Modeling and Analysis of State/Event Fault Trees using ESSaRel

Kavyashree Jamboti*, Michael Roth*, Robin Brandstädter*, Peter Liggesmeyer†
*Technical University of Kaiserslautern, Dept. Software Engineering: Dependability
Building 32, Paul-Ehrlich-Straße, 67663 Kaiserslautern, Germany
{jamboti, michael.roth, r_brands}@cs.uni-kl.de
†Scientific Director
Fraunhofer, Institute for Experimental Software Engineering
Kaiserslautern, Germany
Peter.Liggesmeyer@iese.fraunhofer.de

***Abstract*—Fault Trees (FTs) have been a popular tool used in the industry and academia to model safety related failure scenarios of systems. However, since FTs are incapable of modeling certain type of scenarios involving stochastic dependency, timing and sequencing properties, they need to be extended or modified to handle such scenarios. A State/Event Fault Tree (SEFT) is one such tool for developing and analyzing systems with dynamic behavior involving sequencing, timing and priorities of events that cannot be modeled by ordinary fault trees. SEFTs encompass dynamic behavior in the form of state charts for constituent components of a system where failure propagation between components is made possible by outports and inports. Conceptually, SEFTs borrow the notion of components from Component Fault Trees (CFTs). CFTs are nothing but fault trees which encompass boolean logic related to failure within the corresponding component boundaries. The ESSaRel tool was initially built to model CFTs. In this paper, we describe our experiences with the implementation of an editor for SEFTs by extending the ESSaRel tool. We describe the concepts behind the design decisions of the tool and the challenges that were addressed in order to reduce the burden on the user to develop 'correct' SEFTs. We also give some insights and tips for engineers who would like to use SEFTs as modeling correct SEFTs requires a good understanding of the semantics of its modeling elements.**

Keywords*—*Fault trees, Reliability tool, Safety tool, State/Event Fault Tress, ESSaRel

I. INTRODUCTION

Fault Trees (FTs) are an established method for conducting safety analysis of systems due to their ability to provide both qualitative and quantitative analysis results. They are able to capture those combinations of events that lead to a compromise in safety of systems which might not have been captured by other safety techniques. With the advent of component-based development, it became important to be able to create models for individual components, which can be combined to create a system model. However FTs have no notion of components, a functional/structural change made to one component implies that the entire FT has to be reconstructed, or at least it has to be ensured that there is no need to make further changes to the FT after making changes to the corresponding part of the FT. This can be a time consuming task considering that FTs can run into several pages depending on the size of the system. Component Fault Trees (CFTs) [1] combat this problem very elegantly by introducing component boundaries around failures and gates where failure propagation between components is facilitated using outports and inports. This makes it very easy to identify where the changes in the CFT has to be made while the rest of the CFT remains unchanged. Although one can easily modify existing CFTs and combine CFTs for constituent components to obtain a CFT for the whole system, they are just an extension of FTs and still have the inherent shortcomings of the fault trees' inabilities to model timing and sequencing of failure events. Also, there is no difference between states which refer to a persistent condition of a component and an event which refers to an occurrence without a temporal expansion. State/Event Fault Trees(SEFTs) [2] go one step further by extending CFTs to include the above mentioned features by introducing state-charts in components. These state-charts capture failure related behavior of components. Events and states can influence and be influenced by the behavior of other components through ports. SEFTs offer a wide range of gates that render it a powerful tool to model a variety of safety scenarios. Furthermore, any other specialized gate can be modeled by the user as a component that can be reused whenever it is required. Please note that unlike gates in a FT, which are capable of only enclosing boolean logic, gates in SEFTs can be thought of as components with an internal state-chart of their own which are capable of modeling sequencing, timing and memory. ESSaRel (Embedded Systems Safety and Reliability Analyzer) is a tool that was initially designed to provide an editor to model and analyze CFTs. It was then later extended to enable modeling and analysis of SEFTs. SEFTs have to be transformed to extended Deterministic Stochastic Petri Nets (eDSPNs) in order to be analyzed. The extended version of ESSaRel provides an editor for modeling SEFTs and also implements the translation algorithm that converts an SEFT to an eDSPN in a format compatible with the TimeNET tool [3], which can be used to analyze the translated petri nets.

SEFTs, although powerful, require a thorough understanding for their proper usage. In this paper, we describe how ESSaRel assists a user in creating SEFTs without syntactic errors. We describe our experiences of extending ESSaRel for modeling SEFTs. We also provide pointers regarding how to use SEFTs in the real world in order to create meaningful and semantically correct models.

 ISBN: 978-1-61208-301-8

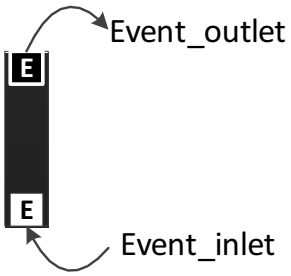

Fig. 1: Event Inlet and Outlet

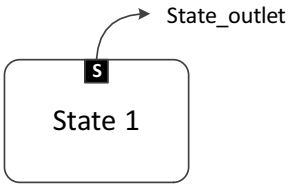

Fig. 2: State Outlet

II. Modeling elements of SEFTs

In this section, we describe the modeling elements of SEFTs, their semantics and constraints associated with them. We first list out the modeling elements defined in [2] for SEFTs and then describe the modeling elements that we added to SEFTs to be able to create them using a tool. The conceptual modeling elements are:

(a) Component which encompasses the behavior of the modeled component in the form of state charts.
(b) State charts that contain state changes facilitated via events. States and events are connected using temporal connections which are represented by arrows with unfilled arrowheads.
(c) Inports and Outports that allow propagation of failures in and out of components. These ports are further typed as 'state' or 'event' ports which are connected to other elements in the state chart or gates by causal connections represented by arrows with filled arrowheads.
(d) Gates are connected to one or more inputs leading to an output that combine states and events using boolean logic, priorities and may have parameters such as delays associated with them. As mentioned earlier, gates in SEFTs are not just boolean gates like the ones used in traditional FTs, but have internal state charts associated with them.
(e) Once an SEFT interface consisting of outports and inports has been created for a component type, it can be instantiated in other components of which it is a part. This can be done by dragging and dropping a component or by creating a proxy from the palette and then choosing the component type in the ESSaReL tool. Both methods result in creation of a component instance with its interface consisting of the inports and outports which we refer to as Proxy State Inlets/Outlets and Proxy Event Inlets/Outlets.

In addition to the above mentioned elements, we add a few modeling elements to the tool which are described below:

(a) Event inlet and Event outlet for triggered or triggering action of events (Fig. 1).
(b) State outlet for connecting outgoing causal edges from states (Fig.2).

A state does not have a state inlet as states cannot be triggered from other components, only events can be triggered which can cause the corresponding state transitions.

III. Design decisions for ESSaRel

As the use of Component Fault Trees increased in the industry and in research, it became clear that the first version of the tool [4] needed to be rewritten. The reason for this was that the existing version was not flexible enough. With the increasing use of CFTs in the academic/research field, new ideas were implemented in different tools and there was need to integrate ESSaRel with these other tools. For example, CFTs created in different front-ends such as Magicdraw [5] needed to be analyzed using ESSaRel. Also multiple back-ends such as Fault Tree+ and Zusim could be used for analysing CFTs. To address this issue, there is a common data model was introduced for both CFTs and SEFTs on which ESSaRel is based. This serves as an intermediate model between ESSaRel and other tools. In addition, for SEFTs, there is a common data model for DSPNs as well so that once an SEFT is translated to a DSPN, different backends capable of analysing DSPNs can be used. Fig. 3, 4 and 5 show a fire alarm system described in [4] modeled in the new version of ESSaRel. In the repository explorer of Fig. 3, we can see the three components: FireScenario (which represents the system under study), FireAlarmUnit and Watchdog. The FireAlarmUnit was modeled just once (Fig.3) but instantiated twice as FireAlarmUnit1 and FireAlarmUnit2 in the FireScenario component. Watchdog has been modeled (Fig.4) and instantiated once in FireScenario, its outport has been connected to inports of both instances of the FireAlarmUnit.

One can notice that there are no outports or inports in the palette in Fig.3. This is because the canvas that is visible is that of the realization of the FireAlarmUnit. There is another canvas for the interface (which has not been shown) whose palette contains the inports and outports. Once a port is added or deleted from the interface, ESSaRel automatically synchronizes the interface and the realization by making the corresponding change in the realization. This ensures consistency by making sure that the only way to add or remove ports in the realization is by modifying the interface. The reason for this kind of separation between the interface and realization is to allow a user to have the flexibility to choose between different realizations for a component. For example, the values at the outports of a component may come from an underlying SEFT or a MATLAB/Simulink model. The realizations for the Watchdog and the FireAlarmUnit are shown in Fig. 4 and 5 respectively.

In the remainder of this section, we describe the design features of the tool to make it easier for users to create syntactically correct SEFTs. Tables I and II below show the constraints associated with causal and temporal connections respectively. We have omitted those modeling elements that cannot be sources on the first column and those that cannot be targets on the first row in order to reduce the size of the tables. The entries with a check mark (✓) indicate the

 ISBN: 978-1-61208-301-8

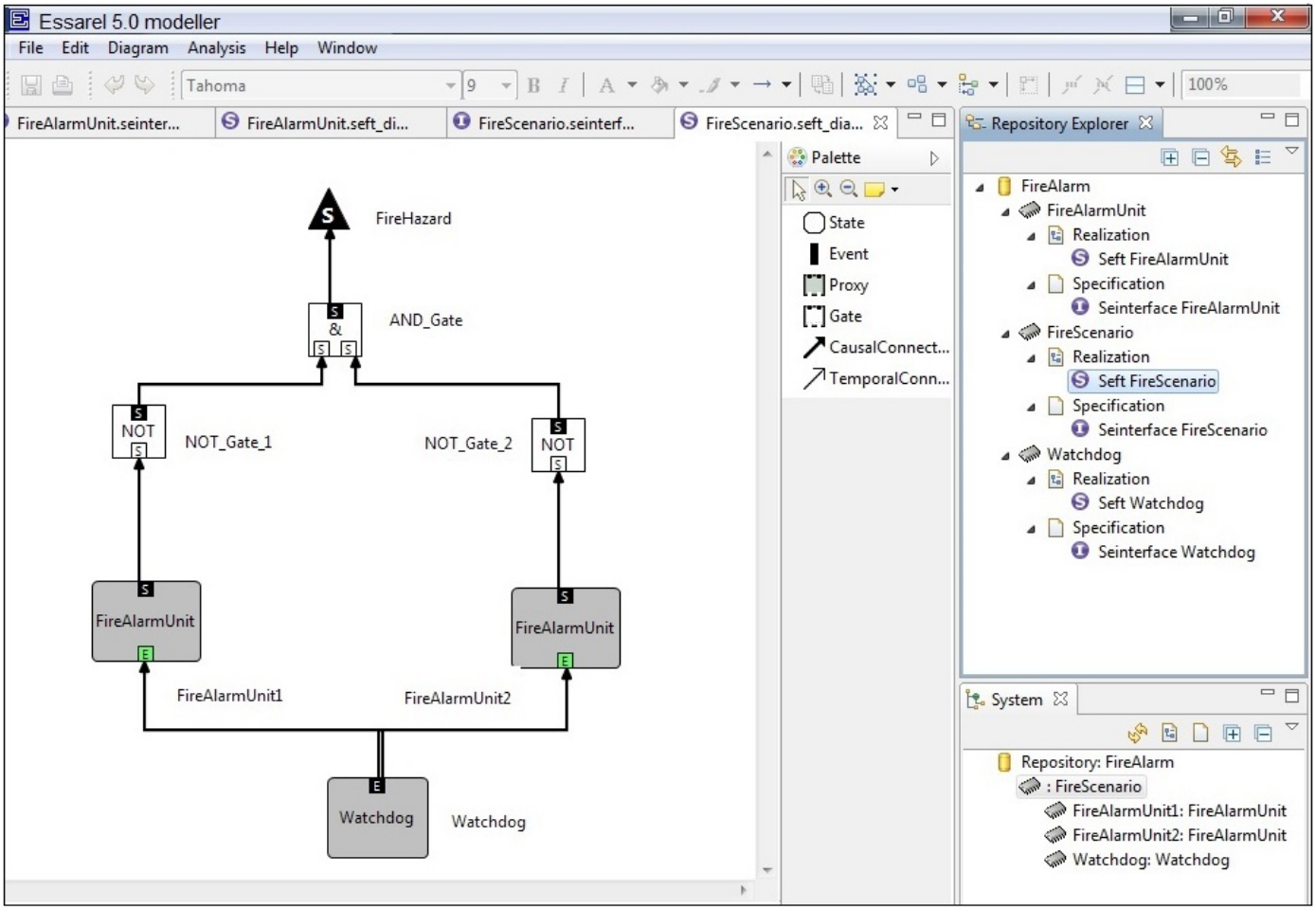

Fig. 3: Fire Scenario modeled in ESSaRel

valid connections and entries with a cross mark (✗) are prohibited. We can see that there are a large number of rules for connecting modeling elements which increase the chances for a user to unknowingly create wrong connections. ESSaRel prevents such errors by inhibiting the creation of prohibited connections.

It may be useful for users to note that only outgoing causal edges are allowed for states through state outlets, but states cannot have incoming causal edges. States can occur only by the triggering of a preceding event connected to the state through a temporal connection. This implies that states cannot be 'propagated'; only events can facilitate propagation by triggering events in other components. States, on the other hand can act as guards for an event so that the event can only be triggered when the guarding state is true. As mentioned in the previous section, this is the reason why a state does not have a state inlet.

From Table 2 for temporal connections, we can see that a state can only be connected to events and vice-versa, i.e., there is a strict alternation between states and events connected by temporal edges. A state can have two or more outgoing and incoming edges, but an event can have only one incoming and one outgoing edge.

Apart from the above constraints, the tool ensures that there is not more than one incoming causal edge for a target. This is necessary to ensure that there is no ambiguity in the cause of an event or state. The tool also ensures prevention of shallow cycles where a component references itself by preventing proxies of a component to be created in its own realization.

IV. Practical Modeling Tips for Users

Often, it can be confusing for users as to which is a good way to model a given scenario as the same scenario can be represented in multiple ways. For example, let us consider a situation where an intermediate failure can be represented as either the output of n-state-AND gate or as the output of the History-AND gate for events. In such a situation, it is important to understand well the difference between states and events. States have a persistent nature, but a component is also capable of changing its state. On the other hand, events do not have any temporal expansion and once they occur, the only way to record their occurrence is through the use of a History-AND gate. Therefore, in a situation where we want to model a scenario where two or more components are required to be present in a certain state for an event to occur, we can use the n-state-AND gate and when we want to model a scenario

 ISBN: 978-1-61208-301-8

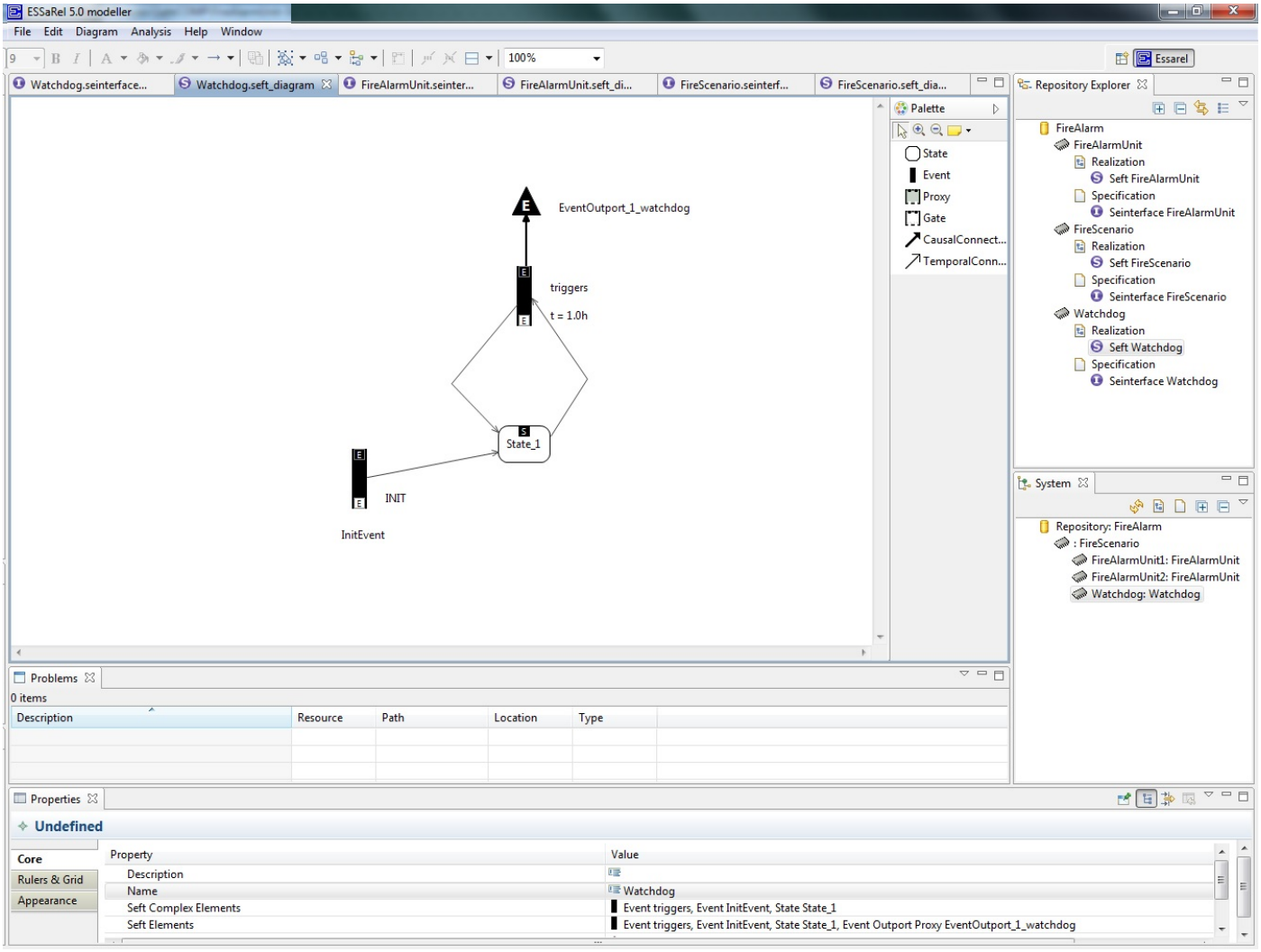

Fig. 4: Realization for Watchdog

where the occurrence of some events cause an intermediate event even when the components may no longer be in the states resulting from those events, we can use a History-AND gate.

As another example, let us consider the situation for modeling state chart interaction in SEFTs where there are two possibilities. The components can interact with each other via state ports or event ports. As input ports both possibilities have to be connected to events, either as a triggered event or as a guard function of a state. But we are of the opinion that the event based communication is better because triggered events are much more intuitive compared to guard functions. It is indeed possible to transform every state based communication into an event based one by using a Flip-Flop gate. To do this, the 'Set' port of a Flip-Flop gate has to be connected to all incoming transient arcs of the state and the Reset port with all outgoing arcs. But it is not advisable to do so unless necessary as the complexity of the transformed state-chart (into Petri Nets for analysis) increases and the advantages of a more intuitive SEFT are nullified. In this case a state-based communication is preferable.

A. *Nomenclature Scheme for SEFTs*

SEFTs, like fault trees, are constructed by humans who may give arbitrary names to failure ports. This may lead to miscalculations during quantitative or qualitative analysis. More information on consequences of wrong or ambiguous nomenclature has been documented in [6], [7], [8], [9]. Here, the authors recommend the use of two fields to construct FT event names:

(a) Component Name, which is the fully qualified name of the component given by the system decomposition hierarchy.
(b) Failure Mode, which describes the nature of the failure.

But since SEFTs are specialized FTs, the above two fields are not sufficient for unambiguous nomenclature of its events. Hence, to ensure unambiguity, we recommend to users to construct names with the following two additional fields along with the ones mentioned above:

(a) Environment Condition, which depends on the context in which the component is deployed.
(b) System Condition, which depends on the configuration

 ISBN: 978-1-61208-301-8

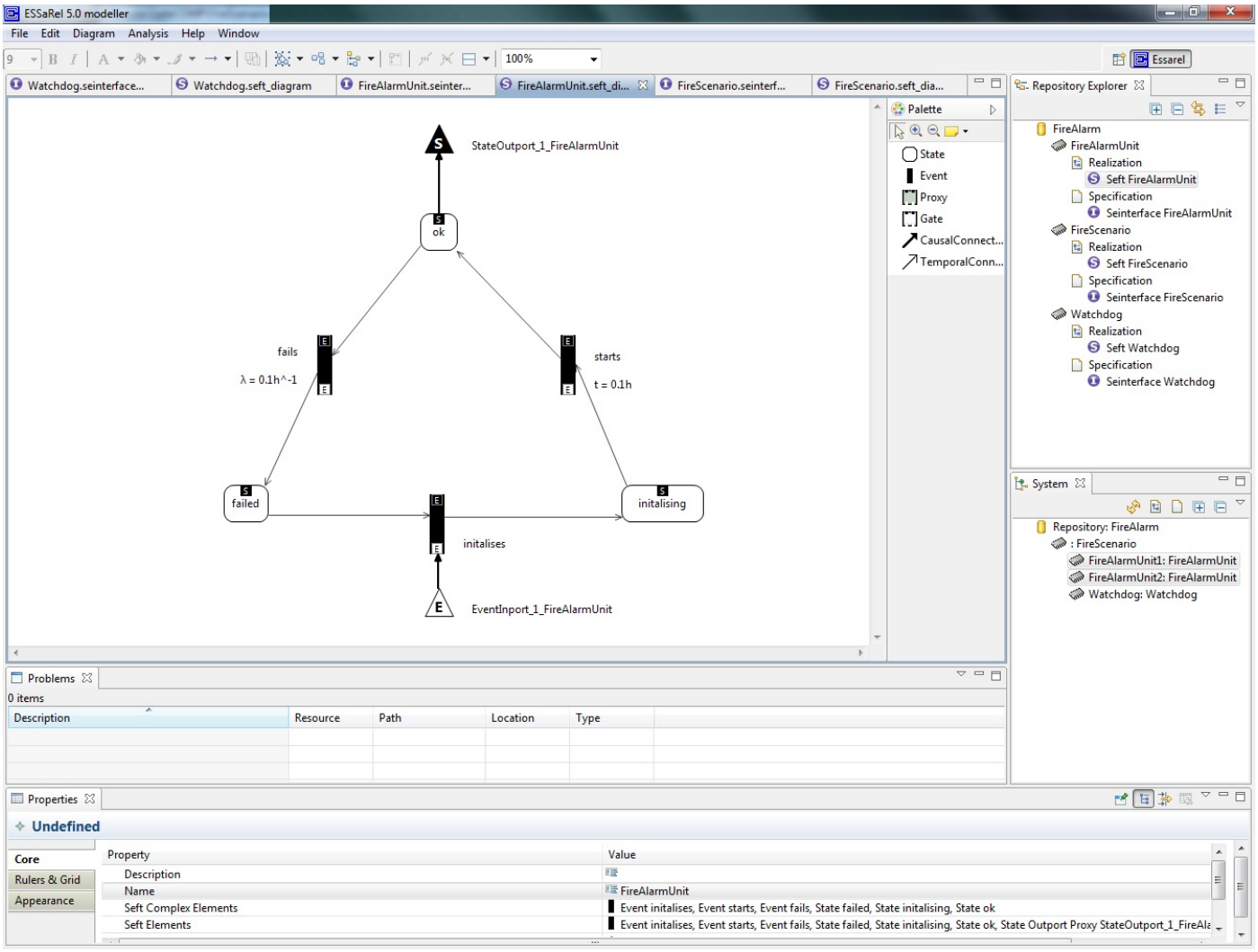

Fig. 5: Realization for Fire Alarm Unit

peculiar to the component itself. This field is useful to distinguish between two or more events that are sources for temporal edges leading to a common target state.

These fields help to distinguish names of failure modes of the same component the working conditions, which depends on the context of deployment and the configuration of the component. For more information on the description and usage of the above fields, users are suggested to read [10].

B. Analysis

This subsection explains the evaluation functions of TimeNET. SEFTs can be component-wise translated into eDSPNs. This operation transforms events into transitions and states into places where the initial state is expressed as the initial marking of the resulting net. Deterministic and exponentially distributed events can directly transfer into the equivalent Petri Net transitions. For triggered events, however, a pattern exists to connect different Petri Nets with each other. According to the gate dictionary [4], for every gate an equivalent eDSPN is available. By the usage of these transformation functions, a Petri Net can easily be built out of every SEFT. In addition to normal DSPNs, eDSPNs support probabilistic values as well as priorities to avoid conflicting situations. The probabilistic functions specify a value which stands for the likelihood that a transition fires after its activation. Further it provides textual elements such as the so called performance measures. These measures represent the asked questions such as "What is the probability for a certain marking of a specific place?" A special grammar has to be used for defining the measures which can be found in [3]. An example of such a performance measure for the probability that place P1 contains more than one token is P{#1>1}.

In TimeNET, there are different evaluation methods for the analyzing these performance measures. The categories are divided into (1) analysis and (2) simulation techniques. To run an evaluation, at least one performance measure has to be specified. The first evaluation category gives an exact numerical result by computation of the reachability graph. Therefore, the reachability graph has to be finite. In case of a transformed SEFT this precondition is always fulfilled because all converted Petri Nets are bounded. This type of

 ISBN: 978-1-61208-301-8

TABLE I: Constraints for Causal Connections

Source \ Target	Proxy State Outport	Proxy Event Outport	State Inport	Event Inport	Event Inlet	Gate State Inlet	Gate Event Inlet
Proxy State Inport	✓	✗	✓	✗	✓	✓	✗
Proxy Event Inport	✗	✓	✗	✓	✓	✗	✓
State Outport	✓	✗	✓	✗	✓	✓	✗
Event Outport	✗	✓	✗	✓	✓	✗	✓
Event Outlet	✗	✓	✗	✓	✗	✗	✗
Gate State Outlet	✓	✗	✓	✗	✗	✓	✗
Gate Event Outlet	✗	✓	✗	✓	✓	✗	✓

TABLE II: Constraints for Temporal Connections

Source \ Target	State	Event
State	✗	✓ (incoming edges = 1)
Event	✓ (outgoing edges = 1)	✗

evaluation technique can be further subdivided into transient and stationary analysis, also called steady-state analysis. The transient evaluation can analyze the net from the point of initial marking during a given time period. The result can be shown as a curve which represents the defined measure for the complete interval or as a point which represents the measure only at the end of the interval. Steady-state evaluation is able to find a result without specifying such a time period. It could be seen as a transient analysis with an infinite time period. For getting a steady-state result it is necessary that the eDSPN is designed without deadlocks. This means that the reachability graph may not have any nodes where no transition is enabled. An additional precondition for the analysis of eDSPNs in TimeNET is the existence of at most one deterministic transition. If this precondition is fulfilled then these evaluation methods can be used to calculate the result for the measures depending on the required time period (finite or infinite).

The second category of evaluation techniques is the simulation methods. These methods estimate the measures by the use of a modified Monte Carlo simulation and can also subdivided into a stationary and a transient method. Because of the inaccuracy, simulation should only be used if there is more than one deterministic transition in the eDSPN. Apart from that, simulation methods need to fulfill the same preconditions as analysis methods. More information can be found in [3].

V. Conclusion and Future Work

ESSaRel has been designed to be a user-friendly tool that aids engineers to create syntactically correct SEFT models. Based on our experience, we have also described some aspects of SEFTs to provide tips to a user to build semantically correct SEFT models. It is possible to perform quantitative analysis on SEFTs, but it is not possible to perform qualitative analysis on SEFTs as not much research has been carried out with respect to this aspect. We intend to integrate qualitative analysis such as minimal cut set generation. We would like to integrate mechanisms for qualitative analysis when they become available. In the future, we would like to extend the functionality of ESSaRel to be able to directly display analysis results for SEFTs directly in ESSaRel without having to manually run the TimeNET tool.

Acknowledgment

This work was funded by the VIERforES project supported by BMBF, Germany.

References

[1] B. Kaiser, P. Liggesmeyer, and O. Mäckel, “A new component concept for fault trees,” in *SCS*, 2003, pp. 37–46.

[2] B. Kaiser, “State event fault trees: a safety and reliability analysis technique for software controlled systems,” Ph.D. dissertation, Technische Universität Kaiserslautern, 2006.

[3] A. Zimmermann and M. Knoke, “TimeNET 4.0 user manual,” Technische Universität Berlin, Faculty of EE&CS, Tech. Rep. 2007-13, 2007.

[4] B. Kaiser and C. Gramlich, “State-event-fault-trees - a safety analysis model for software controlled systems,” in *SAFECOMP*, 2004, pp. 195–209.

[5] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J.-P. Schwinn, and M. Trapp, “Integration of component fault trees into the uml,” in *MoDELS Workshops*, 2010, pp. 312–327.

[6] C. A. Ericson, “Fault tree analysis by design,” in *Proceedings of The 16th International System Safety Conference*, 1998.

[7] A. Long, “Variants of classical cutset characterization,” in *21st International System Safety Conference*, 2003.

[8] Y. Papadopoulos and U. Petersen, “Combining ship machinery system design and first principle safety analysis,” in *IMDC 03, 8th Intl Marine Design Conf*, Athens, May 2003, pp. 1:415–426.

[9] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. M. III, and J. Railsback, *Fault Tree Handbook with Aerospace Applications*. NASA, 2002.

[10] K. Jamboti, C. Gomez, O. Mackel, and P. Liggesmeyer, “Improved nomenclature schemes for component fault trees and state/event fault trees,” in *Annual European Safety and Reliability (ESREL) Conference(In-Press)*, 2013.