formal verification. Our steps were as follows:

(1) Given the system model we performed a safety analysis identifying hazards and component fault modes, cf. Section 2.1.

(2) We used DCCA to determine the cause-consequence relationship between component fault modes and hazards. For this we use the SCADE DV to obtain minimal critical sets of fault modes, cf. Section 2.4.

(3) We performed an analysis of a liveness property arising from the safety requirements of the system.

(4) We applied several strategies in order to deal with complexity issues arising in connection with the model checking performed in steps (2) and (3).

We already explained methodological details of steps (1) and (2) in Section 2. In the remainder of this section we will describe steps (3) and (4) in more detail.

3.1. Liveness Analysis

This step of our case study concerns the analysis of a safety related liveness property derived from a system requirement, see Section 4.3 for the concrete formulation. Roughly, this requirement states that a certain non-safe state of the system may occur, but this state must not be permanent, and it must be left within a certain time interval that depends on the system's configuration. Indeed, our analysis in step (2) revealed that this non-safe state of the system will occur under a certain fault mode. This is still in accordance with the requirement. However, it has to be verified that the system will leave this non-safe state in time.

If φ is a propositional formula describing the non-safe state then this property might be formulated in a temporal logic like CTL as $\mathsf{AG}(\varphi \rightarrow \mathsf{AF}\neg\varphi)$. However, SCADE does not provide a direct way to express a liveness property with an unbounded quantifier like AF. A concrete time limit – and hence a bound for the number of cycles – is not specified, but it is individually set for each configuration.

Thus, we explore the model and try to establish a lower bound on the number n of cycles after which the non-safe state is left: It is clear that once in the non-safe state the system will remain there for at least one cycle, and so we verify whether the non-safe state is left after $n = 2, 3, \ldots$. This can be modeled as a SCADE observer by using the operator ImpliesWithinNTicks from the design verifier library, see Figure 2 for $n = 4$.

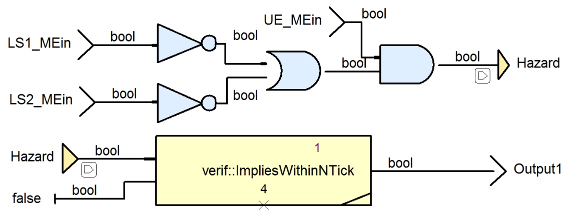

Figure 2: Observer node for the liveness analysis with $n = 4$.

3.2. Dealing with Complexity within SCADE *DV*

Unfortunately, the direct approaches to complete our analysis by just using SCADE DV failed even with small numbers n due to complexity problems, see Section 4 for details. We will now mention the strategies we have applied in order to deal with this problem.

Abstraction. First we generated an abstract version of the design model in SCADE. For this we applied three different techniques: data abstraction, cone of influence reduction and symmetry reduction (see e. g. [16] for an introduction) – the way in which these are applied is detailed in Section 4. However, even for the abstracted model the bounded liveness verification could not be completed for the data-flow model using SCADE DV.

4. Comparative Safety Analysis of Level Crossing Control

Now we present the results of our industrial case study. Our formal safety analysis was performed comparatively for two SCADE models for the modular level crossing controller in [18]. The two SCADE models were developed by different developers with different implementation styles: the first model uses a design based on safe state machines (SSM) [3] and the second one uses data flow diagrams, which are essentially graphical representations of LUSTRE [10]. Whenever information has to be stored for the next execution cycle in the data-flow oriented model, this is done within local variables for which the value is kept using the fby-operator.

Both models implement the same architecture with the following operators, which can be composed in order to obtain a level crossing control logic for a specific level crossing layout:

- *route controller* (LC_Route)
- *site controller* (LC_Site)

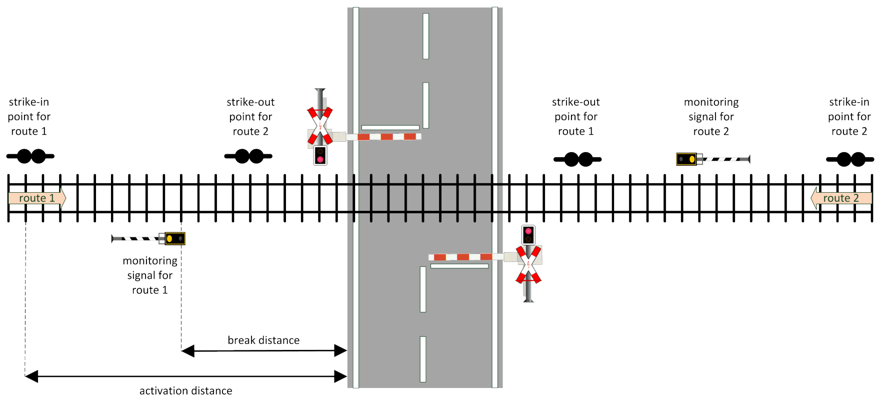
strike-in point for route 1
strike-out point for route 2
strike-out point for route 1
monitoring signal for route 2
strike-in point for route 2
route 1
route 2
monitoring signal for route 1
break distance
activation distance

Figure 3: Sample layout of a locally monitored level crossing

- *group controller* (LB_Group)
- *time controller* (LC_Timer)

The route controller monitors the activation of the lights and barrier groups depending on the activation signal from a particular route, cf. Figure 3. The site controller synchronizes all route controllers and group controllers and acts as a logical connector. The group controller controls the lights and barriers, which are grouped logically, using a hardware abstraction layer. Finally, the time controller monitors the time elapsed since the last activation of the LC until its complete deactivation. Depending on the level crossing layout, numbers of inputs, outputs and nodes in the models can vary. In our case, models are specified for controlling of a level crossing with 2 routes and 2 lights-barrier groups, see Figure 3. They have 18 boolean, 1 integer input variables, 5 boolean output variables and 14 constants. The models are composed of 5 nodes: 2 LC_Route, 2 LB_Group, 1 LC_Site. Specifically, the first model has 12 states + 23 transitions for each LC_Route operator, 14 states + 23 transitions for each LC_Site operator, 27 states + 51 transitions for each LB_Group operator. In both models, the LC_Timer operator has 2 states. The main operators of both models are shown in Figures A.6 and B.7 in the appendix.

4.1. Application of FMEA

The initial steps of an FMEA are the description of the system, definition of its boundaries and the identification of the system functions. As shown in Figure 3, the level crossing (LC) system consists of a software controller and a hardware group, which includes two monitoring signals (MS), two groups of road-side protection elements: two barriers and two warning lights.

The main system function is the protection of the level crossing which can be divided further as follows:

- Track-side protection of the level crossing: The monitoring signal reflects the status of the road-side securing elements, i.e. the lights and barriers. Only if the lights and barriers signal protection is completed, the monitoring signal will be activated, otherwise it is switched off.
- Road-side protection of the level crossing: While a train is approaching and passing the level crossing, barriers must be closed and warning lights must show the halting signal, which usually is a red light.

Using the definition of fault modes from IEC 60812 [8], we deduced the relationship between hazardous situations and the main system function. Fault modes were inspected within the operational context and each fault mode was explained with suitable scenarios. Table 1 shows the summary of the initial analysis step.

Function fault	Scenario	Consequence
LC protection is not executed.	MS shows that LC is protected although it is not; the lights are off, barriers are not closed, despite of an approaching train.	Train drives through a unprotected LC.
LC protection fails to operate in time	Warning lights are not activated within the prescribed time; barriers are activated.	Drivers or pedestrians drive against the closed barriers.
LC protection does not cease in time.	(1) LC is prematurely deactivated. (2) Warning lights stay red and barriers stay closed.	(1) Train drives through a unprotected LC. (2) Road-side traffic waits, although there is no train.
LC protection is spontaneously activated.	LC is spontaneously activated.	Road-side traffic waits, although there is no train.

Table 1: Functional FMEA summary

Based on this initial analysis we identified two hazardous consequences as follows:

(1) train drives through an unprotected level crossing, and

(2) drivers or pedestrians drive against the closed barriers.

The first one is the consequence of an incorrect signal of the monitoring signal which indicates the LC as protected although it is not. The second one is also hazardous, however, human error or technical faults that originate outside of the system contribute to this hazard as well. Thus, we restrict the focus to the train and track-side faults for the remaining analysis, and faulty behavior of road users and its possible consequences are analyzed no further.

In addition, we used some generic hardware fault modes from IEC 60812 and associated them with their functional consequences. Table 2 shows the generic hardware fault modes, their occurrence patterns and the hardware types on which they were applied in the analysis. Since the system contains each hardware component (barrier, light and monitoring signal) twice and the two components can fail independently from one another each fault needs to be assigned two corresponding identifiers. Notice that by a "false detection" of the barrier we mean that the barrier feeds back its status as closed (while it might be open) and "misdetection" means that the barrier feeds back its status as closed.

The hazard of an unprotected LC may occur, if the monitoring signal does not represent the actual status of the LC while the train is approaching. In a regular operation without any faults, the LC control logic shall ensure the correct order and synchronization of the activation and deactivation of a monitoring signal w.r.t. the LBGs. Hence in the absence of faults, the monitoring signal may only be activated after the complete switch-on of both LBGs and it will be deactivated before the switch-off of any LBG element. As activation is modelled by signals (i. e. input $\texttt{UE\_MEin} = \mathsf{true}$ for the first monitoring signal, and inputs $\texttt{LS1\_MEin} = \mathsf{true}$ and $\texttt{LS2\_MEin} = \mathsf{true}$ for both lights and barriers), this property can be formally expressed with the following CTL formula:

$$\mathsf{AG}(\texttt{UE\_MEin} \Rightarrow (\texttt{LS1\_MEin} \wedge \texttt{LS2\_MEin})).$$

In the SCADE framework, the hazard node is a negated node, that means that its output is false in the presence of the corresponding hazard. As explained above, the track-side hazard "*a train drives through an unprotected level crossing*" occurs if and only if the monitoring signal is switched on in spite of at least one non-active LBG. This state can be represented as a propositional formula H as follows:

$$H = (\texttt{UE\_MEin} \wedge \neg(\texttt{LS1\_MEin} \wedge \texttt{LS2\_MEin})) \tag{1}$$

Figure 4 shows the SCADE observer node for the hazard H from (1).

4.2. Application of DCCA

To apply DCCA to our two models we created a hardware model in SCADE simulating stimuli from the hardware of a real level crossing system. Then we

Type	Ids	Explanation	Pattern
Lights	L1, L2	unwanted switch off of traffic light	Persistent
	L3, L4	traffic light fails to switch on	Transient
	L5, L6	traffic light fails to switch off	Transient
Barriers	B7, B8	barrier is stuck	Persistent
	B9, B10	unwanted switch on of barrier	Transient
Barrier Sensors	BS11, BS12	false detection of barrier sensor	Transient
	BS13, BS14	misdetection of barrier sensor	Transient
Monitoring Signals	MS15, MS16	unwanted switch off of monitoring signal	Persistent
	MS17, MS18	monitoring signal fails to switch on	Transient
	MS19, MS20	monitoring signal fails to switch off	Transient

Table 2: Component fault modes grouped by hardware type

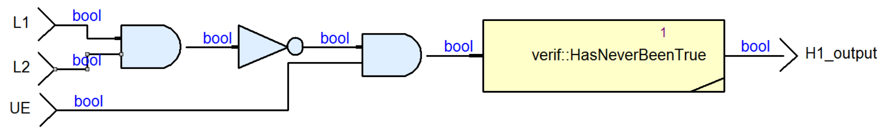

Figure 4: SCADE model of the hazardous event H

extended the hardware model with the fault modes and their occurrence patterns from the FMEA. Finally, we connected the SCADE model of the hazardous event as a synchronous observer to the models composed from the level crossing model and the hardware model. Figure 5 shows the resulting SCADE design with the level crossing controller LC_Main in the upper half and the hardware model `Sim::HAL` in the lower half. The synchronous observer `Proof::H1` is in the lower right-hand corner, and notice that all but one fault modes are disabled (see the inputs of `Sim::HAL`).

Next we need an environment model that constrains the external inputs of the level crossing in a sensible way. Otherwise any formal verification might produce counterexamples that rely on sequences of stimuli that are (physically) impossible in the operation of a level crossing system. The necessity of a realistic environment model in connection with formal verification in SCADE DV is also discussed in [19].

Our level crossing model has one integer input `T_System` and 18 Boolean inputs. The integer input supplies information about the progress of time to the model in form of a cycle count, hence its input flow must be $0, 1, 2, 3, \ldots$, which is modeled by a simple counter (in the upper left-hand corner of Figure 5). 16 of the Boolean inputs take constant values because they serve as configuration variables or correspond to special modes of operation of the level crossing system (e. g. if the level crossing is permanently switched on by maintenance personnel) that are not part of our analysis. The constant values are expressed by SCADE *assumptions*, e. g. `EX1_VEin = true` (see the red assignments on the left-hand side in Figure 5).

The remaining two Boolean inputs `EX1_SEin` and `EX2_SEin` represent the state of occupation by a train of the two routes (from left to right and vice versa) of the level crossing; effectively, a change from `false` to `true` (or vice versa) of these inputs is a command to switch on (off) the level crossing. Since our level crossing has only one track it makes sense to keep `EX2_SEin` constantly false (again, by a SCADE assumption). For `EX1_SEin` we subsequently consider three very simply environmental models:

(1) `EX1_SEin` is constantly `true`; this corresponds to a switch on command for the first route of the level crossing (with no subsequent switch off command),

(2) `EX1_SEin` is left open; any Boolean flow is allowed as an input sequence, and

(3) `EX1_SEin` is set to `true` for 40 cycles and then false forever; this roughly

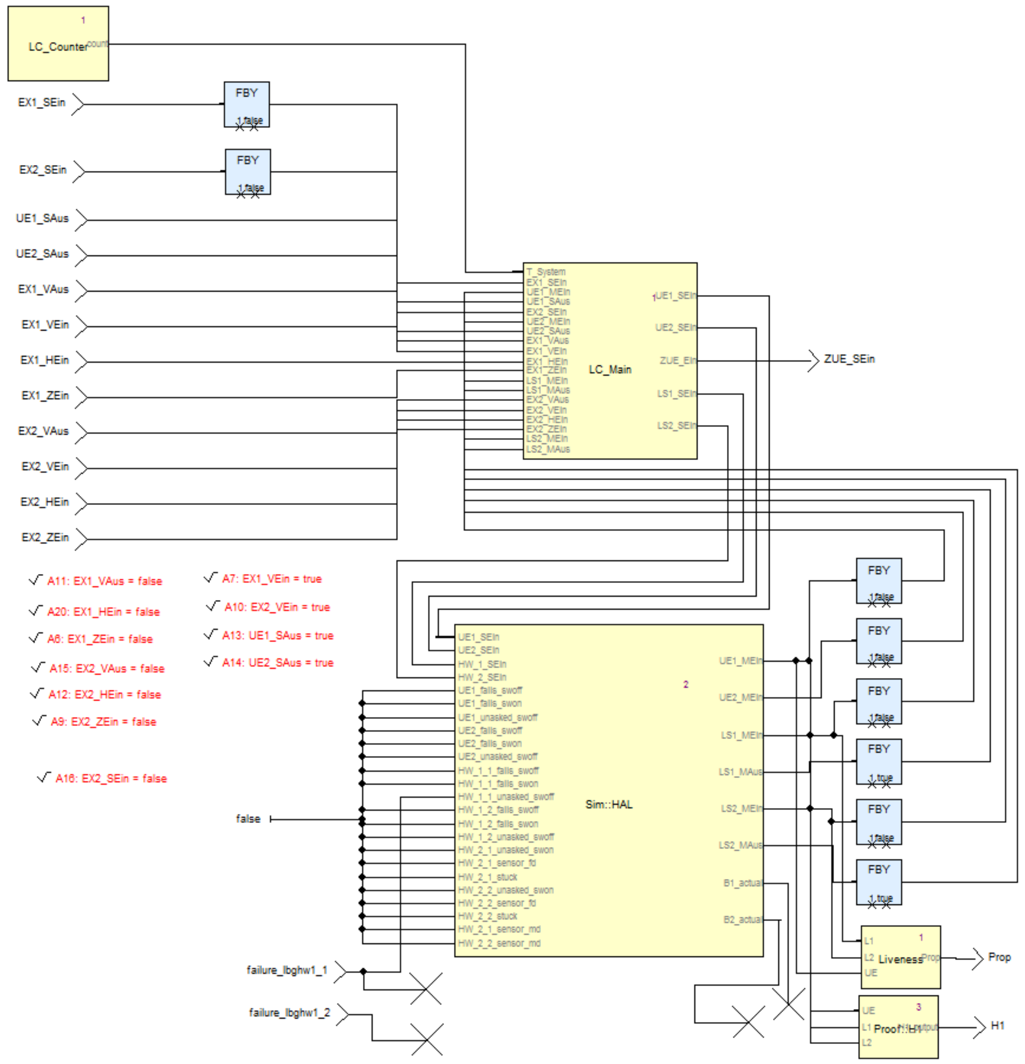

Figure 5: Level crossing control with hardware model and synchronous observer nodes

corresponds to one train passing through the level crossing from left to right.

Notice that the second environmental model can produce unrealistic counterexamples since the environment of the real level crossing system ensures that the inputs `EX1_SEin` and `EX2_SEin` do not sporadically oscillate but always hold their value for a certain interval after any change (as modeled in environmental model (3)).

For each of our subsequent formal verifications we indicate which of those three environmental models were used. All "valid" verification result are obtained with the proof strategy of SCADE DV and all "falsifiable" results with their corresponding counterexamples with the debug strategy.

Functional correctness. Recall from Section 2.4 that the first step of DCCA considers the empty set of fault modes. So we verify functional correctness of the models w.r.t. the safety requirement expressed by the hazard H. With SCADE DV and using environment model (1), the proof took 12 seconds for the state based model and 81 seconds for the data flow one. However, with an open input `EX1_SEin` (environment model (2)) we experienced complexity problems for both models. For example, for the SSM based model it was not possible to obtain a counterexample by unrolling the model until depth 42 using the debug strategy of SCADE DV, and using the proof strategy no result was obtained until depth 23 with 3 days of running time. It is interesting to note that memory consumption does not increase as the verification proceeds. This indicates that the problem with the verification does not originate from a state explosion but from algorithmic difficulties. For environment model (3) the proof of functional correctness took 82914 second (i. e., more than 22 hours).

For the dataflow model with environment model (2) a counterexample was produced after 702 seconds.

The difference in the results for the two models originates from the different implementation styles.

Single fault modes. We checked criticality of singleton fault mode sets Γ w.r.t. the hazard node H using the environment model (1)—because of the running times we saw with the other two environment models for functional correctness the time constraints of our project forbid their use in connection with the hazard node H.

In order to perform the analysis we connect the input of the SCADE node Sim::HAL corresponding to the fault mode to a Boolean input, see Fig. 5.

Again, we start by considering environment model (1), i. e. we analyze criticality of the activation of the level crossing system. Notice that because of sym-

	State-based Model			**Data-flow Model**		
Fault mode	Valid	Critical	Time	Valid	Critical	Time
L1		x	1		x	1
L3	x		95	x		99
L5	x		10	x		89
B7	x		11	x		105
B9			-			-
BS11	x		122	x		86
BS13		x	1		x	1
M15	x		11	x		94
M17	x		83	x		177
M19	x		11	x		91

Table 3: Singleton fault modes

metry of the system model for lights and barriers we only need to consider the fault modes L1, L3, L5, B7, B9, BS11, and BS13. Moreover, only the monitoring signal associated to the route that triggers activation of the level crossing needs to be considered, i. e., we consider the fault modes M15, M17 and M19.

Table 3 shows the results and proof execution times in seconds. For both models L1 and BS13 are identified as critical. For B9 no result could be obtained within reasonable time for either model. For the SSM based model no counterexample could be obtained by unrolling the model until depth 123 using the debug strategy of SCADE DV, and using the proof strategy SCADE DV did not succeed after more than 3 days of running time and unrolling the model up to depth 46. Similarly, for the dataflow based model. The remaining fault modes could be proven to be non-critical.

At most 2 fault modes. This step requires analysis of the fault mode sets Γ with $|\Gamma| = 2$. Tables 4 and 5 present the analysis results for the SSM based and dataflow model, respectively. As L1 and BS13 are already critical, we only analyze sets Γ with L1, BS13 and the symmetrical L2, BS14 not contained in Γ (i. e., the grey fields in the tables are not investigated). In addition we did not investigate fault mode sets Γ containing B9 and the symmetric B10 because we expect the complexity problems from the previous analysis step to persist (the yellow fields in the tables). Mutually exclusive fault modes need not be investigated as indicated by the blue fields, and symmetric cases are indicated in light grey. Finally, we assume that, at each clock cycle, sensor failures (BS11–BS14) can occur either as a false detection or a mis-detection, but not both.

Notice that for the fault mode combination L3, BS11 no result could be ob-

	L1	L2	L3	L4	L5	L6	B7	B8	B9	B10	BS11	BS12	BS13	BS14	M15	M16	M17	M18	M19	M20
L1	x																			
L2		x																		
L3			x																	
L4			101	x																
L5			🟦		x															
L6			98		11	x														
B7			115		12		x													
B8			104		11		12	x												
B9			🟨	🟨	🟨	🟨	🟨	🟨	x											
B10			🟨	🟨	🟨	🟨	🟨	🟨	🟨	x										
BS11			🟨		112		11		🟨	🟨	x									
BS12			107		100		139		🟨	🟨	107	x								
BS13													x							
BS14														x						
M15			105		15		11		🟨	🟨	144				x					
M16			100		14		12		🟨	🟨	124				146	x				
M17			100		71		77		🟨	🟨	127				🟦	73	x			
M18			99		11		11		🟨	🟨	154				12		74	x		
M19			97		11		12		🟨	🟨	122				🟦	12	🟦	71	x	
M20			97		10		11		🟨	🟨	11				11		10		10	x

Table 4: Analysis results for doubleton fault mode sets for the SSM based model

	L1	L2	L3	L4	L5	L6	B7	B8	B9	B10	BS11	BS12	BS13	BS14	M15	M16	M17	M18	M19	M20
L1	x																			
L2		x																		
L3			x																	
L4			95	x																
L5			🟦		x															
L6			105		91	x														
B7			105		106		x													
B8			112		103		121	x												
B9			🟨	🟨	🟨	🟨	🟨	🟨	x											
B10			🟨	🟨	🟨	🟨	🟨	🟨	🟨	x										
BS11			🟨		93		2		🟨	🟨	x									
BS12			112		103		188		🟨	🟨	211	x								
BS13													x							
BS14														x						
M15			111		101		112		🟨	🟨	100				x					
M16			125		105		120		🟨	🟨	181				247	x				
M17			242		181		188		🟨	🟨	162				🟦	675	x			
M18			192		179		187		🟨	🟨	209				191		357	x		
M19			109		92		101		🟨	🟨	95				🟦	179	🟦	176	x	
M20			110		92		109		🟨	🟨	82				195		809		188	x

Table 5: Analysis results for doubleton fault mode sets for the data flow model

tained due to similar complexity issues as for B9 in the previous step. For all other investigated combinations, results are obtained within reasonable computation times.

Since three simultaneous failures are highly unlikely we terminated the analysis with this step. To summarize, while our analysis remains incomplete we have identified the following minimal critical sets: {L1}, {L2}, {B9}, {B10}, {B7, BS11}, {B8, BS11}.

4.3. Liveness analysis

We have shown that both models do have critical sets of fault modes. From a control engineering point of view this was expected to happen, and rather than requiring prevention of the hazard the corresponding safety requirement for the system reads as follows:

Requirement. *A state in which the monitoring equipment feedbacks its status as activated* ($\texttt{UE\_MEin} = \mathsf{true}$) *and at least one LBG is non-active* ($\texttt{LS1\_MEin} = \mathsf{false}$ *or* $\texttt{LS2\_MEin} = \mathsf{false}$) *is non-safe. Such a state must not be permanent and has to be left independently from the input values as quickly as possible by deactivation of the monitoring signal.*

This requirement can be formalized as a liveness property in CTL as follows:

$$\mathsf{AG}(\texttt{UE\_MEin} \wedge \neg(\texttt{LS1\_MEin} \wedge \texttt{LS2\_MEin}) \Rightarrow \mathsf{AF}(\neg\texttt{UE\_MEin})) \qquad (2)$$

As explained in Section 3.3 we used the SCADE DV with an observer as shown in Figure 2 to obtain a lower bound on the number n of cycles the system remains in the non-safe state corresponding to our hazard property H.

In this analysis we investigate the singleton fault modes L1, and BS13 as well as the combinations B7, BS11, and we use the environment model (1) (i. e., input $\texttt{EX1\_SEin}$ is constantly $\texttt{true}$). Clearly, once the hazard occurs it will last for at least one cycle. So we investigate the liveness property with $n = 2, 3, \ldots$. For the SSM based model and $n = 2, \ldots, 6$, SCADE DV delivered "falsifiable" as an answer within a few seconds. But for $n = 7$ the corresponding property is "valid". This reflects the reasonable property that after an unwanted switch off of a traffic light the level crossing control software must be allowed some reaction time before it issues a command to shut off the monitoring signal.

Similarly, for the dataflow model we obtain "falsifiable" for $n = 2, 3$ and "valid" for $n = 4$. The running times are displayed in Table 6. Notice that we also performed the same analysis for B9 and the combination L3, BS11 for which

Fault modes	L1	B9	BS13	L3, BS11	B7, BS11
SSM based model	343	351	271	105	329
Dataflow model	510	621	515	171	500

Table 6: Results for liveness analysis of activation of the level crossing

Fault mode	Valid	Critical	Time
L1	x		1548
L3	x		1359
L5	x		1558
B7	x		1566
B9	x		1759
BS11	x		1750
BS13	x		1439
M15	x		1593
M17	x		1627
M19		x	40440

Table 7: Analysis results for singleton liveness analysis for the SSM based model model

we previously experienced complexity problems. These results seem to indicate that the weakening of our verification property from the hazard node in Figure 4 to the liveness property in Figure 2 somewhat mitigates our previous complexity problems.

Indeed, as a next step we tested if this is also the case when we consider the other two environment models (2) and (3). We start this analysis with the most liberal environment model (2) (i. e., input `EX1_SEin` is left open). As the two models perform very differently, we consider them separately. First we proceed to analysing the SSM based model.

4.3.1. Analysis of SSM based model

In the first step we verified functional correctness (with no fault mode enabled) w.r.t. the synchronous observer in Figure 2 (it appears as the node called "Liveness" at right-hand lower corner of Figure 5); this analysis took 1611 seconds with SCADE DV.

For single fault modes the results are shown in Table 7. As expected L1 and BS13 are no longer critical, but M19 suddenly is. However, it is not a big surprise that this was not discovered in the previous analysis: recall that M19 means that the monitoring signal fails to switch off, and this failure has no effect for the simple environment model where `EX1_SEin` is constantly `true` since the monitoring signal is never switched off if a train does not leave the level crossing.

A manual analysis of the counterexample that SCADE DV produces in the case of M19 shows that it is possible to produce from this a test case that will violate our liveness property from Figure 2 for every number n; so M19 is discovered as a critical fault w.r.t. the unconstrained environment (2). However, this environment is not realistic. In fact, it will always be ensured by the system issuing the switch on command for the level crossing on `EX1_SEin` that the value `true` is held as long as the level crossing is occupied by a train. So we reran the analysis with environment model (3), but this lead to complexity problems. We then simplified the environment by making the enable signal for M19 constantly `true`. This is reasonable in connection with environment (3) since the monitoring signal will only be switched off once in this case and the input for triggering M19 will simply let the hardware model stay in the "on" state. So having the enable signal constantly `true` is a conservative over-approximation to leaving it open.

With the environment thus simplified we verified within 10562 seconds that M19 is uncritical.

Let us now proceed to the analysis of doubleton fault modes w.r.t. the liveness property. The analysis results are displayed in Table 8. Again, for this analysis we used an open input `EX1_SEin` except for fault mode combinations containing M19 where we used environment model (3) and the enable signal for M19 constantly `true`. In addition for some of those fault mode combinations we could only obtain a result by also making the input triggering the second fault mode constantly `true` (those result are displayed with a dark green background). This is no problem for the respective fault modes except for B7 (i. e., when the barrier is stuck). A barrier can be stuck in the open or closed position and so we have only verified the first of those two possibilities. With another simple input pattern for B7 (22 cycles `true` and then `false` forever) one can enable this fault such that the barrier is stuck in closed position. This can also be verified as uncritical within 8950 seconds.

Finally, notice that the two fault mode combinations L1, M19 and BS13, M19 are identified as critical, and analysis of the corresponding counterexamples reveals that they will remain so for every number n considered in the liveness property from Figure 2.

4.3.2. Liveness of the data-flow model

The experiments using the environmental model (1) indicate that the design specific constant for the liveness property in Figure 2 on p. 9 is 4 for the data-

	L1	L2	L3	L4	L5	L6	B7	B8	B9	B10	BS11	BS12	BS13	BS14	M15	M16	M17	M18	M19	M20
L1	x																			
L2	1435	x																		
L3	4382		x																	
L4	1539		1324	x																
L5	1424		🟦		x															
L6	1379		1436		1400	x														
B7	1363		1299		1435		x													
B8	1484		1333		1447		1505	x												
B9	1380		1653		1394		1484		x											
B10	1492		1328		1676		1408		1528	x										
BS11	1425		1469		1714		1376		2183		x									
BS12	1396		1366		1554		1545		1570		1520	x								
BS13	1452		1430		1600		1484		1425		1581		x							
BS14	1553		1560		1587		1433		1516		1508		1494	x						
M15	1501		1476		1580		1608		1483		1555		1455		x					
M16	1535		1437		1396		1554		1629		1377		1586		1424	x				
M17	1659		1605		1527		1580		1508		1693		1674		🟦	1505	x			
M18	1355		1403		1472		1939		1648		1473		1442		1545	🟦	1540	x		
M19	9		15		7070		15		5323		7478		8		🟦	7729	🟦	7306	x	
M20	1511		1387		1535		1377		1562		1441		1458		1540	🟦	1622	🟦	8333	x

Table 8: Analysis results for doubleton liveness analysis for the SSM based model

flow model. As shown in Table 6 we could complete the safety analysis also for the data flow model: All combinations of fault modes that do not satisfy the original hazard property (see Table 3 and 5) were proven uncritical with respect to the weaker bounded liveness property. This completes the formal proof that the data-flow model handles the switch-on of the level crossing in a correct and safe manner even in the presence of up to two faults.

However, our attempts to extend this result to the environmental models (2) and (3) were not successful. As in [5] we tried to apply abstraction techniques:

Cone of Influence Reduction. Only 7 inputs out of 19 of the model (see Figure A.6) have an influence on the observer node. Some of the remaining inputs have constant values as they correspond to configuration settings. These inputs can be replaced by the constants. As a consequence some transition triggers are simplified or even become constantly false, leading to further simplifications.

Symmetry Reduction. Here we remove the identical operators and symmetric configurations in order to reduce the state space. Our initial configuration was a 2 Route+2 LBG level crossing. Obviously, a reduced model (1 Route+1 LBG) needs to satisfy the liveness property, too.

We amply tried to verify the abstracted data-flow model with both environmental models thereby also varying the bound. However, verification did not terminate within days, but SCADE DV continued to unroll the model step by step without finding a suitable induction hypothesis. Thus, safety analysis remains incomplete for the data-flow model.

This concludes our safety analysis of the level crossing model. Agreeing with the supplier and the quantitative safety analysis of the hardware components, we argue that the analysis up to combinations of two faults suffices: The likelyhood of three or more simultaneous faults within the fault disclosure interval[5] is that small that bigger fault sets can safely be neglected.

5. Lessons Learned and Conclusions

We performed a comparative case study on formal safety analysis and verification. Our starting point were two functionally equivalent SCADE models of a level crossing controller software, both developed in industry. With SCADE a formally founded, model-based approach had been in use already. Thus, it was straightforward to seamlessly expand this approach towards formal safety analysis and verification within the same tool environment. Despite of the difficulties we faced and which fill major parts of our experience report, we were able (1) to identify the most relevant critical sets of fault modes as well as (2) to verify major safety requirements. In addition, confidence is strong that the results of the formal verification apply for the finally generated executable, since the target code is automatically generated from the SCADE models by the certified code generator. Such a full integration and tool support for design, code generation and verification is mandatory for the successful application of formal methods in industrial development of safety-critical systems. In this way formal methods can contribute with strong evidence to a safety case which a system manufacturer has to provide to certification authorities.

5.1. Lessons Learned

Modelling Style and Verification. Our verification results strongly support the hypothesis that applicability of formal analysis based on SAT model checkers depends on the design and modeling style. We found that the state-based model lends itself better to abstraction and reduction techniques than the data-flow oriented one. Another argument pro state-based designs is that they facilitate trans-

[5]The *fault disclosure interval* defines the maximal time that may elapse until the system discloses a critical fault. To guarantee the fault disclosure interval the system will regularly test critical components.

formation to other model checkers as we did in [5]. For data-flow oriented designs some abstraction techniques do not seem obviously applicable.

However, the size and complexity of both models are similar and there is no straightforward explanation why the two modeling styles differ w.r.t. verification. But in the data-flow design, the state information is scattered throughout the model and less uniform than in the state-based one. That may be a reason why the model transformation to SCADE DV's SAT-based model checker yields a better result for the state-based model.

Complexity problems. The eminent observation from our case study is that for both moderately sized industrial models we faced severe complexity problems in the DCCA, but in the verification of functional correctness, too. Verification results could be achieved only after rescaling the timers in order to decrease the number of cycles that had to be inspected by SCADE DV to a minimum. In addition, we had to restrict the analysis to one route, i.e. the possible interferences of concurrent route activations are still not investigated. Under these constraints, we yield a good set of verification results proving both models correct and safe for the activation procedure of the level crossing (see Table 3, 4, and 5). Although we tried very hard to extend safety analysis for the original hazard property, we only got very few results, even though we did not obtain further counterexamples.

But the numerous attempts let us discover an admissible weakening of the original safety requirement, namely a bounded liveness property stating that the level crossing controller will stay in the hazardous condition for a limited number of cycles that is definitely neglectable. A hint to this liveness property is given already in the initial specification as ”An unsafe state must be left as soon as possible”. Then the safety analysis could be completed as shown in Table 7 and 8 for the state-based model.

We believe that this bounded liveness property is backed by a design specific invariant of the SSM model utilizable by SCADE DV. In other words, it is significantly simpler to prove that leaving an unsafe state needs at most 7 steps independently from the inputs and the fault sets considered, than showing that an unsafe state is not reached.

However, by applying the same approach to the dataflow model we could complete the safety analysis for the activation procedure only. Any attempt to establish a slightly more general result failed due to complexity problems. Manual abstractions, notably, cone of influence and symmetry, did not succeed to overcome this barrier. This can be understood as an indicator that built-in strategies in SCADE DV work well on these issues.

Formal safety analysis generates a huge amount of verification tasks. In our case study, we restricted ourselves to at most four possible faults per hardware component and focused on a single hazard. After arguing that at most two concurrent faults have to be analysed and strictly applying arguments of symmetry and mutual exclusiveness, we were left with more than a hundred verifications tasks for the DCCA. Even if each task could be individually handled by push-the-button model checking with some heuristic to enhance performance, it is another level of verification challenge to come up with a systematic solution capable of identifying *all* critical fault combinations.

5.2. Directions for Future Work

Intermediate Progress Information on Verification. In practice, the easy case is when an answer - positive or negative - is given by the model checker in reasonable time. But in case of complexity problems, todays verification engines give very little feedback. This situation can be improved significantly by outputting intermediate progress information and impact measures that indicate the relative contribution of selected model elements or system components on the size of the problem representation and computation time, e.g. within an inductive step. Such information will be of great help, when selecting a promising abstraction or reduction heuristic.

Systematics of Verification Heuristics. Our case study exposes the gap between the formal characterisation of safety analysis and a successful handling of the plenty of verification tasks that arise from it in practice. To bridge this gap a methodology and technical guidance are needed that direct the verification engineer through the variety of heuristics offering aid with complexity issues. We employed the following methods: (1) varying the environmental models, (2) symmetry, (3) cone of influence and data abstraction, (4) compositional verification in [5] and (5) a semantic weakening of the key safety property. We asked experts from the tool supplier for advice. But the successful combination we finally found is largely the result of exhaustive experiments. A systematics will be a key factor for establishing formal safety analysis in industrial practice.

Comparison to the Model Transformation Approach. In [5] we used a timing abstraction implemented by a model transformation to UPPAAL in order to finally prove that no single fault is critical. As here, this only worked for the SSM based model. After decomposing the liveness property according to the system architecture, we were able to identify a single component as the performance bottleneck. Then only this one was transferred to UPPAAL and we successfully proved a local derivate of the bounded liveness property to hold on that component for an environmental model slightly more general than (2) on p. 14. That suffices to finalize the safety analysis for those fault combinations containing only faults from that LBG. However, the breakthrough in performance relies on both the timing abstraction and the decomposition into local verification tasks. For an extension, that allows to uniformly deal with all fault combinations, the level crossing has to be transformed to UPPAAL completely. Thereby the number of clocks and variables will increase considerably which are well-known sensitive factors for UPPAAL's verification efficiency. Thus, it bears the risk that complexity problems recur. However, a detailed comparison in terms of figures is still open.

Acknowledgements. We are grateful to Siemens AG, IC MOL RA, for providing the real industrial models for our formal analysis and to Dr. Dirk Peter for fruitful discussions.

Appendix A. State-Based SCADE Model

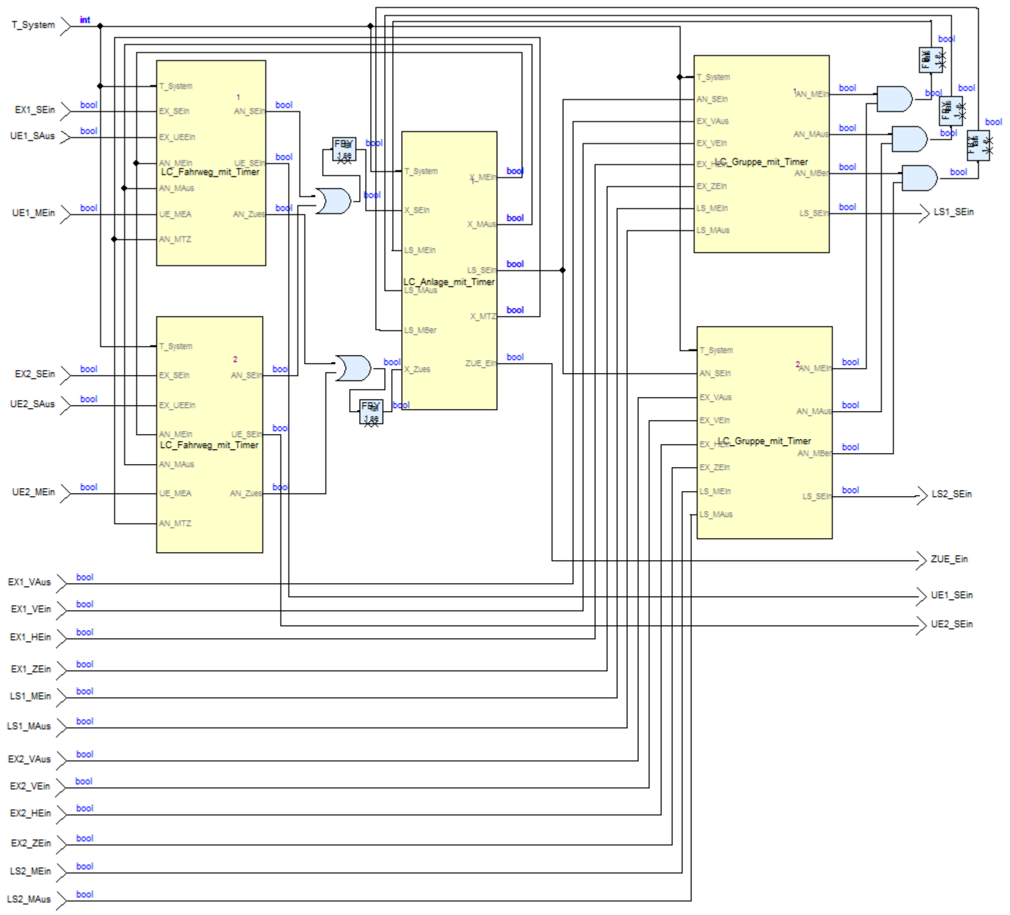

Figure A.6: First models' main wrapper operator

Appendix B. Dataflow SCADE Model

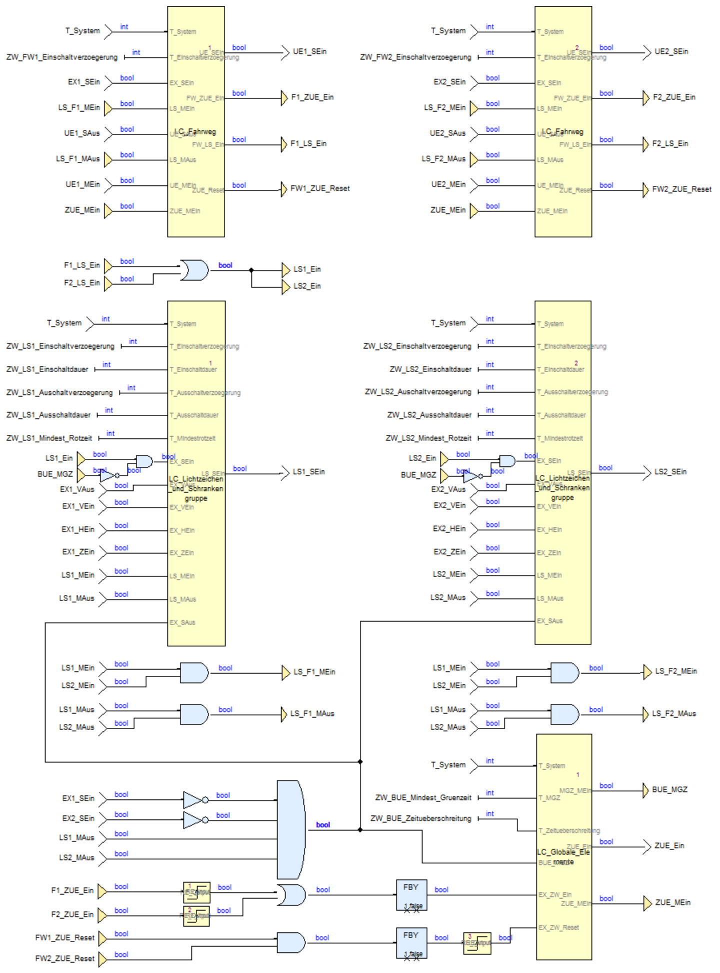

Figure B.7: Second model's main wrapper operator

References

[1] IEC 61508-3, IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements, 1998. Corrigendum 1999.

[2] CENELEC, EN 50128 – Railway Applications – Software for Railway Control and Protection Systems. European Standard., 2001.

[3] C. André, Semantics of S.S.M (Safe State Machine), Technical Report UMR 6070, I3S Laboratory, University of Nice-Sophia Antipolis, 2003.

[4] F. Ortmeier, W. Reif, G. Schellhorn, Deductive cause consequence analysis (DCCA), in: Proc. IFAC World Congress, Elsevier, Amsterdam, 2006, p. 6 pp.

[5] I. Daskaya, M. Huhn, S. Milius, Formal safety analysis in industrial practice, in: G. Salaün, B. Schätz (Eds.), Formal Methods for Industrial Critical Systems (FMICS), volume 6959 of *LNCS*, pp. 68–84.

[6] DIN, EN 50126: Spezifikation und Nachweis der Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS), 1999.

[7] DIN, EN 50129: Bahnanwendungen – Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme – Sicherheitsrelevante elektronische Systeme für Signaltechnik, 2003.

[8] IEC 60812, IEC 60812: Analysis techniques for system reliability, 2006.

[9] J. A. McDermid, M. Nicholson, D. J. Pumfrey, P. Fenelon, Experience with the application of HAZOP to computer-based systems, in: 10th Annual Conference on Computer Assurance (Compass), pp. 37–48.

[10] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud, The synchronous dataflow programming language LUSTRE, in: Proceedings of the IEEE, volume 79:9, pp. 1305–1320.

[11] P. A. Abdulla, J. Deneux, G. Stålmarck, H. Ågren, O. Åkerlund, Designing Safe, Reliable Systems Using Scade, in: T. Margaria, B. Steffen (Eds.), ISoLA, volume 4313 of *Lecture Notes Comput. Sci.*, Springer, 2004, pp. 115–129.

[12] N. Halbwachs, F. Lagnier, P. Raymond, Synchronous observers and the verification of reactive systems, in: Proc. of AMAST, Springer-Verlag, London, UK, 1994, pp. 83–96.

[13] A. Joshi, M. Whalen, ModelBased Safety Analysis: Final Report, Technical Report, NASA, 2005.

[14] M. Bozzano, A. Villafiorita, Improving system reliability via model checking: The fsap/nusmv-sa safety analysis platform, 2003.

[15] M. Güdemann, F. Ortmeier, W. Reif, Using deductive cause-consequence analysis (DCCA) with SCADE, in: Proc. 26th Intern. Conference on Computer Safety, Reliability and Security (SAFECOMP), volume 4680 of *Lecture Notes Comput. Sci.*, Springer, 2007, pp. 465–478.

[16] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, The MIT Press, Cambridge, Massachusetts, 1999.

[17] I. Daskaya, Comparative Safety Analysis and Verification for Level Crossings, Master's thesis, Technische Universität Braunschweig, 2011.

[18] H.-M. Hanisch, T. Pannier, D. Peter, S. Roch, P. Starke, Modeling and formal verification of a modular level-crossing controller design, 2000.

[19] S. D. Brown, D. Cofer, A. Bouali, Formal verification of an avionics sensor voter using SCADE, in: Proc. FORMATS/FTRTFT 2004, volume 3253 of *Lecture Notes Comput. Sci.*, Springer-Verlag, 2004, pp. 5–20.