Model Checking Safety-Critical Systems Using Safecharts

Pao-Ann Hsiung, *Member*, *IEEE*, Yean-Ru Chen, and Yen-Hung Lin

Abstract—With rapid developments in science and technology, we now see the ubiquitous use of different types of *safety-critical systems* in our daily lives such as in avionics, consumer electronics, and medical systems. In such systems, unintentional design faults might result in injury or even death to human beings. To make sure that safety-critical systems are really safe, there is a need to verify them formally. However, the verification of such systems is getting more and more difficult because designs are becoming very complex. To cope with high design complexity, currently, model-driven architecture design is becoming a well-accepted trend. However, existing methods of testing and standards conformance are restricted to implementation code, so they do not fit very well with model-based approaches. To bridge this gap, we propose a model-based formal verification technique for safety-critical systems. In this work, the model-checking paradigm is applied to the Safecharts model, which was used for modeling but not yet used for verification. Our contributions listed are as follows: First, the safety constraints in Safecharts are mapped to semantic equivalents in timed automata for verification. Second, the theory for safety constraint verification is proven and implemented in a compositional model checker (that is, the *State-Graph Manipulator* (SGM)). Third, prioritized and urgent transitions are implemented in SGM to model the risk semantics in Safecharts. Finally, it is shown that the priority-based approach to mutual exclusion of resource usage in the original Safecharts is unsafe and corresponding solutions are proposed here. Application examples show the feasibility and benefits of the proposed model-driven verification of safety-critical systems.

Index Terms—Safety-critical systems, model checking, Safecharts, extended timed automaton.

1 Introduction

Safety-critical systems are systems whose failure most probably results in the tragic loss of human life or damage to human property. There are numerous examples of such mishaps. The accident at the Three Mile Island (TMI) nuclear power plant in Pennsylvania on 28 March 1979 is just one unfortunate example [19]. Moreover, as time goes on, we are becoming more and more dependent on cars, airplanes, rapid transit systems, medical facilities, and consumer electronics, which are all safety-critical systems. When some of them malfunction or fault, a tragedy is inevitable. The natural question here is should we use these systems without a very high confidence in their safety? Obviously, the answer is negative in most cases. That is why we need some methodology to exhaustively verify safety-critical systems.

Traditional verification methods such as simulation and testing can only prove the presence of faults and not their absence. Some methods such as fault-based testing and semiformal verification that integrates model checking and testing can prove the absence of prespecified faults. Simulation and testing [26] are both required before a system is deployed in the field. Although simulation is performed on an abstract model of a system, testing is performed on the actual product. In the case of hardware circuits, simulation is performed on the design of the circuit, whereas testing is performed on the fabricated circuit itself. In both cases, these methods typically inject signals at certain points in the system and observe the resulting signals at other points. For software, simulation and testing usually involve providing certain inputs and observing the corresponding outputs. These methods can be a cost-efficient way to find many errors. However, checking all of the possible interactions and potential pitfalls using simulation and testing techniques is rarely possible. Conventionally, safety-critical systems are validated through standards conformance and code testing. Using such verification methods for safety-critical systems cannot provide the desired 100 percent confidence in system correctness.

In contrast to the traditional verification methods, formal verification is exhaustive. Further, unlike simulation, formal verification does not require any test benches or stimuli for triggering a system. More precisely, formal verification is a mathematical way of proving that a system satisfies a set of properties. *Formal verification* methods such as model checking [5], [6], [25] have been taken seriously in the past few years by several large hardware and software design companies such as Intel, IBM, Motorola, and Microsoft, which goes to show the importance and practicality of such methods for real-time embedded systems and system-on-chip (SoC) designs. For the above reasons, we will thus employ a widely popular formal verification method called *model checking* for the verification of safety-critical systems that are formally modeled.

- *P.-A. Hsiung and Y.-R. Chen are with the Department of Computer Science and Information Engineering, National Chung Cheng University, 168, University Road, Min-Hsiung, Chiayi, Taiwan-621, ROC. E-mail: pahsiung@cs.ccu.edu.tw, d95943037@ntu.edu.tw.*
- *Y.H. Lin is with the National Chiao Tung University, Hsinchu, Taiwan-704, ROC. E-mail: yenhung.cs94g@nctu.edu.tw.*

Manuscript received 22 Mar. 2006; accepted 1 Nov. 2006; published online 13 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0110-0306.
Digital Object Identifier no. 10.1109/TC.2007.1021.

Model checking [5], [6], [25] is a technique for verifying finite-state concurrent systems. One benefit of this restriction is that verification can be performed automatically. The procedure normally uses an exhaustive search of the state space of a system to determine if some specification is true or not. Given sufficient resources, the procedure will always terminate with a *yes/no* answer. Moreover, it can be implemented by algorithms with reasonable efficiency which can be run on moderate-sized machines. In case of a negative result, the user is often provided with a counterexample.

Our safety-critical system model and its model-checking procedures are implemented in the *State-Graph Manipulator* (SGM) model checker [27], which is a model checker for real-time systems modeled by a set of extended timed automata (ETA).

In the course of developing a model-based verification method for safety-critical systems, several issues are encountered, as detailed in the following: First and foremost, we need to decide how to model safety-critical systems. Our decision is to adopt Safecharts [7] as system models. Safecharts are a variant of Statecharts, especially for use in the specification and the design of safety-critical systems. The objective of the model is to provide a sharper focus on safety issues and a systematic approach to deal with them. This is achieved in Safecharts by making a clear *separation* between *functional* and *safety* requirements. Other issues encountered in designing the formal verification methodology for model-based safety-critical systems are listed as follows:

- How can Safecharts be transformed into a semantically equivalent *ETA* model that can be accepted by traditional model checkers? How can the transformation preserve the safety semantics in Safecharts?
- What are the properties that must be specified for model-checking Safecharts?
- Basic states in Safecharts have a risk relation with each other specifying the comparative risk/safety levels. Safety nondeterminism allows only safe transitions between states with known risk levels in Safecharts. How do we represent such information in ETA for model checking?
- The current semantics of Safecharts state that mutual exclusion of resource usages can be achieved through priority. This is clearly insufficient as priorities cannot ensure mutual exclusion.

We propose solutions in this work for the above issues. Basically, a system designer models a safety-critical system using a set of Safecharts. After accepting the Safecharts, we transform them into ETA, while taking care of the safety characterizations in Safecharts, and then automatically generate properties corresponding to the safety constraints. The SGM model checker is enhanced with transition priority and urgency types. Resource access mechanisms in Safecharts are also checked for satisfaction of modeling restrictions that prevent violation of mutual exclusion. Finally, we input the translated ETA to SGM to verify that the safety-critical system satisfies functional and safety properties. Our contributions in this work, as detailed in Section 2, mainly consist of the integration of formal verification techniques with design models, explicit modeling of failures and repairs for verification, and the extension of a conventional model checker for verifying safety-critical systems.

The remaining portion of this paper is organized as follows: Section 2 describes the current state of the art in the verification of safety-critical systems, especially how formal verification has been integrated into conventional techniques. Basic definitions used in our work are given in Section 3, along with an illustrative railway signaling system example. Section 4 will formulate each of our solutions to solving the issues encountered in formally verifying safety-critical systems modeled by Safecharts. Implementation details of the proposed method in the SGM model checker are given in Section 5. Application examples are given in Section 6. The paper concludes and future research directions are given in Section 7.

2 Related Work

Traditionally, safety-critical systems have been verified mainly using *hazard analysis* techniques such as checklists, hazard indices, fault tree analysis (FTA), management oversight and risk tree analysis (MORT), event tree analysis, cause-consequence analysis (CCA), hazards and operability analysis (HAZOP), interface analysis, failure modes and effects analysis (FMEA), failure modes, effects, and criticality analysis (FMECA), and fault hazard analysis (FHA) [19]. Hazard analysis is a continual and iterative process which generally includes the following steps: definitions of objectives and scope, system description, hazard identification, data collection, qualitative ranking of hazards, identification of causal factors, identification and evaluation of corrective measures, verification of control implementations, quantification of unresolved hazards and residual risks, and feedback and evaluation of operational experience. Hazard analysis techniques have been successfully applied to several different real-world safety-critical systems. Nevertheless, a major limitation of hazard analysis is that the phenomena unknown to the analysts are not covered in the analysis and, thus, hazards related to the phenomena are not foreseen. This becomes a severe limitation when the system is complex and analysts may overlook some possible hazards. Safety-critical systems are getting more and more complex and, thus, there is a trend to use methods [4], [14] that are more automatic and exhaustive than hazard analysis, for example, model checking.

The verification of safety-critical systems using formal techniques is not something new [19], as can be seen from methods such as state machine hazard analysis, which was based on Petri nets [20], and the application of model checking to safety-critical system verification based on various formal models such as finite state machines [4], Statecharts [3], Process Control Event Diagrams [28], Scade [8], and Altarica [3]. A common method for the application of model checking to safety-critical system verification is through the specification of safety-related properties using some temporal logic such as *Computation Tree Logic* (CTL) or *Linear Temporal Logic* (LTL) and then checking for the satisfaction of the safety specification [15]. However, as noted by Leveson [19], this approach is inadequate because, in the system models, we are assuming that all of the

components do not fail and the system is proven to be safe under this assumption. However, the assumption is not valid, so transforming each hazard into a formal property for verification, as in [15], is not sufficient. Some works have also integrated traditional FTA techniques with model checking, such as in the *Enhanced Safety Assessment for Complex Systems* (ESACS) project [3], [9], which expressed the *Minimal Cut Sets* (MCS), that is, the minimal combinations of component failures, generated by a model checker, using fault trees. Nevertheless, failure modes of components must still be injected by a safety engineer into the system model before model checking can be performed. Bieber et al. [2] used model checking as a means to check if all unexpected events have been eliminated by conventional FTA techniques. Yang et al. [28] defined various fault modes for each component and used model checking in each fault mode to check for safety properties. In all of the above models and methods, safety-related actions such as failure mode capturing, safety requirements capturing, and model analysis must all be performed separately from the model-checking process. Here, the work on using Safecharts [7], [22], [23], [24] to verify safety-critical systems contributes to the state of the art in verification of such systems in several ways, as described in the following:

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, blend naturally with the semantics of other UML diagrams for the design of safety-critical systems. The work described in this paper automatically transforms Safecharts into the timed automata model, which can be accepted by conventional model checkers. Thus, Safecharts are suitable for both design and verification, thus acting as a bridge between the two, the link between which was seen as an "over-the-wall process" for a long time [10].
2. Safecharts allow and require the *explicit* modeling of component failures and repairs within the safety layer of its models. This is unique and is very helpful not only for safety design engineers but also for safety verification engineers. Further, using Safecharts, there is no need to separately define failure modes and effects, thus preventing accidental omissions and inconsistent specifications.
3. The work here shows how a *conventional* model checker can be extended to perform safety-critical system verification, without the need to integrate conventional methods. Safecharts play a major role in the feasibility of such a simple extension through its unique features of risk states, transition priorities, and component failure and repair modeling artifacts.
4. Due to Safecharts being a variant of the UML Statecharts, automatic code generation is supported through model-driven development, which is becoming a standard way of software code design. Safety requirements proved in the models can thus be preserved in the final software code through this automatic code generation process. This is out of scope here in this work but is an added advantage which must be carefully proved.

3 System Model, Specification, and Model Checking

Before going into the details of how Safecharts are used to model and verify safety-critical systems, some basic definitions and formalizations are required, as given in this section. Both Safecharts and their translated ETA models will be defined. TCTL and model checking will also be formally described. A railway signaling system is used as a running example for illustration. Since Safecharts are based on the UML Statecharts, we first define Statecharts in Definition 1 and then define Safecharts in Definition 2.

Definition 1 (Statecharts). Statecharts *are a tuple* $F = (S, T, E, \theta, V, \phi)$, *where* S *is a set of all states,* T *is a set of all possible transitions,* E *is a set of all events,* θ *is the set of possible types of states in Statecharts, that is,* $\theta = \{\mathrm{AND}, \mathrm{OR}, \mathrm{BASIC}\}$, V *is a set of integer variables, and* $\phi ::= v \sim c | \phi_1 \wedge \phi_2 | \neg\phi_1$ *in which* $v \in V$, $\sim \in \{<, \leq, =, \geq, >\}$, c *is an integer, and* ϕ_1 *and* ϕ_2 *are predicates. Let* F_i *be an arbitrary state in* S. *It has the general form*

$$F_i = (\theta_i, C_i, d_i, T_i, E_i, l_i),$$

where

- θ_i *is the type of the state* F_i, $\theta_i \in \theta$,
- C_i *is a finite set of direct substates of* F_i, *referred to as* child states *of* F_i, $C_i \subseteq S$,
- d_i *is* $d_i \in C_i$ *and is referred to as the* default state *of* F_i. *It applies only to* OR *states,*
- T_i *is a finite subset of* $F \times F$, *referred to as explicitly* specified transitions *in* F_i,
- E_i *is the finite set of events relevant to the specified transitions in* T_i, $E_i \subseteq E$, *and*
- l_i *is a function* $T_i \rightarrow E \times \phi \times 2^{Ei}$, *labeling each and every specified transition in* T_i *with a triple,* 2^{Ei} *denoting the set of all finite subsets of* E_i.

Given a transition $t \in T$, its label is denoted by $l(t) = (e, fcond, a)$, written conventionally as $e[fcond]/a$. e, $fcond$, and a in the latter, denoted also as $trg(t) = e$, $con(t) = fcond$, and $gen(t) = a$, represent, respectively, the triggering event, the guarding condition, and the set of generated actions.

Safecharts [7], [22], [23], [24] are a variant of Statecharts intended exclusively for safety-critical systems design. With two separate representations for functional and safety requirements, Safecharts bring the distinctions and dependencies between them into sharper focus, helping both designers and auditors alike in modeling and reviewing safety features. Fig. 1 shows the *functional* and *safety* layers of a Safecharts model 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. The notations $\nwarrow$ and $\nearrow$ in the safety layer will be defined in Definition 2 and basically restrict setting a route or enforce the release of a route when any of the signals in that route are or become faulty.

Further, Safecharts incorporates ways to represent equipment failures and failure handling mechanisms and

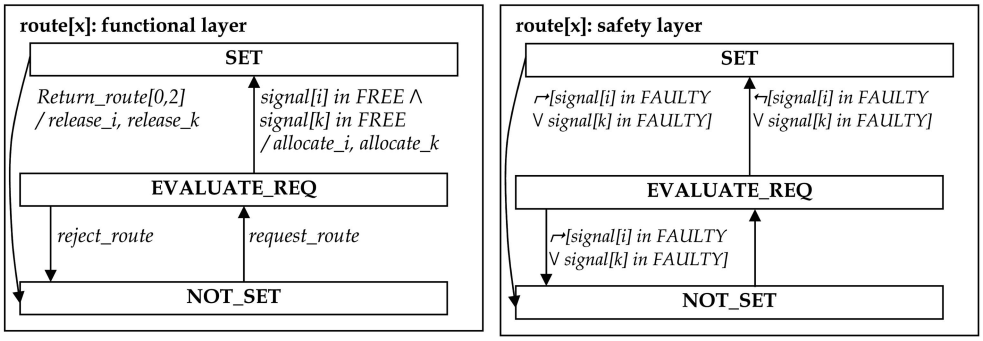

Fig. 1. Safechart for route[x] with functional and safety layers.

use a safety-oriented classification of transitions and a safety-oriented scheme for resolving any unpredictable nondeterministic pattern of behavior. It achieves these through an explicit representation of risks posed by hazardous states by means of an ordering of states and a concept called *risk band* [22]. Recognizing the possibility of gaps and inaccuracies in safety analysis, Safecharts do not permit transitions between states with unknown relative risk levels. However, in order to limit the number of transitions excluded in this manner, Safecharts provide a default interpretation for relative risk levels between states not covered by the risk ordering relation, requiring the designer to clarify the risk levels in the event of a disagreement and thus improving the risk assessment process.

Definition 2 (Safecharts). Safecharts *Z extends Statecharts by adding a safety layer. States are extended with a risk ordering relation and transitions are extended with safety conditions. Given two comparable states s_1 and s_2, a risk-ordering relation $\leq$ specifies their relative risk levels, that is, $s_1 \leq s_2$ specifies that s_1 is safer than s_2. Transition labels in Safecharts have an extended form: $e[fcond]/a[l,u)\ \psi[G]$, where e, $fcond$, and a are the same as in Statecharts. 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* $\nleftarrow$. *Given a transition label of the form* $\nleftarrow[G]$, *it signifies that the transition is forbidden to execute as long as G holds.*
- Mandatory *enforcement value, denoted by* $\nrightarrow$. *Given a transition label of the form* $[l,u)\nrightarrow[G]$, *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.*

A railway signaling system that sets and releases routes is given as an example for modeling using Safecharts. In Fig. 2, we can see that there are two routes, route[x] and route[y], such that route[x] must have signal[i] and signal[k] allocated, whereas route[y] must have signal[j] and signal[k] allocated, where signal[k] is a resource shared between the two routes.

The Safecharts model for route[x] was given in Fig. 1 and that for route[y] will be similar. However, we can also express the two layers through an integrated model, as in Fig. 3. The integrated Safecharts model for signal[i] is given in Fig. 4.

The Safecharts model is used for modeling safety-critical systems; however, the SGM model checker can understand only a flattened model called *ETA* [13], which was enhanced with priority and urgency, as defined in Definition 4.

Definition 3 (Mode Predicate). *Given a set C of clock variables and a set D of discrete variables, the syntax of a* mode predicate *η over C and D is defined as $\eta := false|x \sim c|x-y \sim c|d \sim c|\eta_1 \wedge \eta_2|\neg\eta_1$, where x,*

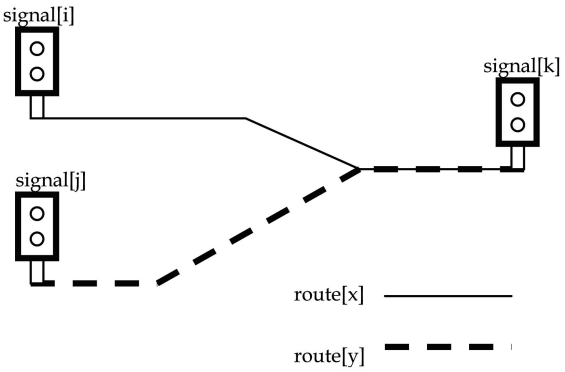

Fig. 2. Routes and signals.

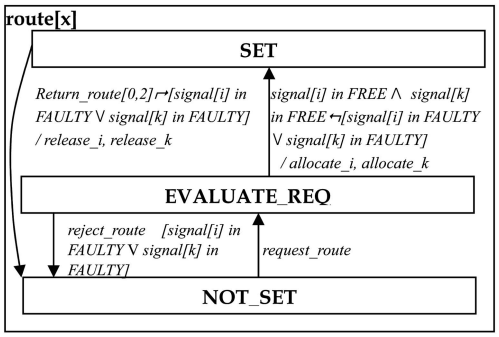

Fig. 3. Integrated Safechart for route[x].

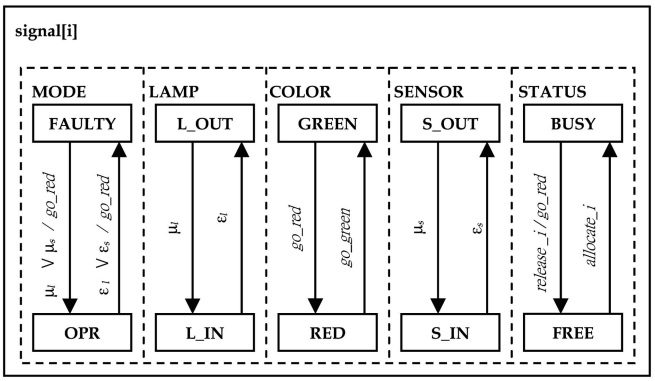

Fig. 4. Integrated Safechart for signal[i].

$y \in C$, $\sim \in \{\leq, <, =, \geq, >\}$, $c \in N$, $d \in D$, *and* η_1 *and* η_2 *are mode predicates.*

Let $B(C, D)$ be the set of all mode predicates over C and D.

Definition 4 (Extended Time Automaton). *An* Extended Time Automaton *(ETA) is a tuple* $A_i = (M_i, m_{i0}, C_i, D_i, L_i, \chi_i, T_i, \lambda_i, \pi_i, \mu_i, \tau_i, \rho_i)$ *such that* M_i *is a finite set of modes,* $m_{i0} \in M$ *is the initial mode,* C_i *is a set of clock variables,* D_i *is a set of discrete variables,* L_i *is a set of synchronization labels, and* $\varepsilon \in L_i$ *is a special label that represents asynchronous behavior (that is, no need for synchronization),* $\chi_i : M_i \rightarrow B(C_i, D_i)$ *is an* invariance *function that labels each mode with a condition true in that mode,* $T_i \subseteq M_i \times M_i$ *is a set of transitions,* $\lambda_i : T_i \rightarrow L_i$ *associates a synchronization label with a transition,* $\pi_i : T_i \rightarrow \mathbf{N}$ *associates an integer priority with a transition,* $\mu_i : T_i \rightarrow \{\text{lazy}, \text{eager}, \text{delayable}\}$ *associates an urgency type with a transition,* $\tau_i : T_i \rightarrow B(C_i, D_i)$ *defines the transition triggering conditions, and* $\rho_i : T_i \rightarrow 2^{Ci \cup (Di \times N)}$ *is an* assignment *function that maps each transition to a set of assignments such as resetting some clock variables and setting some discrete variables to specific integer values.*

Take as an example the ETA that is semantically equivalent to the MODE Safecharts in signal[i] of Fig. 4, as illustrated in Fig. 5.

A system state space is represented by a *system state graph* as defined in Definition 5.

Definition 5 (System State Graph). *Given a system* S *with* n *components modeled by*

$$A_i = (M_i, m_{i0}, C_i, D_i, L_i, \chi_i, T_i, \lambda_i, \pi_i, \mu_i, \tau_i, \rho_i), 1 \leq i \leq n,$$

the system model is defined as a state graph represented by

$$A_1 \times \ldots \times A_n = A_S = (M, m^0, C, D, L, \chi, T, \lambda, \pi, \mu, \tau, \rho),$$

where

- $M = M_1 \times M_2 \times \ldots \times M_n$ *is a finite set of system modes,* $m = m_1.m_2 \ldots..m_n \in M$,
- $m^0 = m_1^0.m_2^0 \ldots..m_n^0 \in M$ *is the initial system mode,*
- $C = \cup_i C_i$ *is the union of all sets of clock variables in the system,*
- $D = \cup_i D_i$ *is the union of all sets of discrete variables in the system,*

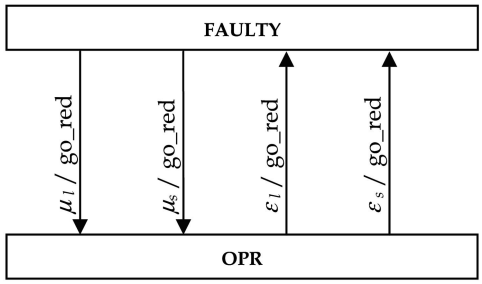

Fig. 5. ETA corresponding to MODE Safecharts in signal[i].

- $L = \cup_i L_i$ *is the union of all sets of synchronization labels in the system,*
- $\chi : M \rightarrow B(\cup_i C_i, \cup_i D_i)$, $\chi(m) = \wedge_i \chi_i(m_i)$, *where* $m = m_1.m_2 \ldots..m_n \in M$,
- $T \subseteq M \times M$ *is a set of system transitions which consists of two types of transitions:*
  - Asynchronous transitions: $\exists\, i, 1 \leq i \leq n$, $e_i \in T_i$ *such that* $e_i = e \in T$.
  - Synchronized transitions: $\exists\, i, j, 1 \leq i \neq j \leq n$, $e_i \in T_i$, $e_j \in T_j$ *such that* $\lambda_i(e_i) = (l, \text{in})$, $\lambda_j(e_j) = (l, \text{out})$, $l \in L_i \cap L_j \neq \emptyset$, *and* $e \in T$ *is synchronization of* e_i *and* e_j *with conjuncted triggering conditions and union of all transitions assignments (defined later in this definition).*
- $\lambda : T \rightarrow L$ *associates a synchronization label with a transition, which represents a blocking signal that was synchronized, except for* $\varepsilon \in L$, ε *is a special label that represents asynchronous behavior (that is, no need for synchronization),*
- $\pi : T \rightarrow \text{N}$ *associates an asynchronous transition with its original priority and a synchronous transition with the maximum priority among all the synchronized transitions,*
- $\mu : T \rightarrow \{\text{lazy}, \text{eager}, \text{delayable}\}$ *associates an asynchronous transition with its original urgency type and a synchronous transition with the eager type if there is one synchronized or with the delayable type if there is one, otherwise, with the lazy type,*
- $\tau : T \rightarrow B(\cup_i C_i, \cup_i D_i)$, $\tau(e) = \tau_i(e_i)$ *for an asynchronous transition, and* $\tau(e) = \tau_i(e_i) \wedge \tau_j(e_j)$ *for a synchronous transition, and*
- $\rho : T \rightarrow 2^{\cup_i C_i \cup (\cup_i D_i \times N)}$, $\rho(e) = \rho_i(e_i)$ *for an asynchronous transition, and* $\rho(e) = \rho_i(e_i) \cup \rho_j(e_j)$ *for a synchronous transition.*

Definition 6 (Safety-Critical System). *A* safety-critical system *is defined as a set of* resource *components and* consumer *components. Each component is modeled by one or more Safecharts. If a safety-critical system* H *has a set of resource components* $\{R_1, R_2, \ldots, R_m\}$ *and a set of consumer components* $\{C_1, C_2, \ldots, C_n\}$, H *is modeled by* $\{Z_{R1}, Z_{R2}, \ldots, Z_{Rm}, Z_{C1}, Z_{C2}, \ldots, Z_{Cn}\}$, *where* Z_X *is a Safechart model for component* X. *Safecharts* Z_{Ri} *and* Z_{Cj} *are transformed into corresponding ETA* A_{Ri} *and* A_{Cj}, *respectively. Therefore,* H *is semantically modeled by the state graph* $A_{R1} \times \ldots \times A_{Rm} \times A_{C1} \times \ldots \times A_{Cn}$, *as defined in Definition 5.*

For our railway signal system,

$$\{\text{signal[i]}, \text{signal[j]}, \text{signal[k]}\}$$

is the set of resource components and {route[x], route[y]} is the set of consumer components. The system is thus modeled by the state graph

$$A_{\text{signal[i]}} \times A_{\text{signal[j]}} \times A_{\text{signal[k]}} \times A_{\text{route[x]}} \times A_{\text{route[y]}}.$$

Timed Computation Tree Logic (TCTL) [11] is a *timed* extension of the well-known temporal logic called CTL, which was proposed by Clarke and Emerson in 1981. We will use TCTL to specify the safety properties that are required to be satisfied by a safety-critical system modeled by Safecharts. The SGM model checker also chooses TCTL as its logical formalism, which is defined as follows:

Definition 7 (TCTL). *A TCTL formula has the following syntax:*

$$\phi ::= \eta | \text{EG}\phi' | \text{E}\phi' \text{U}_{\sim c} \phi'' | \neg\phi' | \phi' \vee \phi'',$$

where η is a mode predicate, ϕ' and ϕ'' are TCTL formulas, $\sim \in \{<, \leq, =, \geq, >\}$, and $c \in N$. $\text{EG}\phi'$ means there is a computation from the current state along which ϕ' is always true. $\text{E}\phi' \text{U}_{\sim c} \phi''$ means there exists a computation from the current state, along which ϕ' is true until ϕ'' becomes true, within the time constraint of $\sim$ c. Shorthand, like EF, AF, AG, AU, $\wedge$, and $\rightarrow$, can all be defined [11].

For the railway signal system, we can specify that the following safety properties must be satisfied:

1. $\text{AG}(\neg(\text{route[x] in SET} \wedge \text{route[y] in SET}))$,
2. AG(signal[i] in FAULTY $\rightarrow \neg$(route[x] in SET)),
3. AG(signal[j] in FAULTY $\rightarrow \neg$(route[y] in SET)), and
4. AG(signal[k] in FAULTY $\rightarrow \neg$(route[x] in SET $\vee$ route[y] in SET)).

The first property states that both routes should not be set at the same time since they share a common resource, signal[k]. The other three properties state that a route should not be set if any of its resources (signals) is faulty.

Definition 8 (Model Checking [5], [6], [25]). *Given a Safechart Z that represents a safety-critical system and a TCTL formula ϕ expressing some desired specification, model checking verifies if Z satisfies ϕ, denoted by $Z \models \phi$.*

Model checking can be either explicit, using a labeling algorithm, or symbolic, using a fixed-point algorithm. *Binary Decision Diagram* (BDD) and *Difference Bound Matrices* (DBM) are data structures used for Boolean formulas and clock zones [6], respectively. For our railway signal system, we must check if $A_{\text{signal[i]}} \times A_{\text{signal[j]}} \times A_{\text{signal[k]}} \times A_{\text{route[x]}} \times A_{\text{route[y]}} \models \phi$, where ϕ is each of the four properties described above.

4 Model-Checking Safecharts

Safecharts have been used to model safety-critical systems, but the models have never been used for verification. In this work, we propose a method to verify safety-critical systems modeled by Safecharts. Our tool is the *State Graph Manipulators* (SGM) [13], [27], which is a high-level model checker for systems modeled by a set of ETA. Several issues to be resolved in model-checking Safecharts were described in Section 1, for which we propose solutions in the rest of this section. First, we show how Safecharts are transformed into ETA models that can be accepted by SGM while maintaining the safety semantics. Second, we show how some properties are generated for model checking. Third, we discuss how risk relationships are represented by transition priorities and urgencies in SGM. Finally, we show how mutually exclusive access of shared resources cannot be guaranteed by the original Safecharts and how we solved the problem.

4.1 Flattening Safecharts and Safety Semantics

Our primary goal is to model check Safecharts, a variant of Statecharts. However, Safecharts cannot be accepted as system model inputs by most model checkers, which can accept only flat automata models such as the ETA accepted by SGM. As a result, the state hierarchy and concurrency in Safecharts must be transformed into semantically equivalent constructs in ETA. Further, besides the functional layer, Safecharts have an extra safety layer, which must be transformed into equivalent modeling constructs in ETA and specified as properties for verification.

There are three categories of states in Safecharts: OR, AND, and BASIC. An OR state or an AND state generally consists of two or more substates. Being in an AND state means being in all of its substates simultaneously, whereas being in an OR state means being in exactly one of its substates. A BASIC state is translated into an ETA *mode*. The translations for OR states and AND states are performed as described in [18].

4.1.1 Safety Semantics

The syntax for the triggering condition and action of a transition in Safecharts is $e\,[fcond]/a[l,u)\psi[G]$, where e is the set of triggering events, $fcond$ is the set of guard conditions, a is the set of broadcast events, $[l,u)$ is the time interval specifying the time constraint, ψ means the execution conditions for safety constraints, and G is the set of safety layer's guards. In Safecharts, $e[fcond]/a$ appears in the *functional* layer, whereas $[l,u)\psi[G]$ may appear in the *safety* layer. The two layers of Safecharts can be integrated into one in ETA as described in the following. However, we need to design three different types of transitions [1]:

- *Eager Evaluation* (ε). Execute the action as soon as possible, that is, as soon as a guard is enabled. Time cannot progress when a guard is enabled.
- *Delayable Evaluation* (δ). You can put off execution until the last moment the guard is true. Therefore, time cannot progress beyond a *falling edge* of guard.
- *Lazy Evaluation* (λ). You may or may not perform the action.

The transition condition and assignment

$$e[fcond]/a[l,u)\psi[G]$$

can be classified into three types as follows:

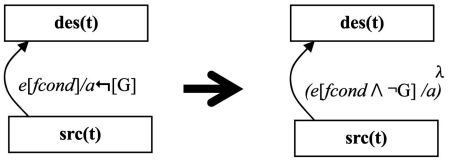

Fig. 6. Transformation of prohibition evaluation.

1. e[fcond]/a. There is no safety clause on a transition in Safechart, thus, we can simply transform it to the one in ETA. We give the translated transition a *lazy* evaluation λ.
2. e[fcond]/a ↰ [G]. There is a *prohibition* enforcement value on a transition t. It signifies that the transition t is forbidden to execute as long as G holds. During translation, we combine them as $e[fcond \wedge \neg G]/a$. We give the translated transition a *lazy* evaluation λ. The transformation is shown in Fig. 6.
3. e[fcond]/a[l, u) ↱ [G]. There is a *mandatory* enforcement value on a transition t. Given a transition label of the form e[*fcond*]/a[l, u) ↱ [G], it signifies that the transition is forced to execute within $[l, u)$ whenever G holds. We translate functional and safety layers into a transition t_1 and a path t_2, respectively. t_1 represents $e[fcond]/a$, which means t_1 is enabled if the triggering event e occurs and its functional conditional $fcond$ is true. We give t_1 a *lazy* evaluation λ. Path t_2 is combined by two transitions t_ε and t_δ. Transition t_ε is labeled $[G]/timer := 0$, where $timer$ is a clock variable used for the time constraint and we give t_ε an *eager* evaluation ε. When G holds, t_ε executes as soon as possible and t_ε's destination is a newly added mode named $translator(t)$. t_δ's source is $translator(t)$ and its destination is t's destination. t_δ's guard is $[timer \geq l \wedge timer < u]$. However, we give t_δ a *delayable* evaluation (δ), which means it can put off execution until the last moment the guard is true. The procedure of translation is shown in Fig. 7.

4.2 Property Specification for Safecharts

In the safety layer of Safecharts, there are two types of safety conditions on a transition: One is *prohibition* and the other is *mandatory*. After parsing the Safechart models of a safety-critical system, corresponding properties are automatically generated without requiring the user to specify again. Such properties are used to verify if the safety layers work or not. As described in the following, to ensure that

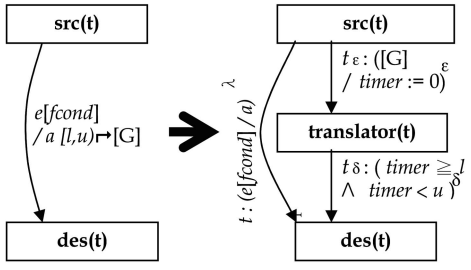

Fig. 7. Transformation of mandatory evaluation.

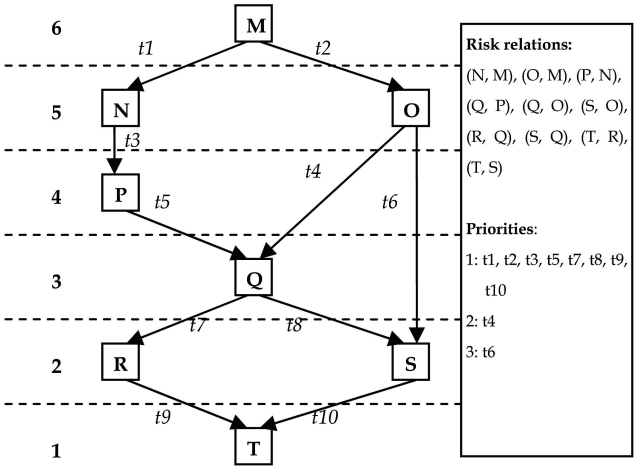

Fig. 8. Risk graph with risk band.

the safety constraints are working, two categories of properties are generated automatically for model checking:

1. For prohibition condition ↰ [G]:

$$\mathrm{AG}((src(t) \wedge G) \rightarrow \neg \mathrm{EX}(des(t))).$$

2. For mandatory condition [l, u) ↱ [G]:

$$\mathrm{AG}((src(t) \wedge G \rightarrow \neg \mathrm{EX}(\neg translator(t)))$$

and $\mathrm{AG}(translator(t) \wedge timer < u)$.

Proving the first property ensures that a safety-critical system will not become dangerous under any prohibited condition. Proving the second set of properties ensures that a safety-critical system will go to a safer state whenever there is some fault.

4.3 Transition Priority

Nondeterminism is often used as a means of modeling data-dependent or runtime choices and occurs whenever system behavior is abstracted in the system model. However, for safety-critical systems, nondeterminism is undesirable and must be avoided or eliminated because it might result in dangerous behaviors presiding over other safer behaviors. This is accomplished in Safecharts by removing nondeterminisms in all cases except when there is no safety implication. In Safecharts, a user specifies the relative safety levels between two states using *risk relation* tuples, which are used to establish a *risk graph* [23] for the Safechart. For example, the tuple (N, M) specifies that state N is of a lower risk compared to state M, as shown in Fig. 8. Noncomparable conditions may still exist in a risk graph.

Risk bands [22], [23] were used in Safecharts to determine the relative risk relations that were not explicitly described. As shown in Fig. 8, since each state belongs to exactly one risk band, which are numbered starting from 1, we decided to associate each Safechart state with an integer risk level. Safe nondeterminism dictates that, whenever there is nondeterminism between two or more transitions, a transition leading to a safer state is given higher priority. Hence, we implemented *transition priorities* based on the risk bands of a transition's source and destination modes as

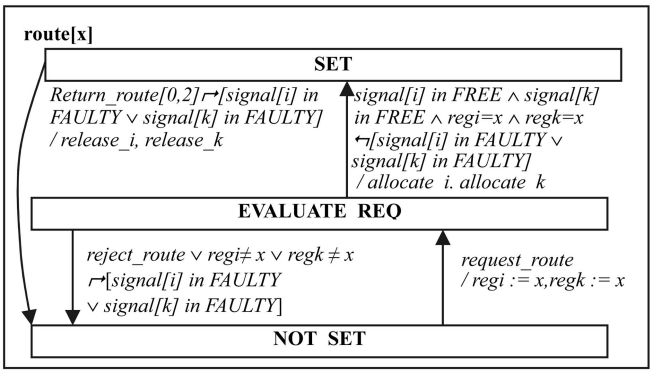

Fig. 9. Safechart for route[x] with mutual exclusion.

$pri(t) = rb_{src(t)} - rb_{des(t)}$, where $pri(t)$ is the priority assigned to transition t, $rb_{src(t)}$ is the risk band of transition t's source mode, and $rb_{des(t)}$ is the risk band of transition t's destination mode. The larger the value of $pri(t)$ is, the higher the priority of transition t is. In Fig. 8, $pri(t4)$ is 2 and $pri(t6)$ is 3. Obviously, when $t4$ and $t6$ are both enabled, $t6$ will be executed in preference to $t4$. With risk bands, we can give a transition leading to a lower risk-band state a higher priority. For two or more transitions leading to modes in the same risk band, one has to look for the risk bands of the two transitions' *future target states* until either the states are in different risk bands or all of the future states are in the same band. Further, urgent transitions are also given higher priority than lazy transitions. A novel clock zone subtraction operation is required here [12].

4.4 Resource Access Mechanisms

Safecharts model both consumers and resources. However, when resources must be used in a mutually exclusive manner, a model designer may easily violate the mutual exclusion restriction by simultaneous checking and discovery of free resources, followed by their concurrent usages. A motivational example can be observed in the railway signaling system, as illustrated in Figs. 2, 3, and 4, where signal[k] must be shared in a mutually exclusive way between route[x] and route[y]. However, each route checks if signal[k] is free and, if it finds it free, then both routes will be SET, assuming G does not hold. This is clearly a modeling trap that violates mutually exclusive resource usages. A serious tragedy could happen as two intersecting routes are set, perhaps resulting in a future train collision.

From the above, we know that, when consumers try to acquire resources that cannot be used concurrently, it is not safe to check only the status of resources. We need some kind of model-based mutual exclusion mechanism. A very simple policy would be like Fischer's mutual exclusion protocol [17]. For each mutually exclusive resource, a variable is used to record the ID of the consumer currently using the resource. Before the consumer uses the resource, it has to check if the variable is set to its ID. Fig. 9 is a corrected variant of the route Safechart from Fig. 3. When route[x] transits into EVALUATE_REQ, it sets variable *reg* to its ID. When route[x] tries to transit into the SET mode to acquire the usage of *resource*, it needs to check if *reg* is still its ID, x. If *reg* is still x, then route[x] acquires the usage of the resource. Other mechanisms such as atomic test-and-set performed on a single asynchronous transition can also achieve mutual exclusion.

5 Implementation in a Model Checker

The model checking of Safecharts has been implemented in the SGM [27], which is a TCTL model checker for real-time and embedded systems. SGM was a conventional model checker, which could take only a set of ETA as the system model. We enhanced SGM in several ways to accommodate the verification of Safecharts. First and foremost, we had to develop an input language that could allow recursive definitions of the hierarchical Safecharts models. Second, we had to employ a flattening algorithm [18] to remove all of the hierarchies and obtain a set of ETA that was semantically equivalent to the originally specified set of Safecharts. Third, the preservation of safety semantics from Safecharts to ETA demanded support for transition urgencies such as eager and delayable, besides the original lazy semantics. We implemented support for transition urgencies in SGM through a novel zone-capping operation. Finally, we had to develop and implement algorithms for the construction of risk bands from the risk relations specified in Safecharts and then use the risk levels as the risk associated with the ETA states. We also had to check if there are any transitions between states with unknown risk levels. For implementing safe nondeterminism, we had to support prioritized transitions in SGM, which we accomplished using a clock zone subtraction operation [12], [21]. We also prove the semantic equivalence between Safecharts and the generated set of ETA. State-space explosion is handled through several state-space reduction techniques, as discussed in Section 5.2.

5.1 Semantic Equivalence

A given set of Safecharts modeling a safety-critical system is transformed into a set of ETA for model checking by an enhanced version of the SGM tool. We show the semantic equivalence between the user-specified set of Safecharts model and the automatically generated set of ETA models in the following theorem:

Theorem 1. *The automatically generated set of ETA models is semantically equivalent to the given set of Safecharts as far as model checking is concerned.*

Proof. We prove semantic equivalence between the two models based on their differences, which include concurrency, hierarchy, safety transitions, and safety nondeterminism:

1. *Concurrency*. Since each AND state in a Safechart is transformed into an equivalent set of concurrent ETA, concurrency is preserved. However, transitions entering or exiting an AND state must be synchronized so that all initial or history states in an AND state are entered simultaneously or all active states left simultaneously. In the automatic generation process, a new initial mode was introduced into each component ETA and incoming and outgoing transitions could thus be synchronized.

2. *Hierarchy*. An OR state represents a lower hierarchy level in Safecharts. In our transformation process, we flatten out this hierarchy by embedding the lower hierarchy level of Safecharts into the higher hierarchy level. However, in doing so, we must preserve the incoming and outgoing transitions of an OR state. Thus, the hierarchy semantics was preserved through flattening.
3. *Safety Transitions*. Safecharts have an extra safety layer besides the conventional functional layer as shown in Fig. 1, with two possible types of safety transitions: prohibitory and mandatory. Safety semantics preservation is discussed for each, as given below:
   a. Since the prohibitory transition only specifies a condition G under which the functional transition should not take place, our transformation through conjunction of $\neg G$ with the triggering condition of all corresponding functional transitions, as shown in Fig. 6, naturally prohibits the functional transition to take place when G holds, thus preserving the prohibitory transition semantics.
   b. Unlike prohibitory transitions, a mandatory transition specifies a functionally independent requirement of transiting to a safer successor state when a condition G is satisfied and the transition must take place within some time interval $[l, u)$. Our transformation, as shown in Fig. 7, generates two independent ETA paths leading to the same safer successor state, that is, one is the nominal functional transition and the other is a path with two transitions: an eager transition and a delayable transition. The urgency semantics associated with the eager and delayable transitions in the ETA models preserve exactly the mandatory semantics of safety transitions in Safecharts because not only the time requirement is preserved, but also the eager transition is given a higher priority than other lazy transitions (the functional ones), thus the eager transition will be taken whenever enabled, irrespective of other functional ones, which is exactly what the mandatory safety transition in Safecharts requires.
4. *Safety nondeterminism*. Nondeterminism is only allowed in Safecharts when transitions lead to states with equal levels of safety; otherwise, transitions leading to safer states are given higher priority. In our automatic transformation, we have preserved these semantics through the derivation of a risk level for each state based on the user-specified risk relations that can be used to construct a risk graph and then converted into risk bands. The risk level of each state is then used to assign an integer priority for each nondeterministic transition. Prioritization of transitions was then supported in the SGM model checker through the zone subtraction operation [12]. Transitions with higher priority levels led to safer states. Thus, safety nondeterminism semantics was preserved.

In summary, we can see that the semantics of the above four features of Safecharts were all preserved in the ETA models. All other semantics of Safecharts, such as state, transition, and computation run, are similar to that in ETA. Thus, we can conclude that the automatically generated ETA models are semantically equivalent to the given set of Safecharts. □

5.2 Scalability and State-Space Reduction

In model checking, state-space explosion is an infamous problem that must be dealt with for any real-world system of moderate size. We investigated different techniques for reducing the state-space sizes of the ETA models that were generated automatically from Safecharts for safety-critical systems. The techniques included variable usage and clock hiding, clock number reduction, safety transition reduction, and safe time abstractions. Due to page limits, we informally describe each of them here. The experimental results are shown in Section 6.4.

Variable usage and clock hiding are general techniques based on the analysis of discrete and clock variables, respectively. In variable usage reduction, we record all possible variable values in each mode and automatically prune the transitions that cannot be taken from that mode. This might seem to be trivial; however, since ETA are merged two at a time in a compositional way in the SGM, this reduction needs to check all other ETA that are not yet merged into the global state graph. In clock hiding, we hide the clocks whose values are not read before being reset. Both of these techniques have been formally described in [27].

Clock number reduction tries to minimize the number of clocks by reusing them whenever possible. In Safecharts, clocks are mainly used to timeout mandatory transitions. Hence, each Safechart with a mandatory transition must have a clock. However, we can reduce the number of clocks by reusing the same clock for all mandatory transitions that have the same deadline. This is a stricter semantics than the original Safecharts semantics. However, if this model is proven to be safe, then the original model would also be safe and this technique results in significant reductions.

Safety transition reduction tries to replace mandatory transitions that are not related to safety with normal transitions. In Safecharts, it may happen that a mandatory transition is imposed when it is actually not directly related to safety. Such mandatory transitions need not have a deadline and, thus, we can transform them into normal lazy transitions. State-space size is also reduced significantly using this reduction technique.

Safe time abstraction is a technique whereby the time interval $[l, u)$ on a mandatory transition is replaced with [0, 0]. This results in stricter semantics, which means that, if proven safe, then the original model is also safe. However, state-space reduction can be achieved in such a stricter semantics model.

TABLE 1
Results of the Railway Signaling System

Components		Each Safechart		Each ETA		
Name	#	$\|S\|$	$\|T\|$	#	$\|M\|$	$\|T\|$
route[x, y]	2	4	7	1	5	13
signal[i,j,k]	3	16	16	5	10	16
full system		56	62	17	40	74

6 Application Examples

The proposed model-checking verification methodology was applied to several safety-critical systems, including several variants of the basic railway signaling system, the hydraulic system in the Airbus A320 airplane, and a nuclear reactor model of the TMI accident. Safecharts were automatically transformed into ETA, input to SGM, merged into a global state graph along with state-space reduction techniques, and then model checked with corresponding safety properties. Due to page limits, the ETA models are given in online supplementary materials, which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm.

6.1 Railway Signaling System

The basic railway signaling system consists of two routes: route[x] and route[y], where route[x] requires signal[i] and signal[k] and route[y] requires signal[j] and signal[k]. The numbers and sizes of the Safecharts and the generated ETA are given in Table 1. As illustrated in the supplementary materials, which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm, for each route Safechart, one ETA is obtained and, for each signal Safechart, five ETA are generated. Thus, in the full system consisting of five Safecharts, 17 ETA are generated. It can be observed that the number of ETA modes, 40, is less than the number of Safecharts states, 56. The reason for this reduction is that hierarchical states do not exist in ETA. The mutual exclusion issue was resolved as described in Sections 4.4, respectively.

As illustrated in the figures given in the supplementary materials, which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm, a number of variants of the basic railway signaling system were used for validating the proposed method. Varying the number of routes and the number of signals in each route increases the complexity and the concurrency of the system. However, we can observe from the verification results in Table 2 that the amount of time and memory expended for model transformations are very small. However, the global state graph increases exponentially in size and, thus, the state-space reduction techniques proposed in Section 5.2 were all applied. The results will be given in Section 6.4. The number of properties to be verified also increases and, thus, their automatic generation is also a crucial step for successful and easily accessible verification of safety-critical systems. The safety properties that were verified for these railway signaling systems are as given in Section 3 after TCTL was defined.

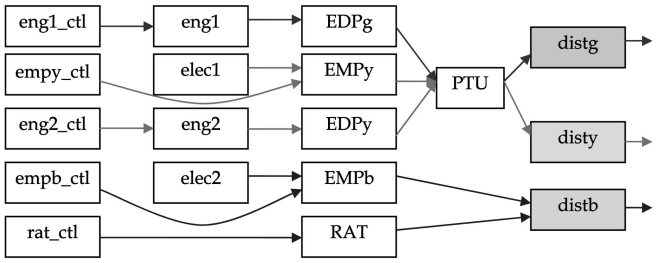

Fig. 10. A320 hydraulic system model.

6.2 Airbus A320 Hydraulic System

The hydraulic system in an advanced airplane such as an Airbus A320 supplies hydraulic power for aircraft control during flight and on the ground [2]. The safety requirements are listed as follows: 1) We need to ensure that we are not in a state of total loss of hydraulic power, which is classified as catastrophic, and 2) we need to verify that a single failure does not result in total loss of power. As shown in Fig. 10, there are three kinds of pumps: Electric Motor Pump (EMP) powered by the electric system, Engine-Driven Pump (EDP), powered by the two aircraft engines, and one RAT Pump, powered by Ram Air Turbine. There are three power distribution channels: Green, Blue, and Yellow. The Blue channel is made up of one electric pump, EMPb, one RAT pump, and a distribution line, distb. The Green channel is made up of one pump driven by engine 1, EDPg, and a distribution line, distg. The Yellow channel is made up of one pump driven by engine 2, EDPy, one electric pump, EMPy, and a distribution line, disty. A power transfer unit (PTU) opens a transmission from the green hydraulic power to the yellow distribution line and vice versa as soon as their differential pressure is higher than a given threshold.

This system was originally modeled by the A320 development teams in Altarica [2], [16]; however, as shown in Section 6.5, our method has several advantages compared to

TABLE 2
Results of Application Examples

	System		Safecharts			ETA			$\|\phi\|$	ME	Time	Mem
	$\|R\|$	$\|SIG\|$	#	$\|S\|$	$\|T\|$	#	$\|M\|$	$\|T\|$			(μs)	(MB)
A	2	3(1)	5	56	62	17	40	74	20	1	230	0.12
B	2	4(1)	6	72	81	22	50	94	25	1	292	0.12
C	2	4(2)	6	72	84	22	50	98	30	2	337	0.13
D	3	4(1)	7	76	85	23	55	103	30	1	326	0.14
E	3	5(2)	8	92	110	28	65	131	45	2	515	0.14
F	4	5(1)	9	96	108	29	70	132	40	1	634	0.14

$|R|$: total number of routes, $|SIG|$: total number of signals (number of shared signals), and $|\phi|$: number of properties generated.

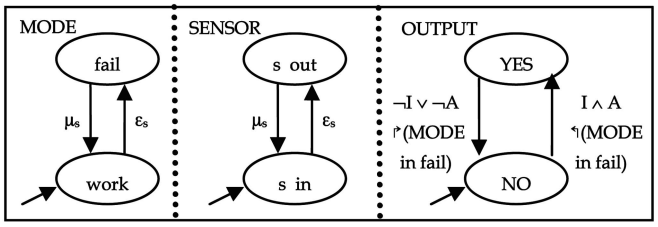

Fig. 11. Integrated Safecharts model for a basic block in A320.

Altarica. There are two types of models in this system, namely, basic block and controller block. Each basic block was modeled by a Safechart with a MODE OR state representing if it is working or faulty, a SENSOR OR state modeling its sensor, and an OUTPUT OR state showing if the basic block is providing hydraulic power. Each controller block, besides the MODE and SENSOR states, also has a STATUS OR state, representing whether the system is on the ground or in flight, and an OUTPUT state, representing if it is currently activated or not. As we can see, the failure modes for each component block are explicitly modeled into Safecharts, thus eliminating the need to separately define the failure modes and effects. Figs. 11 and 12 show the generic models for a basic and a controller block, respectively.

Each of the five pumps and their corresponding controllers, along with the three distribution lines, were all modeled by Safecharts. However, since the blue system is independent of the other two systems, we verified the blue system separately. To check for total loss of hydraulic power, we had to check if the blue system does not supply power (that is, EF (EG $(\text{powerB} = 0)$)) and if the rest of the system (green and yellow systems) also does not supply power (that is, EF (EG($\text{powerG} = 0$ and $\text{powerY} = 0$))). It was easy to check the second safety requirement, that is, if a single failure results in total loss of hydraulic power, which, expressed in TCTL, is EF(EG(MODE in $\text{fail} => \text{powerB} = 0$ and $\text{powerG} = 0$ and $\text{powerY} = 0$)).

The A320 hydraulic system was modeled by 13 Safecharts with 145 states and 105 transitions, which were automatically transformed into 44 ETA models with 98 modes and 125 transitions. As shown in Section 6.4, several state-space reduction techniques were implemented into the SGM model checker to successfully verify the A320 hydraulic system.

6.3 Nuclear Reactor System

The Three Mile Island 2 nuclear reactor was also modeled [22] to see if we could verify the failure that occurred during the accident on 28 March 1979. As shown in Fig. 13, there are two circuits: a primary circuit and a secondary circuit. The primary circuit, as modeled in Fig. 14, generated steam due to the reactor heat and was confined in a containment building to avoid the leakage of radiation. A pressurizer valve and some control rods, as modeled in Fig. 15, controlled the nuclear reaction. The secondary circuit fed water to the primary circuit for cooling and heat exchange and used steam from the primary circuit to drive a turbine.

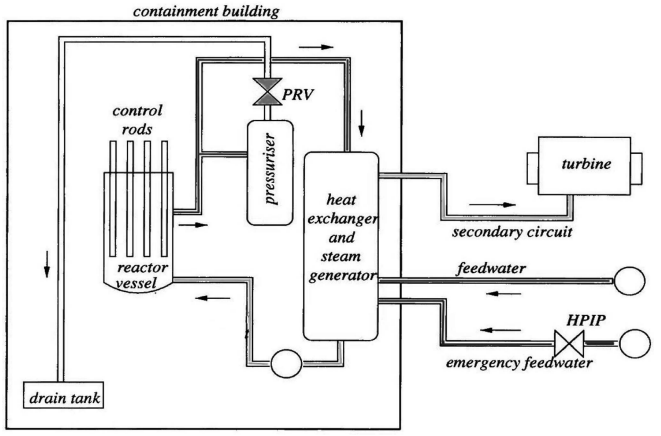

Fig. 13. Three Mile Island 2 nuclear reactor.

We summarize the events that led to the Three Mile Island accident as follows: Due to some manual errors, the feedwater valve in the secondary circuit was closed; heat exchange was thus stopped, resulting in a temperature increase in the primary circuit. As expected, the pressure relief valve (PRV) was opened and the control rods were dropped to slow down the nuclear fission. This is the *loss of coolant accident* (LOCA). When the pressure and temperature dropped, the PRV should have closed and control rods lifted. However, the PRV was stuck open and this was not detected by the solenoid (sensor). As a result, the coolant from the primary circuit kept draining, thus endangering its environment with possible radiation.

This well-known example is used to illustrate how Safecharts can also be used to detect a known fault, that is, the PRV stuck-opened fault, which is conventionally known as fault injection [4] or error seeding [29]. The fault is modeled into Safecharts and then detected by model checking the Safecharts against safety properties.

Safecharts [22] can be used to model the components in the TMI nuclear reactor and, since Safecharts can explicitly model all of the failure modes of a component such as the stuck-open failure mode of the PRV, we can use Safecharts

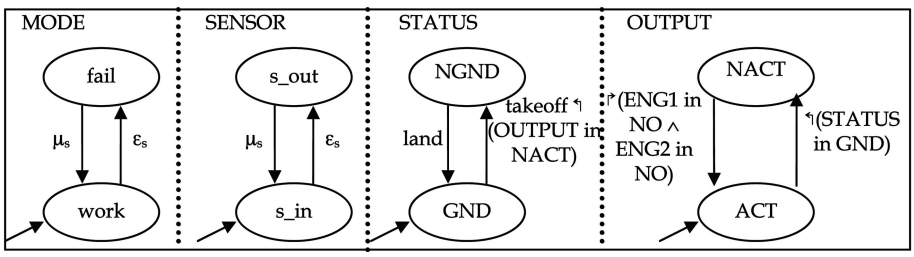

Fig. 12. Integrated Safecharts model for a controller block in A320.

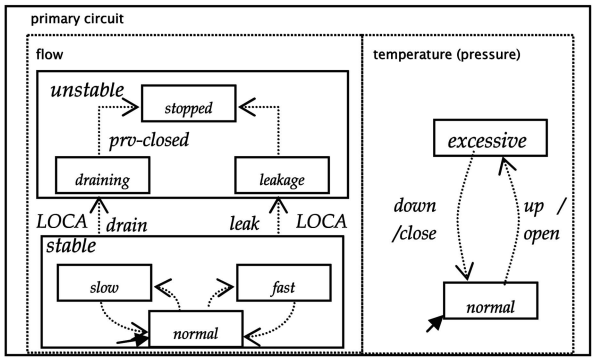

Fig. 14. Safechart model of primary circuit at Three Mile Island.

to check for such safety violations in the model itself. Fig. 15 illustrates the Safechart model for PRV, where we can see that the PRV stuck-open failure is detected by the sensor in the model; however, in the actual TMI reactor, during the accident, the PRV-close transition was taken and, thus, the sensor could not detect the failure of the PRV to close. Fig. 14 illustrates the Safechart model for the primary circuit in the TMI nuclear reactor along with LOCA. The hazard that was checked for this model is EF(status in stuck-opened and PRV position in PRV-closed and flow in draining). Any witness to the satisfaction of this TCTL property generated by the SGM model checker was evidence that the hazard could happen in this model. There were two Safecharts with 27 states and 28 transitions, which were transformed into six ETA models with 19 modes and 30 transitions. As shown in Section 6.4, several state-space reduction techniques were implemented into the SGM model checker to successfully verify this system.

6.4 State-Space Reduction

The state-space reduction techniques proposed for safety-critical systems in Section 5.2 were all applied to the three examples described in this section. Experiments were

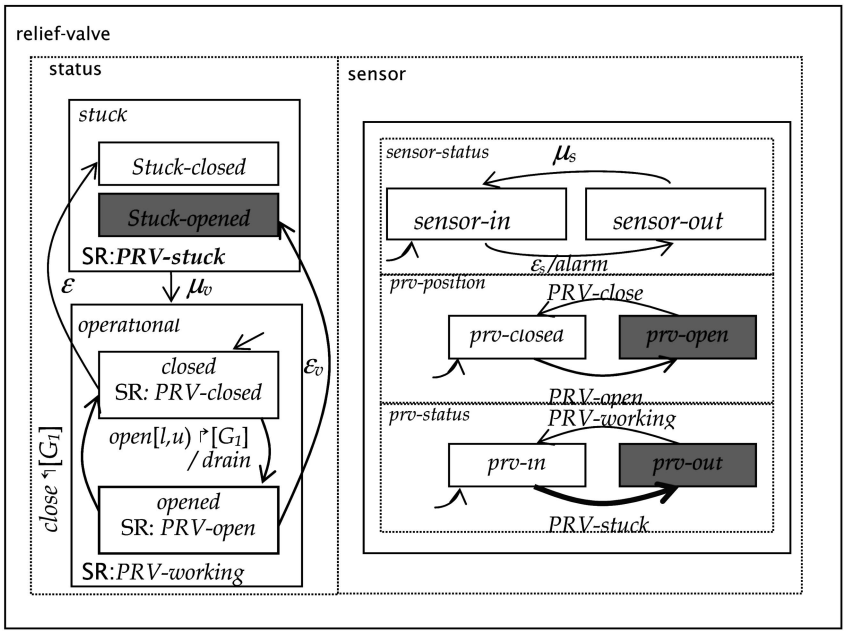

Fig. 15. Safechart model of PRV at Three Mile Island.

TABLE 3
State-Space Reduction for A320 Hydraulic System (Blue Line)

Reductions	#C	|M|	|T|	#Modes	#Trans	MB	sec
0	5	39	50	>429K	>5117K	Out of memory	
1	5	39	50	21,055	230,724	790.3	1344.0
2	1	39	50	5,184	55,488	135.0	95.4
3	1	37	44	3,456	32,576	91.4	49.4
4	1	34	41	1,024	9,216	28.0	10.1

0: No reduction, 1: VCT, 2: VCT+CNR, 3: $\mathrm{VCT}+\mathrm{CNR}+\mathrm{STR}$, *4:* $\mathrm{VCT}+\mathrm{CNR}+\mathrm{STR}+\mathrm{STA}$, *VCT: variable usage and clock hiding technique, CNR: clock number reduction, STR: safety transition reduction, STA: safe time abstractions, #C: number of clocks.*

TABLE 4
State-Space Reduction for TMI Nuclear Reactor

Reductions	#C	|M|	|T|	#Modes	#Trans	MB	sec
0	1	19	30	96	270	1.93	0.40
1, 2, 3	1	19	30	61	199	3.48	0.55
4	1	18	29	38	128	2.96	0.44

TABLE 5
Comparison of Different Verification Methods for Safety-Critical Systems

Feature	Z99 [29]	YC+00 [28]	BCS02 [2]	J02 [15]	BV03 [4]	KS+04 [16]	Safe-charts
System Model	Promela	PCED	Altarica	UML	NuSMV	Altarica	UML/ Safechart
Fault Modeling	Seeding Errors	Fault Mode	User Modeled	None	Fault Injection	User Modeled	Built-In
Timing Verification	No	No	No	No	No	No	Yes
Verification Target	Operator Procedure	Safety Property	Activation Property	Hazard Property	Safety Property	Safety Property	Safety Property
Traditional Methods	None	None	FTA	Simulation	FTA, FOA	OCAS Simulation	None
Model Checker	SPIN	SMV	Cadence SMV	SMV	NuSMV2	Cadence SMV	SGM
State-space Reduction	Incremental	No	No	No	No	No	Yes

conducted on a Linux machine with a Pentium 4 2.8-GHz CPU and a 2-Gbyte RAM and swap space. The results of applying the reduction techniques to the blue subsystem of the A320 hydraulic system are tabulated in Table 3, where we observe that, when the techniques are applied incrementally, the state-space sizes are reduced significantly such that model checking becomes feasible, whereas the global state graph could not be constructed without the techniques.

For the Three Mile Island nuclear reactor example, the results are tabulated in Table 4, where we can see that the model being moderately sized, the effects of reduction were not significant; however, it did not matter as the model was feasible for model checking.

6.5 Evaluation and Validation of Safecharts Model Checking

The proposed method of Safecharts model checking for verifying safety-critical systems can be evaluated and validated by comparing it with other methods. However, for safety-critical systems, it is not always possible to do an exact quantitative comparison, so we performed a qualitative comparison, as summarized in Table 5.

In comparison with other methods in Table 5, the advantages provided by the Safecharts model checking include the following: 1) Fault-modeling techniques are built-in within the popular UML design models, 2) timing verification is performed as we use an ETA model checker, namely, SGM, and 3) several state-space reduction techniques are employed to verify large system models.

7 Conclusions

Nowadays, safety-critical systems are becoming more and more pervasive in our daily lives. To reduce the probability of tragedy, we must have a formal and accurate methodology to verify if 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 widely applied to safety-critical systems with a model-driven architecture. Through several examples, we have shown the benefits of the proposed verification method and system model. We hope our methodology can have some real contribution such as making the world a safer place to live.

References

[1] K. Altisen, G. Gössler, and J. Sifakis, "Scheduler Modeling Based on the Controller Synthesis Paradigm," *Real-Time Systems*, vol. 23, pp. 55-84, 2002.

[2] P. Bieber, C. Castel, and C. Seguin, "Combination of Fault Tree Analysis and Model Checking for Safety Assessment of Complex System," *Proc. Fourth European Dependable Computing Conf.*, pp. 19-31, 2002.

[3] M. Bozzano, A. Cavallo, M. Cifaldi, L. Valacca, and A. Villafiorita, "Improving Safety Assessment of Complex Systems: An Industrial Case Study," *Proc. Int'l Formal Methods Europe Symp.*, pp. 208-222, 2003.

[4] M. Bozzano and A. Villafiorita, "Improving System Reliability via Model Checking: The FSAP/NuSMV-SA Safety Analysis Platform," *Proc. Int'l Conf. Computer Safety, Reliability and Security*, pp. 49-62, 2003.

[5] E.M. Clarke and E.A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic," *Proc. Logics of Programs Workshop*, pp. 52-71, 1981.

[6] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking*. MIT Press, 1999.

[7] H. Dammag and N. Nissanke, "Safecharts for Specifying and Designing Safety Critical Systems," *Proc. 18th IEEE Symp. Reliable Distributed Systems*, pp. 78-87, 1999.

[8] J. Deneux and O. Akerlund, "A Common Framework for Design and Safety Analyses Using Formal Methods," *Proc. Int'l Conf. Probabilistic Safety Assurance and Management (PSAM) and European Safety and Reliability Conf.*, 2004.

[9] European Union, "Enhanced Safety Assessment for Complex Systems (ESACS) Project," http://www.esacs.org, 2003.

[10] P. Fenelon, J.A. McDermid, M. Nicholson, and D.J. Pumfrey, "Towards Integrated Safety Analysis and Design," *Applied Computing Rev.*, vol. 2, pp. 21-32, 1994.

[11] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," *Proc. IEEE Int'l Conf. Logics in Computer Science*, pp. 394-406, 1992.

[12] P.-A. Hsiung and S.W. Lin, "Model Checking Timed Systems with Priorities," *Proc. Int'l Conf. Real-Time and Embedded Computing Systems and Applications*, pp. 539-544, 2005.

[13] P.-A. Hsiung and F. Wang, "A State-Graph Manipulator Tool for Real-Time System Specification and Verification," *Proc. Fifth Int'l Conf. Real-Time Computing Systems and Applications*, pp. 181-188, 1998.

[14] J. Jacky, "Formal Safety Analysis of the Control Program for a Radiation Therapy Machine," *Proc. 13th Int'l Conf. Use of Computers in Radiation Therapy*, pp. 68-70, 2000.

[15] M.E. Johnson, "Model Checking Safety Properties of Servo-Loop Control Systems," *Proc. Int'l Conf. Dependable Systems and Networks*, pp. 45-50, 2002.

[16] C. Kehren, C. Seguin, P. Bieber, C. Castel, C. Bougnol, J.-P. Heckmann, and S. Metge, "Advanced Multi-System Simulation Capabilities with AltaRica," *Proc. Int'l System Safety Conf. (ISSC '04)*, pp. 489-498, Aug. 2004.

[17] K.G. Larsen, B. Steffen, and C. Weise, "Fischer's Protocol Revisited: A Simple Proof Using Model Constraints," *Hybrid System III,* pp. 604-615, 1996.
[18] *A Methodology for Formalizing Concepts Underlying the DESS Notation,* L. Lavazza, ed. ITEA, 2001.
[19] N.G. Leveson, *Safeware: System Safety and Computers.* Addison-Wesley, 1995.
[20] N.G. Leveson and L. Stolzy, "Safety Analysis Using Petri Nets," *IEEE Trans. Software Eng.,* vol. 13, no. 3, pp. 386-397, Mar. 1987.
[21] S.-W. Lin and P.-A. Hsiung, "Model Checking Prioritized Timed Automata," *Proc. Int'l Symp. Automated Technology for Verification and Analysis,* pp. 370-384, Oct. 2005.
[22] N. Nissanke and H. Dammag, "Risk Bands—A Novel Feature of Safecharts," *Proc. 11th Int'l Symp. Software Reliability Eng.,* pp. 293-301, 2000.
[23] N. Nissanke and H. Dammag, "Risk Ordering of States in Safecharts," *Proc. 19th Int'l Conf. Computer Safety, Reliability, and Security,* pp. 395-405, 2000.
[24] N. Nissanke and H. Dammag, "Design for Safety in Safecharts with Risk Ordering of States," *Safety Science,* vol. 40, no. 9, pp. 753-763, 2002.
[25] J.-P. Queille and J. Sifakis, "Specification and Verification of Concurrent Systems in CESAR," *Proc. Int'l Symp. Programming,* pp. 337-351, 1982.
[26] I. Sommerville, *Software Engineering,* seventh ed. Addison-Wesley, 2004.
[27] F. Wang and P.-A. Hsiung, "Efficient and User-Friendly Verification," *IEEE Trans. Computers,* vol. 51, no. 1, pp. 61-83, Jan. 2002.
[28] S.H. Yang, P.W. Chung, S. Kowalewski, and O. Stursberg, "Automatic Safety Analysis of Computer Controlled Plants Using Model Checking," *Proc. 10th Symp. Computer-Aided Process Eng. (ESCAPE 10),* 2000.
[29] W. Zhang, "Model Checking Operator Procedures," *Proc. Fifth and Sixth Int'l Software Process Improvement Network (SPIN) Workshops Theoretical and Practical Aspects of SPIN Model Checking,* pp. 200-215, 1999.

Pao-Ann Hsiung received the BS degree in mathematics and the PhD degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, Republic of China (ROC), in 1991 and 1996, respectively. From 1996 to 2000, he was a postdoctoral researcher at the Institute of Information Science, Academia Sinica, Taipei. From February 2001 to July 2002, he was an assistant professor at the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, ROC, where he is currently an associate professor. He was the sole recipient of the 2001 ACM Taipei Chapter Kuo-Ting Li Young Researcher Award for his significant contributions to the design automation of electronic systems. He was also a recipient of the 2004 Young Scholar Research Award given by the National Chung Cheng University. He is a member of the IEEE, the IEEE Computer Society, and the ACM, and a life member of the IICM. He has been included in several professional listings such as *Marquis' Who's Who in the World*, *Who's Who in Asia*, *Outstanding People of the 20th Century*, and *Rifacimento International's Admirable Asian Achievers*. He is on the editorial board of the *International Journal of Embedded Systems* (Inderscience Publishers) and guest edited two special issues in 2004 for that journal. He has been on the program committee of several international conferences. He has published more than 120 papers in international journals and conferences. He has been taking an active part in paper refereeing for international journals and conferences. His main research interests include System-on-Chip (SoC) design and verification, embedded software synthesis and verification, real-time system design and verification, hardware-software codesign and coverification, and component-based object-oriented application frameworks for real-time embedded systems.

Yean-Ru Chen received the BS degree in computer science and information engineering from the National Chiao Tung University, Hsinchu, Taiwan, Republic of China (ROC), in 2002 and the MS degree in computer science and information engineering from the National Chung Cheng University, Chiayi, Taiwan, ROC, in 2006. She is currently working toward the PhD degree at the Graduate Institute of Electronics Engineering of the National Taiwan University, Taipei, Taiwan, ROC. From 2002 to 2003, she was employed as an engineer at the SoC Technology Center, Industrial Technology Research Institute, Hsinchu, Taiwan, ROC. Her current research interests include model checking, safety-critical systems, coverification, and Electronic System Level (ESL) design.

Yen-Hung Lin received the BS degree in computer science and information engineering from the National Chung Cheng University, Chiayi, Taiwan, Republic of China (ROC), in 2005. He is currently working toward the MS degree at the National Chiao Tung University, Hsinchu, Taiwan, ROC. His current research interests include VLSI CAD, safety-critical systems, and formal verification.