Modeling and Automatic Failure Analysis of Safety-Critical Systems Using Extended Safecharts

Yean-Ru Chen[1], Pao-Ann Hsiung[2], and Sao-Jie Chen[1]

[1] Graduate Institute of Electronics Engineering,
National Taiwan University, Taipei, Taiwan−106, ROC
d95943037@ntu.edu.tw, csj@cc.ee.ntu.edu.tw
[2] Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi, Taiwan−621, ROC
hpa@computer.org

Abstract. With the rapid progress in science and technology, we find ubiquitous use of *safety-critical systems* in avionics, consumer electronics, and medical instruments. In such systems, unintentional design faults might result in injury or even death to human beings. To avoid such mishaps, we need to verify safety-critical systems thoroughly, where formal verification techniques such as model checking play a very promising role. Currently, there is practically no automatic technique in formal verification used to formally model system faults and repairs. This work contributes in proposing an extension to the *Safecharts* model, with which faults and repairs can be easily modeled. Moreover, these Safecharts can be directly transformed into semantically equivalent *Extended Timed Automata* models for model checking. That is, after these models were integrated into a model checker, such as our previously proposed *State Graph Manipulators* (SGM) model checker, we can verify safety-critical systems. An application example is run to show the feasibility and benefits of the proposed model-driven verification method for safety-critical systems. As observed, the checking results, such as witnesses of property specifications representing hazards, provide more concrete and useful failure analysis information than the conventional Fault Tree Analysis (FTA).

Keywords: Safety-critical systems, Safecharts, FTO-failure, SO-failure, NC-failure, Effective repair actions, Ineffective repair actions.

1 Introduction

Safety-critical systems are systems whose failure causes damages to its environment. Traditional hazard analysis techniques, such as fault tree analysis (FTA), failure modes and effects analysis (FMEA), failure modes, effects, and criticality analysis (FMECA) have been successfully applied to several different real-world safety-critical systems [12]. Recently, formal verification techniques such as model checking [4] has become a promising verification method due to its automatic analysis capabilities. We propose an extended Safecharts model [5] to model and analyze failures and repairs in safety-critical systems and integrate it into a formal verification tool. Our contributions are three-folds. First, according to the classification of failure modes, we propose an intuitive representation of the faulty states in a safety-critical system using Safecharts model. Second, we transform the Safecharts model into a semantically equivalent *Extended Timed Automata* (ETA) model that can be directly input to model checkers. Finally, a compositional model checker, namely the *State Graph Manipulators* (SGM) [15], is used to not only verify a safety-critical system formally, but also provide automatic failure analysis results to help users rectify that system.

The remaining of this article is organized as follows. Section 2 describes the current state-of-the-art in the verification of safety-critical systems. Section 3 describes our work flow and how failure analysis is performed in model checking. The extension of Safecharts to cover faults and repairs is given in Section 4. Section 5 shows the details of the transformation from Safecharts to ETA. The implementation of our proposed method in the SGM model checker is described in Section 6. An application example is given in Section 7 along with several analysis results. The article is concluded and future research directions are given in Section 8.

2 Related Work

The conventional hazard analysis techniques, such as FTA and FMEA, have been successfully applied to several different real-world safety-critical systems. Nevertheless, a major limitation of hazard analysis is that phenomena unknown to the analysts are not covered in the analysis and thus hazards related to these phenomena are not foreseen. Safety-critical systems are getting more and more complex and thus there is a trend to use methods [3] that are more automatic and exhaustive than hazard analysis, for example model checking.

The verification of safety-critical systems using formal techniques is not new [12]. However, as noted by Leveson [12], this approach is inadequate because in the system models we are assuming that all the components do not fail and the system is proved to be safe under this assumption. However, the assumption is not always valid, so transforming each hazard into a formal property for verification as in [9] is not sufficient. As described in the following, our work on using Safecharts to verify safety-critical systems contributes in several ways to the state-of-the-art in formal verification.

1. The Unified Modeling Language (UML) is an industry de facto standard for model-driven architecture design. Safecharts, being an extension of the UML Statecharts, blends naturally with the semantics of other UML diagrams for the design of safety-critical systems. The work described in this article automatically transforms Safecharts into the timed automata model which can be accepted by conventional model checkers.
2. The extended Safecharts model allows and requires explicit modeling of component failures and repairs within a newly proposed failure layer in the model. This is very helpful not only for the safety design engineers but also for the safety verification engineers. Through its unique features of risk states, transition priorities, and component failure and repair modeling artifacts, Safecharts can be successfully used for verifying safety-critical systems.

3. The model checking paradigm helps generate witnesses to the satisfaction of properties that provide more accurate and informative safety analysis results than conventional methods such as FTA.

3 Automatic Failure Analysis

The failure analysis performed in a model checker is as proposed and illustrated in Figure 1. A safety-critical system is first analyzed for all possible failures and corresponding repairs. Safecharts are then used to model the functional, safety, and failure characteristics of the system, according to the description in Section 4. As described in Section 5, these Safechart models are then automatically transformed into equivalent ETA models that are input to the SGM model checker as system models. Each hazard is then represented as a temporal logic property. On model checking the ETA against the properties, we obtain witnesses to the hazard properties that represent how the system might face a hazard. The witnesses are system traces that contain more information than traditional minimum cut results from FTA.

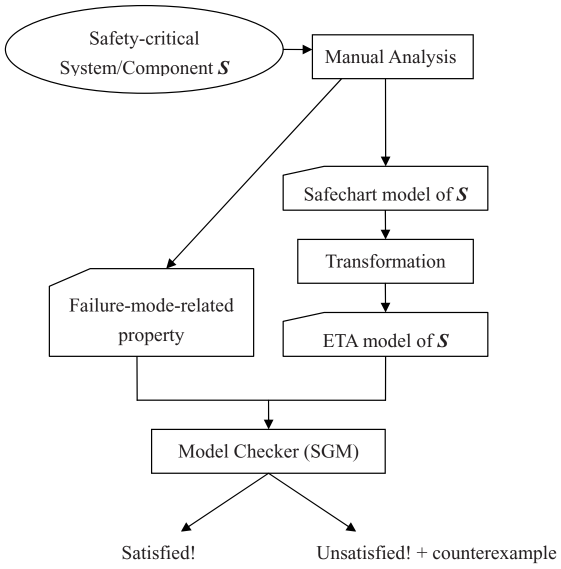
Safety-critical System/Component S
Manual Analysis
Safechart model of S
Transformation
Failure-mode-related property
ETA model of S
Model Checker (SGM)
Satisfied!
Unsatisfied! + counterexample

Fig. 1. Work flow of automatic failure analysis in model checking

4 Extending Safecharts

The main focus of the original Safecharts [5], as defined in Definition 1, was on proposing functional and safety layers. Failures are due to faults, but not all faults result in failures. In this work, we focus on failures and extend the original Safecharts with a more

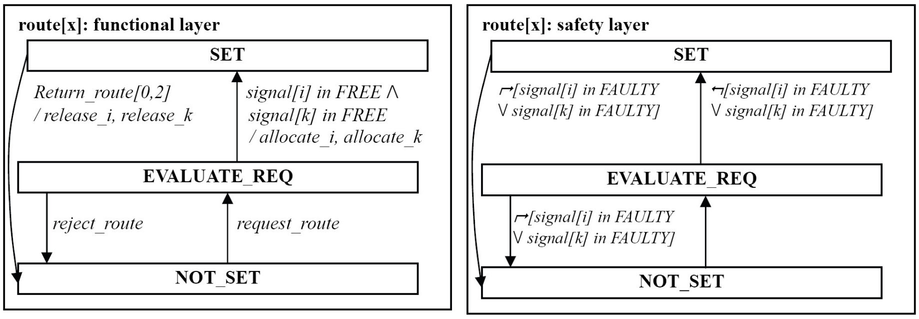
route[x]: functional layer
SET
Return_route[0,2] / release_i, release_k
signal[i] in FREE ∧ signal[k] in FREE / allocate_i, allocate_k
EVALUATE_REQ
reject_route
request_route
NOT_SET
route[x]: safety layer
SET
↱[signal[i] in FAULTY ∨ signal[k] in FAULTY]
↰[signal[i] in FAULTY ∨ signal[k] in FAULTY]
EVALUATE_REQ
↱[signal[i] in FAULTY ∨ signal[k] in FAULTY]
NOT_SET

Fig. 2. Safechart for route[x] with functional and safety layers

comprehensive modeling capability for different types of failures and repairs, in a new and explicit *failure layer*. In this extension, we use a generic classification of failures and repairs to help safety-critical system designers categorize the possible failures and repair actions in the system to be modeled and verified.

4.1 Safecharts

Figure 2 shows the *functional* and *safety* layers of a Safecharts model route[x] for setting a route of a railway system. The functional layer specifies the normal functions of requesting and setting or rejecting a route. The safety layer enforces the safety restrictions for setting or unsetting a route. Notations ↰ and ↱ in the safety layer will be defined in Definition 1, which, respectively, restricts setting a route or enforces the release of a route when any of the signals in that route is faulty. Recognizing the possibility of gaps and inaccuracies in safety analysis, Safecharts do not permit transitions between states with unknown relative risk levels [14].

Definition 1. *Safecharts*
Given two comparable states s_1 and s_2, a risk ordering relation $\preccurlyeq$ specifies their relative risk levels, that is $s_1 \preccurlyeq s_2$ specifies s_1 is safer then s_2. Transition labels in Safecharts have an extended form: $e[fcond]/a[l,u)\Psi[G]$ where e, $fcond$, and a are the conventional Statechart event, transition guard, and action, respectively, and $/$ separates the transition guard from the transition action. The time interval $[l,u)$ is a real-time constraint on a transition t and imposes the condition that t does not execute until at least l time units have elapsed since it most recently became enabled and must execute strictly within u time units. The expression $\Psi[G]$ is a safety enforcement on the transition execution and is determined by the safety clause G. The safety clause G is a predicate, which specifies the conditions under which a given transition t must, or must not, execute. Ψ is a binary valued constant, signifying one of the following enforcement values:

– *Prohibition enforcement value*, denoted by ↰. Given a transition label of the form ↰ $[G]$, it signifies that the transition is forbidden to execute as long as G holds.
– *Mandatory enforcement value*, denoted by ↱. Given a transition label of the form $[l,u)$ ↱ $[G]$, it indicates that whenever G holds the transition is forced to execute within the time interval $[l,u)$, even in the absence of a triggering event.

4.2 Failure Modes

The failure modes of safety-critical systems can be classified into three types as defined in the following [2].

- *Fail-To-Operate* (FTO-failure): A system does not respond correctly to an abnormal operating condition. This is a critical failure.
- *Spurious Operation* (SO-failure): A system initiates an automatic unexpected action without the presence of an abnormal operating condition. This is also a critical failure.
- *Non-critical* (NC-failure): In this case, maintenance of a system is required even though the main service of the system has been preserved, that is it is neither an FTO-failure nor an SO-failure in the safety-critical system. With this type of failure, a system can still work, but its performance may be poorer than expected. We assume that when a system has an NC-failure, it must be repaired before other critical failures can occur.

The representation of failure modes in Safecharts is given in Section 4.4.

Example: Take an automatic electric lamp as an example, which is supposed to be turned on automatically within three seconds when the room is too dark, however if it fails, then an FTO-failure occurs. In addition, if the lamp is turned on unexpectedly when the room is bright, it is said that an SO-failure occurs. However, if the lamp blinks after being turned on, we take this as an NC-failure.

After an FTO- or SO-failure, as shown in Figure 3 and Figure 4, respectively, when more than one intermediate repair actions are needed to return to normal state, then we assume that the component or system enters an NC-failure mode, which is an intermediate state between an FTO- or SO-failure and a normal state.

A component FTO- or SO-failure can lead to a system FTO- or SO-failure unless this is prevented by fault-removal or fault-prevention techniques [2]. In this work, we assume that each component is under the control of its controller and the controllers could be taken as error-free. We do not discuss controller fault tolerance techniques in this work.

4.3 Safety Repair Actions

A system may return to normal operating condition after a failure is either automatically or manually repaired. Repair actions of safety-critical systems can be classified into two types as follows.

- *Effective repair actions*: A failure is completely eliminated after an effective repair action is taken. We denote the set of all effective repair actions as $U = \{\mu\}$.
- *Ineffective repair actions*: A failure is not completely eliminated, but its severity may be reduced after ineffective repair actions are taken. These repair actions could be taken as intermediate safety actions. Let $U' = \{\mu'\}$ denote the set of all ineffective repair actions. Moreover, $U \cap U' = \phi$. □

For example, in an automatic electric lamp system, if the electric lamp is not turned on automatically within three seconds when the room is too dark, an FTO-failure occurs. We should turn it on manually to eliminate this failure. This repair action is effective if the electric lamp becomes bright after we turn it on. Otherwise, it is an ineffective repair action.

4.4 Representation of Failures and Repairs in Extended Safecharts

Using some conventional safety analysis technique, a designer can identify all the possible failures and their effects in a system to be modeled and verified. Further, based on the classifications of failures and of repair actions, as described in Sections 4.2 and 4.3, respectively, a system designer can categorize all the possible failures and corresponding repair actions of the system. We now give the representations of these different failures and repair actions in the extended Safecharts model.

Recall from Definition 1, in Safecharts, the transition label $e[fcond]/a$ appears in the *functional* layer, while $[l{,}u)\Psi[G]$ may appear in the *safety* layer. In this work, as mentioned before, we propose a new *failure layer*, and three new symbols in Safecharts to represent failure conditions, namely, *inoperable* ($\overline{\Rsh}$), *spurious* ($\overline{\Lsh}$), and *non-critical* ($\looparrowleft$) failures, which are modeled in the failure layer of extended Safecharts.

In this work, we found that there is an interesting coupling between safety conditions and failure conditions. Whenever a mandatory condition ($[l, u) \Rsh [G]$) is violated, an FTO-failure ($[u, \infty)\overline{\Rsh}[G]$) occurs. Whenever a prohibitory condition ($\Lsh [G]$) is violated, an SO-failure ($\overline{\Lsh}[G]$) occurs. As far as the non-critical ($\looparrowleft$) condition is concerned, it can be used in more flexible ways than the other two critical failure conditions. System designers can model an NC-failure for a component or a system wherever it is required.

Figures 3, 4, and 5 show the representation of different failure modes and their corresponding repair actions for a component or system modeled in Safecharts. Each component or system may be modeled with one or more failure modes. The condition and assignment label $e[fcond]/a[l, u)\Psi[G]$ on a transition t, from the original Safecharts (Definition 1), is now extended to cover both safety conditions as well as failure conditions, as classified in the following.

1. *Safety conditions*: According to [5], safety conditions are modeled into the functional and safety layers.
   - $e[fcond]/a$: This is just a normal functional transition without any safety clause.
   - $e[fcond]/a \Lsh [G]$: There is *prohibition* enforcement value on a transition t. It signifies that the transition t is forbidden to execute as long as G holds.
   - $e[fcond]/a[l, u) \Rsh [G]$: There is *mandatory* enforcement value on a transition t. It signifies that the transition is forced to execute within $[l{,}u)$ whenever G holds.
2. *Failure conditions*: Failure conditions are modeled into the failure layer of Safecharts.
   - $e[fcond]/a[u, \infty)\overline{\Rsh}[G]$: Transition t represents an FTO-failure. It signifies that under the condition G, if a corresponding mandatory transition t' with label

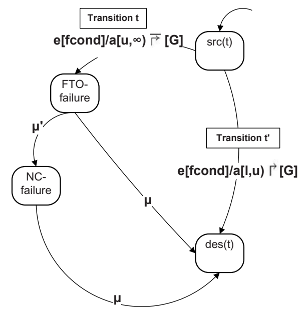
Transition t
e[fcond]/a[u,∞) $\overline{\Rsh}$ [G]
src(t)
FTO-failure
μ'
NC-failure
Transition t'
e[fcond]/a[l,u) $\Rsh$ [G]
μ
des(t)
μ

Fig. 3. FTO-failure in Safechart

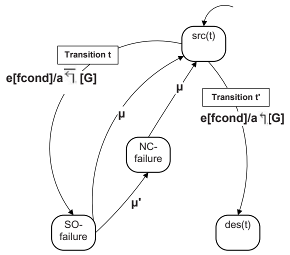

Fig. 4. SO-failure in Safechart

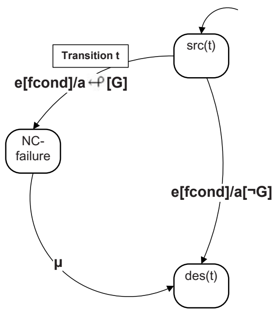

Fig. 5. NC-failure in Safechart

$e[fcond]/a[l,u) \Rsh [G]$ is not executed within the time interval $[l, u)$, then there is a critical FTO-failure.

- $e[fcond]/a\overline{\Lsh}[G]$: Transition t represents an SO-failure. It signifies that under the condition G, if a corresponding prohibitory transition t' with label $e[fcond]/a \Lsh [G]$ is unexpectedly executed, then there is a critical SO-failure.
- $e[fcond]/a \hookleftarrow [G]$: Transition t represents an NC-failure. The condition G means that the main service of the safety-critical system is not supported as good as expected. It signifies that repair actions are needed when G holds.

Here, we assume that when a component or system has failed, the failure must be repaired before another failure can occur.

Let us take the automatic electric lamp system as an example again to illustrate the three failure modes represented by the above described transitions. Table 1 shows the conditions and assignments for normal safety operation and for the three possible failure modes of the electric lamp.

In Figure 3, the transition from *src(t)* to *des(t)* has mandatory evaluation. By definition, the transition is supposed to be executed before the deadline u. If it fails, by the failure definitions, we can specify that an FTO-failure has occurred. Similarly, we can find an SO-failure may occur in Figure 4. Note that there is an important concept here: the transition from *src(t)* to *des(t)* is executed, therefore, in a real system, the *des(t)* state appears to be the same as the SO-failure state. However, in modeling, we cannot model the contradictory conditions of taking and not taking the safety transitions in the same mode. In Safecharts modeling, we have to specify these conditions into two different modes. When the safety transitions are executed under the expected semantics, then the *des(t)* state is a correct final destination. Otherwise, this state represents a failure state. As shown in Figure 5, we allow more flexible ways for system designers to model non-critical failures, because non-critical failures are defined case by case. In our work, we also assume that each type of failure could be recovered by some repair actions. If not, there might be too many dead states in the model. Nevertheless, we do not focus on how to deal with the dead states, except using them to specify properties representing system hazards.

Table 1. Safecharts Normal operations and possible failure modes for the Electric Lamp System

Type of condition (Transition Label)	**Safecharts Transition Label Descriptions**
Mandatory $(e[fcond]/a[l,u) \Rsh [G])$	e: someone enters the room
	$fcond$: lamp is normally functioning
	a: automatically turn on the lamp
	$[l,u) = [0,3)$
	G: insufficient light indoor
Prohibitory $(e[fcond]/a \Lsh [G])$	e: someone enters the room
	$fcond$: lamp is normally functioning
	a: automatically turn on the lamp
	G: sufficient light indoor
FTO $(e[fcond]/a[u,\infty)\overline{\Rsh}[G])$	electric lamp does not operate normally by the deadline time u.
SO $(e[fcond]/a\overline{\Lsh}[G])$	electric lamp automatically turns on when there is sufficient light indoor.
NC $(e[fcond]/a \looparrowleft [G])$	e: someone enters the room
	$fcond$: true
	a: record non-critical problem
	G: the main service of the lamp is not supported as good as expected, for example, illumination is poor.

5 Transformation from Safecharts to Extended Timed Automata

The user-given Safecharts are automatically transformed into extended timed automata (ETA) models [8], which are simply timed automata [1] with discrete variables and synchronization labels. We first show in detail how system designers might model their safety-critical system in Safecharts, and then how the model is transformed into ETA for model checking.

5.1 Transformation from Safecharts to ETA

To analyze failures automatically in model checking, one of the primary goals is to model check the Safecharts model of a component or a system. However, Safecharts cannot be accepted as system model input by most model checkers, most of which accept only flat automata models. For example, extended timed automata (ETA) can be accepted by SGM. The three Safecharts layers, namedly safety, functional, and failure, must also be transformed into equivalent modeling constructs in ETA and specified as properties for verification.

There are three types of states in Safecharts: OR, AND, and BASIC. An OR-state or an AND-state, consists generally of two or more substates. All the substates in an AND-state are active simultaneously, while an OR-state is in exactly one of its substates.

A BASIC-state is represented simply by an ETA mode. The translations for OR-states and AND-states are performed as described in [11].

Before we translate Safecharts to ETA, we need to introduce three different types of transition urgency semantics [10]:

1. *Eager Evaluation* (ε): Execute the transition as soon as possible, *i.e.*, as soon as a guard is enabled. Time cannot progress as soon as a guard is enabled.
2. *Delayable Evaluation* (δ): Can put off execution until the last moment the guard is true. So time cannot progress beyond the *falling edge* of a guard, except when there is another enabled transition outgoing from the same state.
3. *Lazy Evaluation* (λ): The transition may or may not be executed.

The transition condition and assignment label $e[fcond]/a[l,u)\Psi[G]$ as described in Section 4.4 can be translated to the following equivalent semantics in ETA.

- $e[fcond]/a$: There is no safety clause on a transition in Safecharts, thus we can simply transform it to a similar transition in ETA. We have translated it in [6].
- $e[fcond]/a \Lsh [G]$: There is *prohibition* enforcement value on a transition t. It signifies that the transition t is forbidden to execute as long as G holds. We also have performed this transformation in [6].
- $e[fcond]/a[l,u) \Rsh [G]$: There is *mandatory* enforcement value on a transition t. It signifies that the transition is forced to execute within $[l,u)$ whenever G holds. The transformation to ETA was done in [6].
- $e[fcond]/a[u,\infty)\overline{\Rsh}[G]$: As shown in Figure 6, the transformations of the safety and the functional layers are same as those for a transition t' with the mandatory enforcement value, that is, $e[fcond]/a[l,u) \Rsh [G]$. A transition from some state *src(t)* to an *FTO-failure* state in Safecharts, as depicted in Figure 3, will be translated into a transition labeled with $(timer \geqslant u)^{\epsilon}$ from state *translator(t)* to an FTO-failure state in ETA, where $translator(t)$ is a state generated during the transformation of mandatory evaluation [6].
- $e[fcond]/a\overline{\Lsh}[G]$: As shown in Figure 7, the transformations of the safety and the functional layers are same as those for a transition t' with the prohibition enforcement value, that is, $e[fcond]/a \Lsh [G]$. The transition from some state *src(t)* to an *SO-failure* state in Safecharts, as depicted in Figure 4, will be translated into a transition labeled with $(e[fcond \wedge G]/a)^{\lambda}$ from state *src(t)* to an SO-failure state in ETA.
- $e[fcond]/a \looparrowleft [G]$: As shown in Figure 8, we translate a transition from some state *src(t)* to an *NC-failure* state in Safecharts, as depicted in Figure 5, into a transition labeled with $(e[fcond \wedge G]/a)^{\lambda}$ from state *src(t)* to an NC-failure state in ETA.

As shown in Figures 6, 7, and 8, the transitions representing repair actions in Safecharts are translated into similar corresponding transitions, without any special transformation. All the proposed transformations from the Safecharts layers to ETA modeling artifacts are trivially equivalent in semantics.

6 Implementation

The proposed model extensions in Safecharts for explicit failure representation and the related techniques for supporting failure analysis have all been integrated into the SGM

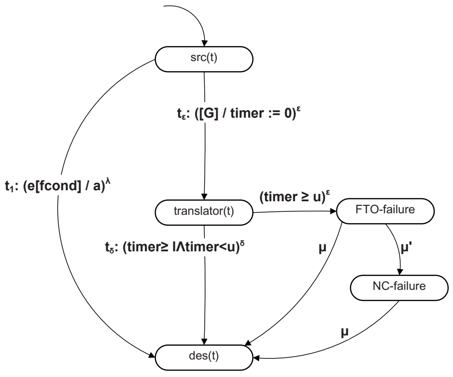

Fig. 6. FTO-failure in ETA

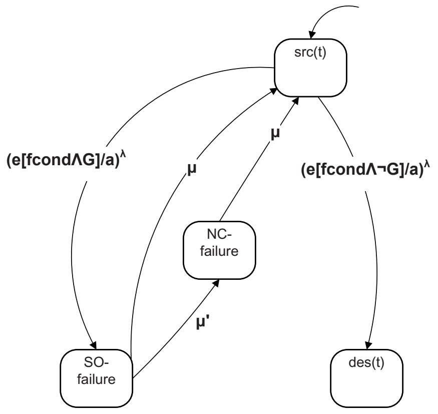

Fig. 7. SO-failure in ETA

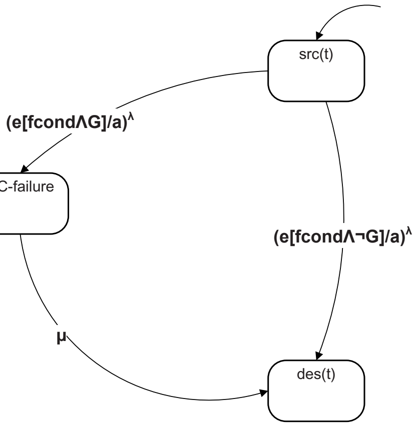

Fig. 8. NC-failure in ETA

model checker, on which we can perform verification and failure analysis. Mainly, what we had done are listed in the following.

- Extend the semantics of Safecharts to include the failure layer,
- Flatten the extended Safecharts into ETA models, and
- Transform all safety and failure modeling artifacts into equivalent constructs in the ETA models, which include the following.
  - Transition priority calculation from Safechart risk bands,
  - Support for prioritized transitions, and
  - Support for urgent transitions.

Details of the implementation for supporting prioritized transitions in the SGM model checker were given in [13]. The main issue is priority results in non-convex clock zones which cannot be represented by a single difference bound matrix (DBM), a data-structure for representing time in a model checker. We solved this issue by proposing an optimal zone partitioning algorithm.

In our work, we also implemented support for urgent transitions in the SGM model checker by proposing a novel zone capping operation that restricts the clock zone in a mode (symbolic set of states) [7]. Due to page-limit, this part of the work is out-of-scope here and will not be described in details.

7 Application Example

To illustrate how the failure-extended Safecharts model aids in verifying safety-critical systems, we use an automatic water sprinkler system that is found in almost all high-rise apartments and garages for safety from fires. In such a system, the most important functionality is that when the sensors detect an abnormally high temperature degree, the controller sends a command to the sprinkler, which is supposed to start sprinkling water. We assume that the controller polls the temperature sensor once every second. We have used the extended Safecharts model to specify the relations between the failure state and the normal operating state for the sprinkler system, as shown in Figure 9, where *clock* is the triggering event for the system, and it arrives once every second. Totally, four failure states are specified, of which two are FTO-failures and two are SO-failures.

To analyze the possible failures in this sprinkler system, we have to first introduce its design. When the sensors detect an abnormally high temperature degree, it asserts a signal F, which is polled by the controller. If the sprinkler system is in the *Non-sprinkling* state, it is supposed to start sprinkling water within two seconds after the assertion of signal F. Otherwise, an FTO-failure occurs. When the sprinkler system is in the *Sprinkling* state, if the sensors detect a normal temperature degree, then the system is supposed to stop sprinkling within three seconds. Otherwise, there is also an FTO-failure. It is prohibited that the sprinkler system transits from the *Non-sprinkling* state to the *Sprinkling* state without an asserted signal F. It is also required in the design that the sprinkler system is prohibited from transiting from the *Sprinkling* state to the *Non-sprinkling* state while the signal F is asserted. Otherwise, an SO-failure occurs.

In Figure 9, $F = 1$ represents abnormally high temperature, $F = 0$ represents normal temperature, $R := 0$ is the action to stop sprinkling, and $R := 1$ is the action to start sprinkling. The sprinkler system is quite simple, thus only two effective repair actions, namely stop sprinkling and start sprinkling, are modeled in this system. There are no ineffective repairs and no NC-failure in this system.

We transformed the Safecharts model for this system automatically and generated the ETA model as shown in Figure 10, which was then input to the SGM model checker for verification and failure analysis. As we can see, the ETA model in Figure 10 is much more complex than the Safecharts model in Figure 9. Table 2 shows the transformation results for the sprinkler system. Even for this small example, the Safecharts model gives a reduction of 25% in the number of states and 25% in the number of transitions, compared to the generated ETA model. For larger and more complex systems, our proposed Safecharts with failure extensions will give greater benefits in terms of reduced sizes of models, compared to the more complex ETA models. The designers can thus focus

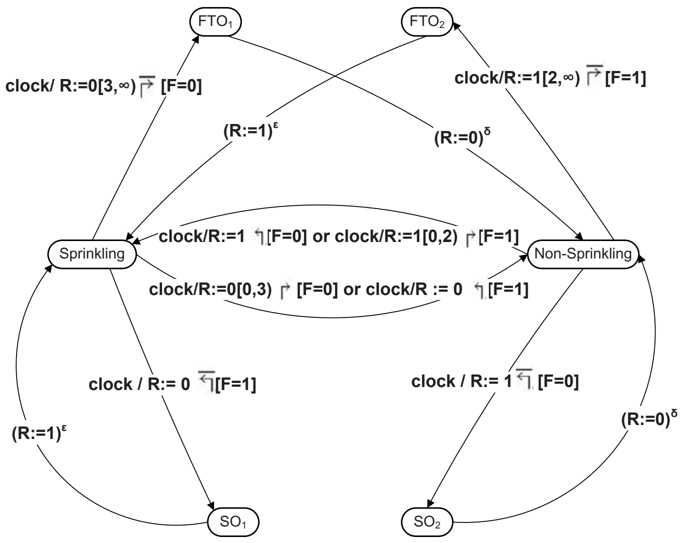

Fig. 9. Sprinkler System Controller in Safecharts

their attention and time to failure identification, classification, and safety analysis, rather than spending a lot of time and efforts in modeling them.

For this sprinkler system, we specified four different properties in CTL to analyze the four failures in the Safecharts model, namely, FTO_1, FTO_2, SO_1, and SO_2, as follows.

$$EF(mode = HAZARD), \text{where } HAZARD \in \{FTO_1, FTO_2, SO_1, SO_2\}$$

On model checking, the sprinkler system model satisfied all of the above properties. Witnesses to the satisfaction of these properties, generated by SGM, helped us in analyzing how the system would be faced with such hazards. For example, the witness $\langle(\textit{Non-Sprinkling}, F = 0), (\textit{Non-Sprinkling}, F = 1), (\textit{Non-Sprinkling}, F = 1 \wedge timer \geqq 2), (FTO_2)\rangle$, shows that one possible way in which the sprinkler system could face the FTO_2 failure is when the system is not sprinkling (*Non-Sprinkling*), the temperature becomes abnormally high ($F = 1$), and the timer has expired ($timer \geqq 2$). This property witness generated from a model checker contains more information than the minimum cut generated from *Fault Tree Analysis* (FTA) [12], which is a conventional and widely used analysis method for safety-critical systems. A minimum cut only

Table 2. Transformation Results for the Sprinkler System Example

	Numbers of Modes	Numbers of Transitions
Safecharts	6 (-25%)	12 (-25%)
ETA	8	16

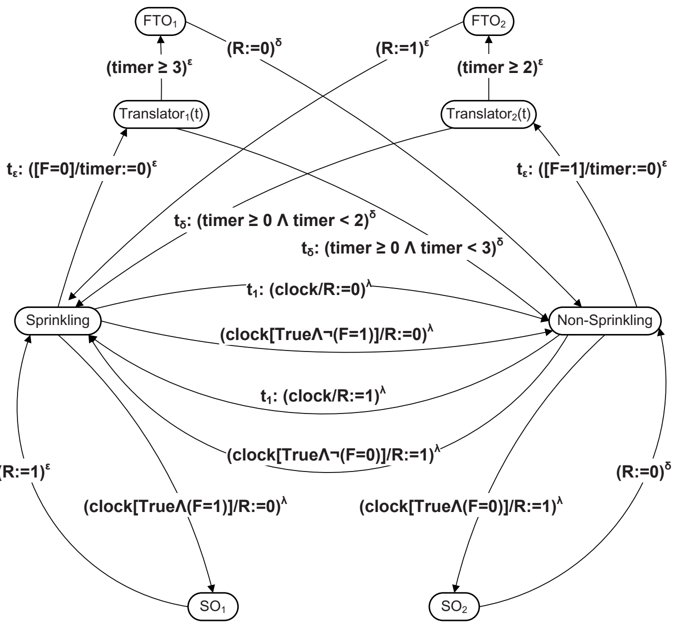

Fig. 10. Sprinkler System Controller in ETA

shows the logical combination of different faults or failures in a system that results in some hazard, however it cannot show the temporal sequencing of the faults that leads to a hazard. The same logical combination could have different temporal sequences, of which only some might result in a hazard and others would not. However, property witnesses show the exact computation run that would result in a hazard. Hence, there is more accurate information in a witness compared to that in a minimum cut.

Besides automatically generating a hazard witness for a single component failure, model checking our failure-extended Safecharts can also be used to verify a common safety-critical system property that specifies no single component failure results in a system failure, where a system failure is defined as the simultaneous occurrence of two component failures. We can thus check if the following property is satisfied.

$$AG\neg \bigvee_{i,j} [(mode(C_i) = HAZARD) \wedge (mode(C_j) = HAZARD)] \qquad (1)$$

where $HAZARD$ is either an FTO-failure or SO-failure and C_i is the ith component.

8 Conclusion

Nowadays, safety-critical systems are becoming more and more pervasive in our daily lives. To reduce the probability of tragedy, we must use a formal and accurate methodology to verify whether a safety-critical system is safe or not. We have proposed a formal method to verify safety-critical systems based on the Safecharts model and model checking paradigm. Our methodology can be applied widely to safety-critical systems with a model-driven architecture. Through examples, we have shown the benefits of the proposed verification method and system model. We hope our methodology can have some real contribution in making the world a safer place to live in.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Bodsberg, L., Hokstad, P.: A system approach to reliability and life-cycle-cost of process safety systems. IEEE Transactions on Reliability 44(2), 179–186 (1995)
3. Bozzano, M., Villafiorita, A.: Improving system reliability via model checking: the FSAP/NuSMV-SA safety analysis platform. In: Anderson, S., Felici, M., Littlewood, B. (eds.) SAFECOMP 2003. LNCS, vol. 2788, pp. 49–62. Springer, Heidelberg (2003)
4. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
5. Dammag, H., Nissanke, N.: Safecharts for specifying and designing safety critical systems. In: Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems, October 1999, pp. 78–87. IEEE Computer Society Press, Los Alamitos (1999)
6. Hsiung, P.-A., Chen, Y.-R., Lin, Y.-H.: Model checking safety-critical systems using Safecharts. IEEE Transactions on Computers 56(5), 692–705 (May 2007)
7. Hsiung, P.-A., Lin, S.-W., Chen, Y.-R., Huang, C.-H., Yeh, J.-J., Sun, H.-Y., Lin, C.-S., Liao, H.-W.: Model checking timed systems with urgencies. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 67–81. Springer, Heidelberg (2006)
8. Hsiung, P.-A., Wang, F.: A state-graph manipulator tool for real-time system specification and verification. In: Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA), pp. 181–188 (1998)
9. Johnson, M.E.: Model checking safety properties of servo-loop control systems. In: Proceedings of the International Conference on Dependable Systems and Networks, pp. 45–50. IEEE Computer Society Press, Los Alamitos (2002)
10. Gößler, G., Altisen, K., Sifakis, J.: Scheduler modeling based on the controller synthesis paradigm. Real-Time Systems 23, 55–84 (2002)
11. Lavazza, L. (ed.): A methodology for formalizing concepts underlying the DESS notation. In: ITEA (2001)
12. Leveson, N.G.: Safeware: System Safety and Computers. Addison-Wesley, Reading (1995)
13. Lin, S.-W., Hsiung, P.-A., Huang, C.-H., Chen, Y.-R.: Model checking prioritized timed automata. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 370–384. Springer, Heidelberg (2005)
14. Nissanke, N., Dammag, H.: Risk bands - a novel feature of Safecharts. In: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE), October 2000, pp. 293–301 (2000)
15. Wang, F., Hsiung, P.-A.: Efficient and user-friendly verification. IEEE Transactions on Computers 51(1), 61–83 (2002)