State-Event-Fault-Trees – A Safety Analysis Model for Software Controlled Systems

Bernhard Kaiser and Catharina Gramlich

Hasso-Plattner-Institute for Software Systems Engineering, Department of Software-Engineering and Quality Management, Prof.-Dr.-Helmert-Str. 2-3, Potsdam, Germany
{bernhard.kaiser,catharina.gramlich}@hpi.uni-potsdam.de

Abstract. Safety models for software-controlled systems should be intuitive, compositional and have the expressive power to model both software and hardware behaviour. Moreover, they should provide quantitative results for failure or hazard probabilities. Fault Trees are an accepted and intuitive model for safety analysis, but they are incapable of expressing state dependencies or temporal order of events. We propose to enrich Fault Trees with State/Event semantics. We use a graphical notation that is similar to Statecharts. Our model subsumes deterministic state machines that are suited to describe software behaviour and Markov Chains that model probabilistic failures. We allow exponentially distributed probabilistic events, deterministic delays and triggered events. The model is compositional and joins components by ports. Quantitative evaluation is achieved by translating the component models to Deterministic and Stochastic Petri Nets (DSPNs) and using an existing tool for analysis. We introduce the model and the analysis procedure and provide a small case study of a fire alarm system, completed by an outlook on our tool project ESSaRel.

1 Introduction

In technical systems, more and more mechanical and electrical components are replaced with software-controlled components. This includes safety critical domains such as avionics, automotive or industrial control. In these application fields safety and reliability analysis is a mandatory part of the development and must be supported by appropriate models and tools. Fault Tree Analysis (FTA) is one of the most widely used techniques in this context. Fault Trees (FTs) are intuitive for practitioners due to their hierarchical structure and the familiar logical symbols. They provide a set of qualitative and quantitative analyses. They have been used for several decades in the context of mechanical or electrical systems and are gaining importance in the context of software-controlled systems.

Nevertheless, some fundamental differences between Fault Trees and the models commonly used for embedded system design are obvious: Models for complex systems must be compositional. Modularisation of FTs, however, is only defined in a restricted way. Safety is principally a matter of behaviour and, in contrast to the state-space models used in systems design, FTs are not suitable for modelling behaviour. FTs are a combinatorial model that cannot capture sequences of actions and state history. The two-state abstraction (working or failed) of Fault Trees is not adequate for systems with complex state spaces.

M. Heisel et al. (Eds.): SAFECOMP 2004, LNCS 3219, pp. 195–209, 2004.

These differences not only hamper the application of FTA to software-controlled systems, but also obstruct the integration of state-based submodels into an FTA. This integration would be desirable for two reasons: first the reuse of state-based models from the design phase for safety analysis and second the integration of Markov Chains, which are an important state based safety and reliability model.

Existing approaches to overcome the semantic weaknesses of FTs often rely on formal methods that are not familiar to practitioners and do not offer visual integration for FTs and state based models.

We take a different approach by adding a notion of states and events to FTA: States describe conditions that last over a period of time whereas events are sudden phenomena, including state transitions. We call this extended model State-Event-Fault-Trees (SEFTs). States and events are depicted by different symbols. We propose typed FT gates for states and events (e.g. an OR gate with two event inputs and another OR gate with two states inputs). Regarding the AND gate that joins two events we distinguish a History-AND that remembers events that have occurred in the past and a Sequential-AND that remembers also if they have occurred in a given order (also known as Priority-AND gate). State-Event-Fault-Trees are partitioned into components which are interconnected by ports. Other kinds of state based models such as Markov Chains or state diagrams from CASE tools can be integrated.

SEFTs are well suited for industrial use since they unite familiar graphical notations; nevertheless, their semantics allows quantitative analysis. The analysis is performed by component-wise translation of the SEFT models into Deterministic and Stochastic Petri Nets (DSPNs) [6], a class of Petri Nets for which analysis tools exist (e.g. the tool TimeNET [19]). In the Petri Net domain the component models are merged to one flat model that is passed to an existing analysis tool.

In this paper we explain the application of SEFTs and the steps necessary for their translation to DSPNs. To illustrate the procedure we refer to a small case study of a fire alarm system. The rest of the paper is organised as follows: In Section 2 we give a short overview over FTA and previous adaptations to software-controlled systems. In Section 3 we introduce the modelling elements of SEFTs in summary and explain the analysis by translation to DSPNs. In Section 4 we introduce the case study and show how SEFT analysis is applied in practice. Section 5 concludes the paper and gives some pointers to ongoing and future research steps, in particular the implementation of the algorithm into our research tool ESSaRel (Embedded Systems Safety and Reliability Analyser [7], which is a successor of our current FTA tool UWG3.

2 Foundations and Previous Work

2.1 Introduction to Fault Tree Analysis

FTs [18] are a widely accepted model that graphically shows how influence factors (in general component failures) contribute to some given hazard or accident. They provide logical connectives (called gates) that allow decomposing the system-level hazard recursively. The AND gate indicates that all influence factors must apply together to cause the hazard and the OR gate indicates that any of the influences causes the hazard alone. The logical structure is usually depicted as an upside-down tree with the hazard to be examined (called top-event) at its root and the lowest-level influence

factors (called basic events) as the leaves. Note that in the context of FTA the term "event" is applied in its probability theory meaning: an event is not necessarily some sudden phenomenon, but can be any proposition that is true with a certain probability.

The analyses to be performed on FTs can be qualitative or quantitative. Qualitative analyses list, for instance, all combinations of failures that must occur together to cause the top-level failure. Quantitative analysis calculates the probability of the top-event from the given probabilities of the basic events. Combinatorial formulas indicate for each type of gate how to calculate the output probability from the given input probabilities. These probabilities are either probabilities that an event occurs at all over a given mission time or they are understood with respect to a given point in time. The evolution of a system over time or any dependencies between the present system behaviour and the history cannot be modelled. An important assumption to obtain correct results is the stochastic independence of the basic events, which is hard to achieve in complex networked systems. Most current FTA tools use the efficient representation of Boolean terms as Binary Decision Diagrams (BDDs) to compute the quantitative results.

2.2 Fault Tree Analysis for Software-Controlled Systems

Like many safety and reliability analysis models, FTs were originally designed for non-programmable systems. When more and more technical systems became software-controlled, the need to adapt FTs to this application field grew.

There have been several attempts to adapt FTA to software or embedded systems, to derive FTs from software models and to enhance the expressive power of FTs. [16] integrate FTs with formal program specifications and use Interval Temporal Logic to give a formal semantics to Fault Trees. Formal methods are also used in [2] and [10]. Other approaches to model dynamic behaviour and multi-state components map FTs to Markov Chains [2] or different variants of Petri Nets [4][5][11][14]. Some researchers [10][2] proposed additional Fault Tree gates, for instance describing conditional probability, sequence enforcing or various spare usage situations (hot, cold and warm spare) in order to model special cases of dependencies.

For an efficient and sound development process different modelling techniques from system design and safety / reliability analysis should smoothly integrate with each other. Research projects aiming at the integration of different models can increasingly be observed during the last years [8][3]. Many of them consider FTs, but often they are applied in a rather informal or qualitative way.

2.3 Component Fault Trees

Models for complex technical systems must be compositional in order to be manageable. Traditional FTs have this property only in the sense that independent subtrees (called *modules*) can be cut off and handled separately. Technical components, however, are often influenced by other components and thus cannot be modelled by independent subtrees. To allow for a suitable modularisation in these cases, we recently proposed a more advanced component concept [15]. It allows cutting arbitrary parts out off a fault tree so that they can be modelled and stored independently. This allows a modularisation that reflects the actual technical components. The model is inte-

grated and flattened during analysis. We call this enhanced model Component Fault Trees (CFTs). We introduced input and output ports that serve as interfaces to put the components together. Subcomponents are represented as black boxes with the ports visible at the edges.

From semantics point of view CFTs are ordinary FTs with the mentioned restrictions. However, apart from the better compositionality, the CFT concept prepared the ground for the use of ports to achieve integration of other models. We later refined our ports into State Ports and Event Ports, as will be explained in detail in the following. When we started to integrate components that are described by Markov Chains or Statecharts as subcomponents into CFTs, we found that the lack of semantic precision of FTs made it hard to connect states or events consistently to a FT. In response we took the approach of enhancing FTs by a State/Event distinction to allow the combination of different modelling elements techniques.

3 State-Event-Fault-Trees

3.1 Introduction to the SEFT Notation Elements

State-Event-Fault-Trees (SEFTs) are a model that combines elements from FTA and from Statecharts [13], ROOMcharts [17] or similar notations. We deal with a finite state space for each component, an abstraction that is sufficient for safety and reliability considerations. Each component is in exactly one state at each instant of time, called the active state (we leave out state hierarchy for now). We denote states by rounded rectangles, as in Statecharts. For safety analysis we consider *states* as conditions that remain valid for a non-empty interval of time. We call a propositional term over states a *state term* (e.g. "Component C1 is in state S1 *or* in state S2"). Note that more than one state term can be true at the same time. For each point of time we assign a value 0 or 1, representing the Boolean values false and true, to any state term. For probabilistic analysis the annotation domain is extended to a real value p with $0 \leq p \leq 1$ that represents the probability that the component is in this state at the given instant of time. In this case the meaning of AND and OR transfers from the propositional meaning to the meaning "probability, that state 1 AND / OR state 2 are active at the same time".

Event is the term we use for atomic phenomena that do not take time to occur (this is in contrast to the standard FT definition). In particular, state transitions are events, but there may be independent events as well, e.g. spontaneous actions that occur in the environment (e.g. "Tube breaks"). For quantitative analysis a probability density must be assigned to events. If an event is a transition from one state to another we call these states predecessor and successor state. We distinguish the *event* (denoting a class of similar phenomena that can happen at different times) from the *occurrence,* which is associated with an instant of time. Since we refer to a continuous time scale for our model we assume that any two independent events cannot occur at exactly the same time.

We mark events by solid bars. The resemblance to Petri Net transitions is not coincidental: we later translate events to Petri Net transitions for analysis. In our model events occur in one of three ways: either they are *triggered* by other events, or they occur after a deterministic delay t upon entry of their predecessor state, or they occur

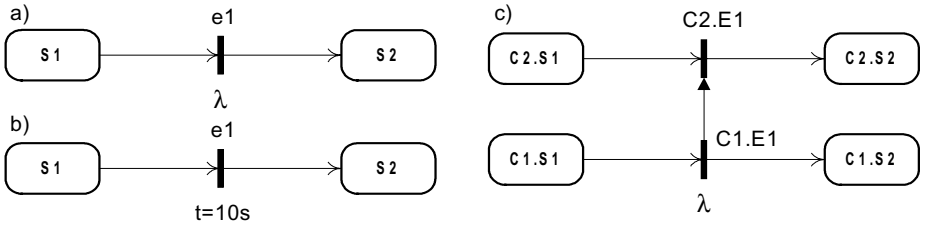

Fig. 1. a) Exponentially Distributed Transition b) Deterministic Delayed Transition c) Triggered Transition

after an exponentially distributed probabilistic delay. Thus, the expressive power of SEFTs subsumes both Statecharts that appropriately describe software behaviour and Markov Chains that are a customary state-based model for hardware failures. Fig. 1 shows all of these variants.

The example of the triggered transition is simplified for explanation purpose; it ignores the fact that the states and event belong to different components. Note that the two different kinds of directed edges: Those with light arrowheads mark the predecessor-successor relation between states and events (temporal edges) and those with bold arrowheads mark the triggering relation (causal edges). Causal edges between two events have the semantics that each time the source event occurs, the target event occurs as well, provided that it is enabled. Enabled means that the component the target event belongs to is in one of the predecessor states of the target event. If the source event happens at an instant t, then the target event occurs at t^+, so triggering does not encompass any delay. If, however, the modeller wants to introduce some explicit deterministic or probabilistic delay between source and target event, SEFTs offer a DELAY gate.

Causal edges can also have states as their source. States cannot trigger other states or events, but state terms can serve as guards for events, meaning that the event can only occur if the state term evaluates to true.

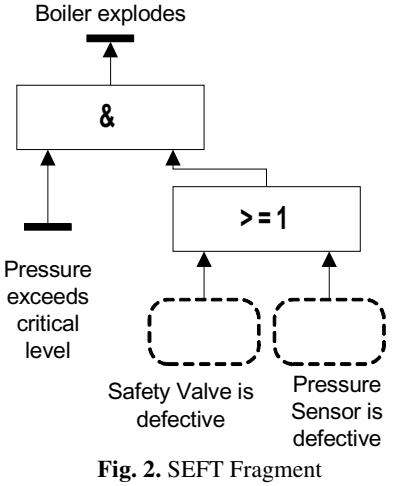

Fig. 2. SEFT Fragment

As in FTA, gates add logical connectors to the causal paths. Consequently the edges that connect gates are called causal edges as well. The most important gates are AND, OR and NOT in their different variants. SEFT gates are typed in the sense that they have different semantics depending on whether they are applied to state terms or to event triggering relations. For instance, the fragment in Fig. 2 has the semantics that the event "Pressure exceeds critical level" triggers the event "Boiler explodes" only if the state term "Safety Valve is defective" or the term "Pressure Sensor is defective" is true. In a complete example these unspecified state terms (drawn as dotted state symbols) could be states of two other components "Valve" and "Sensor".

SEFTs allow the extension of Fault Trees to Directed Acyclic Graphs (the same cause triggers multiple effects) and deal with repeated events or states correctly. Causal cycles without explicit delay are not allowed, because this would raise some semantic problems during analysis.

Just as Component Fault Trees, SEFTs are organized by components. *Components* are prototypes and must be instantiated. Components can be referenced as subcomponents of another component, forming a component hierarchy. The component on top of the hierarchy is the system to be examined. Each instance of a component defines a separate name space and all internal states and events are distinct from the state and events of other instances and hidden from the environment.

Ports achieve the connection of components across hierarchy levels. We distinguish input ports from output ports and state ports from event ports. Subcomponents appear as black boxes where only the ports are visible. Examples can be found in the case study in Section 4. Event ports allow triggering relations from one component to another: The information that some event occurs is transferred from the source component (the component where the output port belongs to) to the destination component (the component possessing the input port). There it can provoke some effect, provided that the destination component is ready to accept it. Otherwise, the event in the source component is neither blocked nor stored, but just discarded. The semantics of a state port is that the destination component has access to the information whether or not the state term in the source component is active, without having any means to influence that state.

3.2 Application of SEFTs

SEFTs are constructed like traditional FTs. Starting with some undesired system state (hazard) or event (accident), the analyst traces back its influences and finds out which system states or events play a role in initiating, propagating or inhibiting the fatal behaviour. The richer variety and semantic precision of gates allows better capturing chains of embedded systems behaviour. The basic events of standard FTA correspond to solitary exponentially distributed events in SEFTs. The project is structured hierarchically using the component concept.

Models that explain the relevant behaviour of subcomponents can be plugged in where necessary. For stochastic failures Markov Chains are appropriate. They have traditionally been used for hardware wear-and-tear, but there can also be stochastic models for software failures. To model software and control aspects of the system, Statecharts or similar models can be reused from the design phase, e.g. by importing them from a CASE tool. The visible difference is that the transitions, which are originally represented by labelled edges, now appear as explicit transition symbols. Software models can serve to check the reaction of the correct software on rare or unforeseen events from the environment or probabilistically model software failures.

3.3 Analysis by Translation to DSPNs

A model should not only be a graphical notation, but also provide analysis for relevant properties, supported by usable tools. Computer based analysis requires a formal semantics. Defining a formal semantics for a human-centred notation is a difficult

task, as various attempts to formalise Statecharts or FTs show. A second issue is that being state-based models, SEFTs cannot be evaluated by the traditional combinatorial FTA algorithms. To tackle both issues at the same time we propose to translate SEFTs into an accepted formal notation where known analysis algorithms exist.

Petri Nets (PNs) are a model for discrete state systems that supports the concurrency we have to deal with in component based systems and provides stochastic variants. We chose Deterministic and Stochastic Petri Nets (DSPNs) [1] since they possess all kinds of transitions we need and provide analysis techniques for the properties we are interested in (in particular the probability of a place to be marked). They are an extension of Genaralized Stochastic Petri Nets (GSPNs) that lack deterministic delay that often has to be considered in software behaviour. Assuming some basic knowledge about Petri Nets we briefly point out the main features of DSPNs: DSPNs are a timed variant of Petri Nets, i.e. the (possibly probabilistic) time that a transition waits before it fires after becoming enabled is specified in the model. In particular, DSPN transitions fire in one of three ways: immediately on activation, after a constant delay (specified by an annotated time parameter) or after an exponentially distributed random delay (specified by an annotated rate parameter). Firing of transitions is atomic and takes no time. In the graphical representation, black bars depict immediate transitions, empty rectangles depict transitions with exponentially distributed firing time, and black filled rectangles depict transitions with constant delay unequal to zero. Transitions are joined to places by input arcs, output arcs or inhibitor arcs. The latter forbid firing as long as the corresponding place is marked. Priorities can be attached to immediate transitions to resolve conflicts. Places can have a capacity of more then one token and arcs can have a multiplicity of greater than one, but we currently do not exploit this property. The underlying time scale is continuous.

Analysis of DSPNs has been described in [6][9] and several tools are available. We are currently carrying out some experiments using TimeNET [19] that require manual translation, but we are working on an automated integration of both tools.

The translation of SEFT states and events to DSPN places and transitions is straightforward: each state is mapped to a place and each event to a transition. SEFT gates are translated as a whole by looking up the corresponding DSPN structure in a dictionary. A part of this dictionary, containing different kinds of AND, OR and NOT gates can be found in Fig. 3. The dashed places or transitions signify import places / transitions, which are references to other places / transitions (marked with "export") in other component DSPNs. During flattening (integration of the partial nets), the import elements will be merged with the corresponding export output. The semantics of the gates is best understood by playing the “token game”. For instance, the export place P_{out} of the AND (State x State) gate net is marked if and only if someone puts tokens to both import Places P_{in1} and P_{in2}. The situation is different with the Sequential-AND (Event x Event) gate: here the left input transition must fire first and then the right input transition to make the output transition fire.

Our semantics that lies in the composition by state ports and event ports cannot be translated directly to a composition in the Petri Net domain, so special patterns are necessary when translating SEFT ports to DSPNs. The reasons are that ports must not introduce any backward influences and events are not stored as tokens are.

To translate an event port an additional place is added where the triggering transition puts a token when firing. Triggering only occurs if the triggered transition is feasible at the same instant of time and there is no storage of events. To capture this,

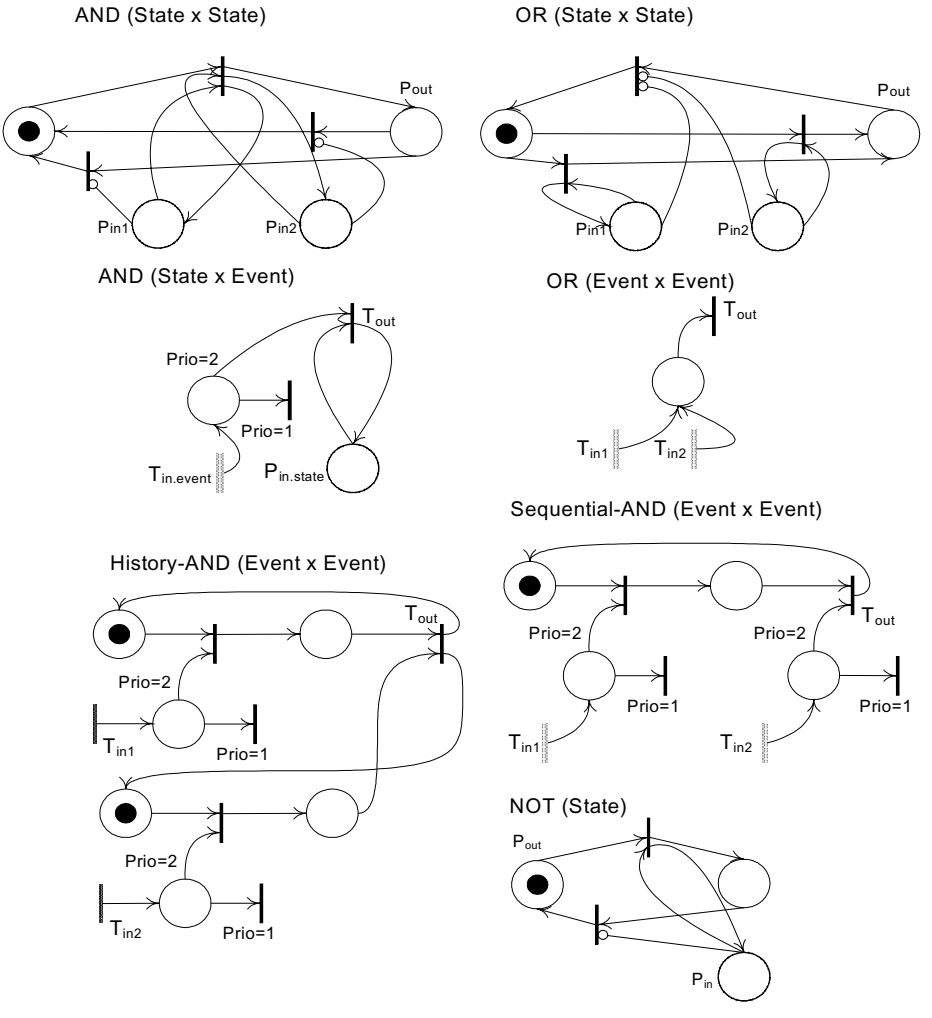

Fig. 3. Gate Translation Dictionary (Excerpt)

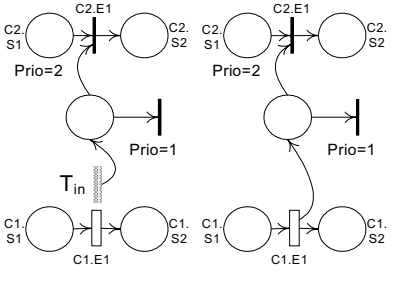

Fig. 4. Trigger / Event Port Pattern

we add an artificial immediate transition with lower priority than the triggered transition so that it consumes the token in those cases when it is not immediately used. The event port (or trigger) pattern can be seen in Fig. 4, before and after flattening. Note that n DSPNs priority 2 takes precedence over priority 1.

Regarding state ports, it is important that the target component must not backward modify the source component

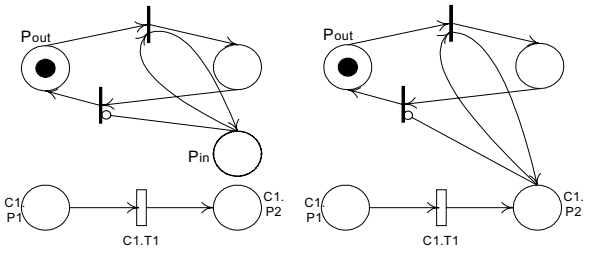

Fig. 5. State Port, exemplified by a NOT gate

state (and thus the DSPN marking). For this purpose we enrich the target component by a state machine with two states, reflecting whether or not the source component is in the relevant state. DSPN inhibitor arcs (depicted with a small circle at the end) are a useful means to sense that a place is *not* marked without reverse influence, because inhibitor arcs never consume a token. Unfortunately they have no positive counterpart that senses if a token *is* present without removing it. A way out is to accompany each arc that removes a token by a complementary arc that puts it back immediately after. As an example we show how in Fig. 5 a NOT-Gate (see dictionary) is merged to an export state (state connected to an output port) of some state machine. Notice how the inhibit edge and the pair of normal edges are connected in a way that excludes backward influences. We call this constellation the state port pattern.

3.4 The Translation Algorithm

The translation and preparation comprises the following steps:

Preconditions:

1. The component model to be analysed and all nested subcomponent models (recursively) are available at analysis time and are valid SEFT models
2. The component nesting hierarchy is free from cycles (a component must not refer directly or indirectly to itself as a subcomponent)
3. All component input ports are connected on the higher hierarchy level
4. All causal paths are free from cycles (this must be checked across component borders and hierarchy levels)

1st Step: Component-wise translation (creates a DSPN for each component)
For each component

- Prefix the IDs of all modelling elements by the IDs of the component instance they belong to (to avoid name conflicts)
- Represent each SEFT state by a DSPN place
  - If the state is the initial state, mark it by a token
- Represent each deterministic or exponential event by a deterministic or exponential DSPN transition and transfer the parameter
  - If a transition is solitary (no predecessor and successor state modelled) then add one marked DSPN place which is both predecessor and successor place
- Represent each triggered event or event with zero delay by an immediate transition
- Transform each causal edge joining two events to the trigger pattern
- Represent each temporal edge from SEFT by a DSPN edge

- Replace each Fault Tree Gate by the DSPN structure indicated by the dictionary
- For each port of the component and its subcomponents
  - Create for each connected causal edge an entry in the connection table (source and destination ID)
  - If a state / event is the source of the causal edge leading to a port, then label this state / transition as "*export*"
  - If a state / event is the target of the causal edge, then apply the state port / event port pattern and label the input state / transition of this pattern as "*import*"

2nd step: Flattening (creates one DSPN out of the component DSPNs)

- Resolve the connection table to remove any ports except output ports of the system
- Merge *import* with corresponding *export* places and transitions. The counterparts are found in the connection table

One exported place or transition can be merged with several counterparts. It is a failure if port references cannot be resolved or if import places or transitions have no export counterpart.

After the SEFT has been translated and flattened, the initial markings of the DSPN places must be applied where missing and the place or transition connected to the output port selected for analysis must be identified.

We intended to introduce a net simplification step before the flattening steps on each hierarchy level to reduce the state space.

3.5 Performing the Analysis

To do the analysis, the requested measure (e.g. average probability of a state term that is connected to an output port) must be translated into a measure that can be determined by the DSPN analysis tool (e.g. the marking probability for a place that corresponds to the system state of interest). Then a suitable analysis procedure must be started. The tool TimeNET that we are currently using offers both transient and steady-state analysis for DSPNs plus simulation. Analysis is faster but due to the used analysis algorithm it can only be applied in cases where at most one deterministic transition is enabled in any marking. If this condition is violated, simulation is still possible. At present state, we read back the results of the analysis manually. Our tool ESSaRel that is currently under development will start the analyser via API or command line call and later read the calculated values from the TimeNET result file to display them on its own GUI.

4 The Fire Alarm System

4.1 Description of the Example System

In this section we demonstrate the presented approach by the example of a fire alarm system. The system consists of two redundant fire alarm units which may fail stochas-

tically. The hazard to be analysed is the situation when both alarm units are simultaneously in the state "failed", since in this case a fire might break out without being noticed. In order to restart a failed alarm unit a watchdog that periodically checks the alarm units and restarts them if they are in failed state.

4.2 Constructing the SEFT

We start by modelling each of the units as independent components. The alarm units are instances of the component "fire alarm unit" shown in Fig. 6a.

An alarm unit may be running properly or fail stochastically with a failure rate of λ=1/10 hour^{-1}. In order to restart a unit an external trigger is needed. Before running normally, some initialization steps taking a deterministic time of 0.1 hours have to be performed. In order to notice when a unit is out of work, a state output port (depicted by the filled S-triangle) that senses if the unit is running is used in the model. For externally triggering the initialization routine an event input port (the empty E-triangle) is needed. The watchdog, as shown in Fig. 6b, is simply depicted as a component with only one state. The triggering event is produced once every hour and can be connected to other components via an event output port.

Now that the technical units are given, we combine the modelled units to form the SEFT that describes the complete fire alarm system and explains the causal paths that lead to the hazard situation. Figure 7 shows how the fire alarm system is modelled using two instances of the "fire alarm unit" and an instance of the "watchdog" component. The inner structure of the instances is omitted in this view. The watchdog is connected to the event input ports of both alarm units so that it can trigger a restart of a failed unit when necessary. Since the fire hazard is present when both of the redundant alarm units are not working at the same time they are combined with a NOT gate each, which in turn serve as inputs for a state AND gate. The output of the state AND gate represents the hazard situation.

All preconditions for analyzing this model are fulfilled: There are no cycles in the component hierarchy or in any causal relation and all state and event port have been connected to their counterparts. The state output port of the AND gate does not need to be connected since it represents the hazard situation to be analyzed.

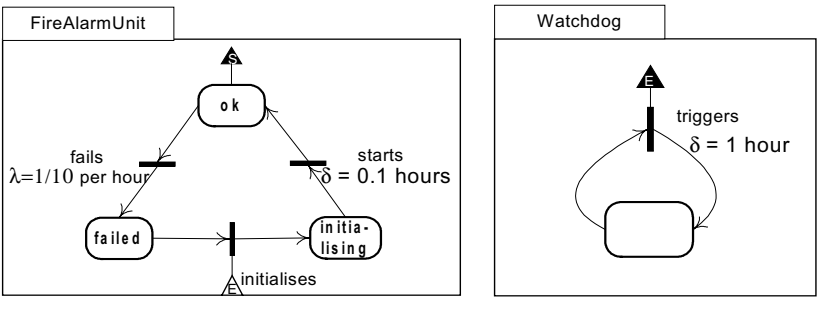

Fig. 6. a) Fire alarm component b) Watchdog component

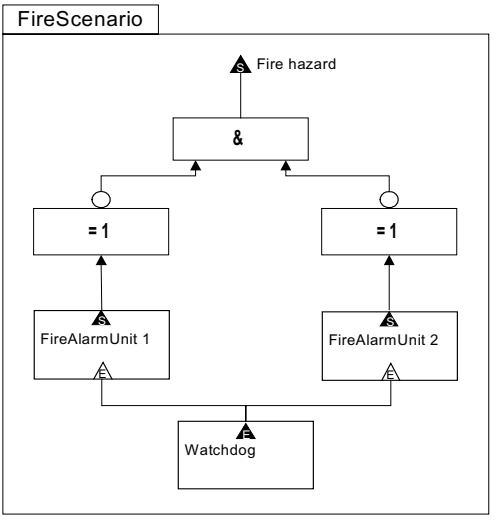

Fig. 7. The fire alarm system

4.3 Translation into DSPN

Before performing the analysis the model has to be translated into a DSPN. For the NOT and AND gates the corresponding DSPNs have already been shown in Fig. 3. The "alarm unit" and "watchdog" components can be translated step by step. Each state is mapped onto a DSPN place, the events that occur stochastically onto exponential transitions, the events occurring after a certain time onto deterministic transitions and externally triggered events onto immediate transitions respectively. The resulting parts have to be combined with the gate nets to form the complete flattened DSPN shown in Fig. 8. The dotted rectangles show the DSPN parts representing the components. The markings were set in a way so that the initial state of the alarm system is conceived: the alarm units are working properly and there is no fire hazard. Note the applied trigger pattern for combining the watchdog and the externally triggered events in the alarm units.

4.4 Analysis of the Flattened DSPN

The DSPN can now be examined using TimeNET. This step as well as the translation into DSPNs must currently be done manually but an export filter to the TimeNET file format will be integrated in our tool ESSaRel. The hazard situation is represented by the place marked with POut which was mapped on the output of the state AND gate. The probability that one token lies in place POut denotes the probability that both alarm units are out of work simultaneously. The corresponding expression in TimeNET is P{#POut=1}.

TimeNET cannot accomplish an analysis of the net, since more than one deterministic transition might be enabled at the same time. A steady-state simulation (con-

tinuous time) with the parameters given in Fig. 8 and a maximum relative error of 1% (a parameter TimeNET uses to specify the desired accuracy of the simulation) returns a resulting probability for the fire hazard as 0.003975. In about 0.4% of the time, both alarm units are not working so that a fire hazard persists.

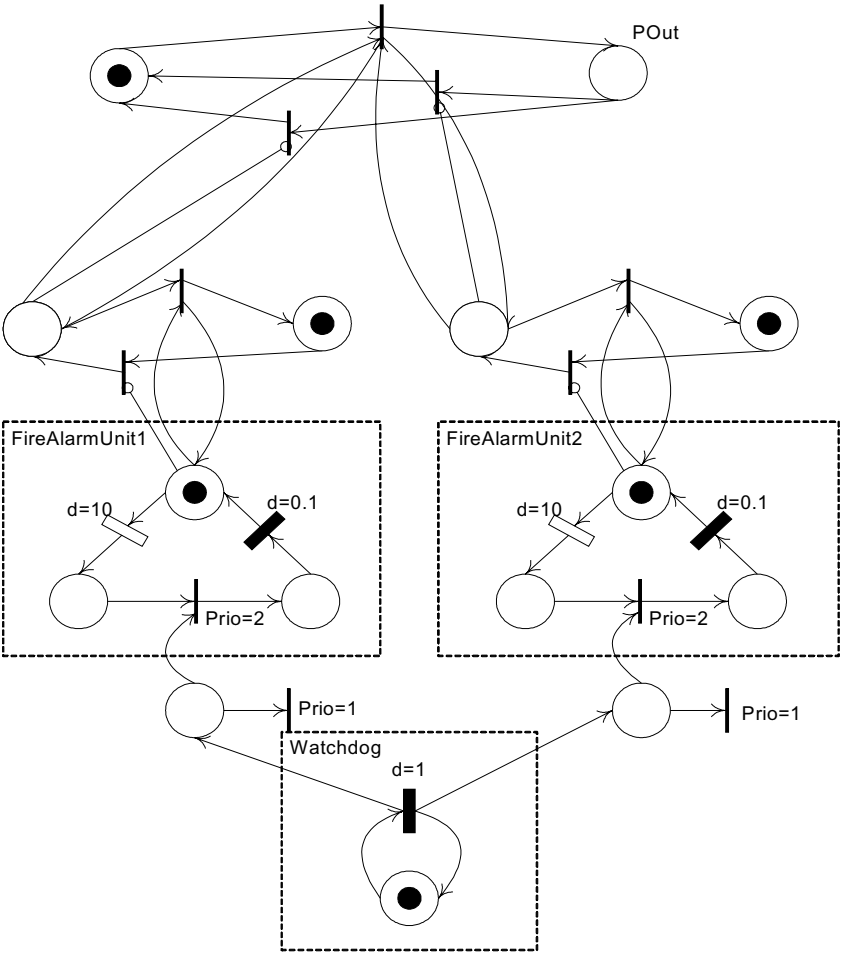

Fig. 8. DSPN of the fire alarm system after flattening

5 Conclusion and Further Research

We have proposed State-Event-Fault Trees as an extension to the FT concept that allows distinguishing states and events. This enables us to integrate finite state models with FTA. In particular we can integrate Statecharts and Markov Chains. This increases expressive power and allows the reuse of design models for safety analysis. We adapt the traditional set of FT gates by introducing typed gates for states and

events. Phenomena like temporal order of events that were not expressible by FTs before can be expressed by SEFTs, because additional gates such as History-AND or Sequential-AND exist. The component concept allows partitioning SEFTs in accordance with the actual technical component structure and to handle complex projects. In summary, the established top-down analysis search for hazard causes provided by FTA can now be combined with powerful and reusable description of embedded system behaviour brought in by the integration of state-based models.

For quantitative probabilistic analysis of SEFTs we translate them component-wise into DSPNs, apply simplifications on component level and merge them to one flat net. Existing Petri Net analysis tools, for instance TimeNET, calculate the measures that correspond to the hazard probabilities in the original SEFT. We have shown the applicability of our method by the example of a fire alarm system. We carried out other small studies to validate the model; larger studies with industrial partners are under preparation.

SEFTs allow a more formal modelling of some gates that are part of standard FTA or later extensions, for instance inhibit, functional dependency, and different kinds of spare gates. We aim at developing new kinds of gates that match frequent structures of safety-critical systems, such as History-AND with a Reset input (for Repair / Restart situations) or History / Sequential-AND with a time parameter that indicates how close both events must occur to each other. We work towards a closer integration of state-based and combinatorial analysis approaches to obtain better analysis performance for subcomponents that can be described by a combinatorial model, similar to the approaches given in [12]. We also plan to do some simplification on component level before integration, exploiting the fact that only a few of the functional states of each component are relevant for safety or reliability considerations.

The ongoing integration of SEFTs into a usable tool is part of our research project ESSaRel. The platform for this integration is the existing CFT tool UWG3 which has been developed in cooperation with Siemens. We have defined the translation and flattening algorithm and are now implementing the export filter TimeNET and an import filter to read models from the CASE tool Rational RoseRT that are mainly based on ROOMcharts. In the end we are working towards an integrated tool chain for safety and reliability analysis of embedded systems.

References

1. Ajmone Marsan, M., Chiola, G.: On Petri nets with deterministic and exponentially distributed firing times. European Workshop on Applications and Theory of Petri Nets 1986. Lecture Notes in Computer Science, volume 266, pages 132-145. Springer 1987
2. Bechta Dugan J., Sullivan, K., Coppit, D.: Developing a low-cost high-quality software tool for dynamic fault tree analysis. Transactions on Reliability, December 1999, pages 49-59
3. Bloomfield, E., Cheng, J.H., Górski, J.: Towards A Common Safety Description Model, Proceedings of the 10th International Conference on Computer Safety, Reliability and Security SAFECOMP'91 (Edited by J. F. Lindeberg), pp. 1-6, Pergamon Press, 1991
4. Bobbio, A., Franceschinis, G., Gaeta, R., Portinale, L.: Exploiting Petri nets to support fault tree based dependability analysis. In: Proc. 8th Int. Workshop on Petri Net and Performance Models (PNPM'99), 8-10 October 1999, Zaragoza, Spain, pages 146-155. 1999

5. Buchacker, K.: Combining Fault Trees and Petri-Nets to Model Safety-critical Systems. In: A. Tentner (Ed.): High Performance Computing 1999. The Society for Computer Simulation International, 1999, pp. 439-444
6. Ciardo, G., Lindemann, C.: Analysis of deterministic and stochastic Petri nets. In Proc. of the Fifth Int. Workshop on Petri Nets and Performance Models (PNPM93), Toulouse, France, Oct. 1993
7. ESSaRel. Embedded Systems Safety and Reliability Analyser. http://www.essarel.de
8. Fenelon, P., McDermid, J.A.: An Integrated Toolset For Software Safety Analysis, Journal of Systems and Software, 21(3), 1993, pp. 279-290
9. German. R., and Mitzlaff, J.: Transient analysis of deterministic and stochastic Petri nets with TimeNET. in Proc. 8th Int. Conf. on Computer Performance Evaluation, Modelling Techniques and Tools and MMB (LNCS 977), Heidelberg, Germany, 1995, pp. 209-223
10. Górski, J.: Extending Safety Analysis Techniques with Formal Semantics, In Technology and Assessment of Safety Critical Systems (Edited by F.J. Redmill and T. Anderson), Springer Verlag, 1994
11. Górski, J., Wardzinski, A.: Timing Aspects of Safety Analysis, in F. Redmill and T. Anderson, Eds.: Safer Systems, Springer Verlag 1997, pp. 231-244
12. Gulati R., Bechta Dugan J.: A modular approach for analyzing static and dynamic fault trees. In Proceedings of the Reliability and Maintainability Symposium, January 1997.
13. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8(3): 231--274, June 1987
14. Hura, G.S., Atwood, J.W.: The Use of Petri Nets to Analyze Coherent Fault Trees. In: IEEE Trans. Reliab. (USA), Vol. 37, No. 5, pages 469-474. 1988
15. Kaiser, B., Liggesmeyer, P., Mäckel, O.: A New Component Concept for Fault Trees. Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03), P. Lindsay & T. Cant, Eds.
16. Schellhorn, G., Thums, A., Reif, W.: Formal Fault Tree Semantics. In: Proceedings of The Sixth World Conference on Integrated Design & Process Technology , Pasadena, CA, July 2002
17. Selic, B., Gullekson, G., Ward P.T:. Real-Time Object-Oriented Modeling, John Wiley & Sons, 1994
18. Vesely, W. E., Goldberg, F. F., Roberts, N. H., Haasl, D. F.: Fault Tree Handbook. U. S. Nuclear Regulatory Commission, NUREG-0492, Washington DC, 1981
19. Zimmermann, A., German, R., Freiheit, J., Hommel, G.: TimeNET 3.0 Tool Description. Int. Conf. on Petri Nets and Performance Models (PNPM'99), Zaragoza, Spain, 1999