SOFTWARE SAFETY ANALYSIS OF DIGITAL PROTECTION SYSTEM REQUIREMENTS USING A QUALITATIVE FORMAL METHOD

NUCLEAR PLANT OPERATIONS AND CONTROL

KEYWORDS: *software requirements, safety analysis, formal method*

JANG-SOO LEE* and KEE-CHOON KWON
Korea Atomic Energy Research Institute, MMIS Team, 150, Duckjin-dong Yuseong-gu, Daejeon 305-353, Korea

SUNG-DEOK CHA *Korea Advanced Institute of Science and Technology Computer Science Division, EECS Department and Advanced Information Technology Research Center (AITrc) 373-1, Kusong-dong, Yuseong-gu, Daejeon 305-701, Korea*

Received October 17, 2001
Accepted for Publication October 3, 2003

The safety analysis of requirements is a key problem area in the development of software for the digital protection systems of a nuclear power plant. When specifying requirements for software of the digital protection systems and conducting safety analysis, engineers find that requirements are often known only in qualitative terms and that existing fault-tree analysis techniques provide little guidance on formulating and evaluating potential failure modes. A framework for the requirements engineering process is proposed that consists of a qualitative method for requirements specification, called the qualitative formal method (QFM), and a safety analysis method for the requirements based on causality information, called the causal requirements safety analysis (CRSA). CRSA is a technique that qualitatively evaluates causal relationships between software faults and physical hazards. This technique, extending the qualitative formal method process and utilizing information captured in the state trajectory, provides specific guidelines on how to identify failure modes and the relationship among them. The QFM and CRSA processes are described using shutdown system 2 of the Wolsong nuclear power plants as the digital protection system example.

I. INTRODUCTION

A hybrid system consists of a discrete program within a continuous plant that is governed by the laws of physics. Examples of hybrid systems include the satellites, the missiles, and the digital protection systems embedded in nuclear power plants. Obviously, safety is vitally important for the hybrid systems. The traditional safety analysis methods,[1–3] however, allow us to analyze the safety of the control software and the plant separately. The primary goal of a safety analysis is to demonstrate that the system as a whole will not result in hazardous behavior. Since safety is the system property, proper software safety analysis must always be conducted within the context of the environment that it is required to control; i.e., safety analysis must demonstrate the truth of the proposition (P and $C \rightarrow Sp$), where P, C, and Sp refer to the behavioral model of the plant, controller, and requirements for control software, respectively.

In this paper, we propose a safety analysis method of the hybrid system requirements, based on a practical formal method.[4] Although there are many formal methods[4–8] for hybrid systems with their different levels of rigor, practical approaches[4,6] for specifying and analyzing systems containing both discrete and continuous quantities are lacking. The major obstacle to using formal methods for hybrid real-time systems in industry is the difficulty that engineers have in understanding and applying the quantitative methods in the abstract requirements phase. There should be a formal yet abstract method to specify

*E-mail: jslee@kaeri.re.kr

and validate the hybrid system requirements effectively. It is the interaction of continuous and discrete state changes that make requirements specification and safety analysis of a hybrid system challenging.

Fault-tree analysis (FTA) is arguably the most widely used safety analysis technique in industry. Advantages include ease of understanding due to graphical representation and ability to flexibly describe various failure scenarios such as hardware failures, software errors, operator mistakes, poorly designed human interfaces, or events in the environment external to the system. On the other hand, the technique itself is nothing more than a notation to systematically organize analysis done by safety engineers.[1] Therefore, effectiveness of the technique is highly dependent on the ability of safety engineers, and the fault trees can be different from one safety engineer to another even though they are based on the same information. In addition to informality inherent in fault-tree analysis, the difficulty of building fault trees during early phases of a hybrid system development is compounded because one must analyze complex interaction patterns involving discrete and continuous events stated in abstract terms.

Several techniques have been proposed to perform safety analysis. Most safety analysis techniques were initially proposed to perform backward and causality analysis at the system level, but some have been adapted to software. As discussed in Ref. 9, an authoritative survey on the subject, there are many safety analysis techniques, including checklists; FTA; event-tree analysis; cause-consequence analysis; failure modes and effects analysis; failure modes, effects, and criticality analysis; and hazards and operability analysis.

Fault-tree analysis has been the subject of most safety research in the past. Leveson and Harvey[3] adapted system-level fault trees to software analysis, and they used sequential program structures and their failure semantics to derive fault-tree nodes. The technique has since been applied to all phases of a software development life cycle, including requirements engineering,[7,10] design,[11–13] and concurrent and object-oriented codes.[2,14,15] In Ref. 7, specifications are written in real-time interval logic, and fault trees are derived from a temporal logic formula capturing the causal relationship between states. Gorski and Wardzinski,[16] similar in approach to Ref. 17, attempted to formalize fault trees. Subramanian et al.[13] discussed how to analyze software safety in the requirements engineering and design phases using state charts. When evaluated in the context of performing safety analysis for a hybrid system, past research on fault-tree analysis is inadequate because

1. only the discrete aspects of a behavioral model can be modeled and analyzed
2. guidelines or heuristics used in fault-tree generation do not work well in the early phase of requirements engineering when abstract requirements are given.

In Refs. 4 and 18, we proposed a qualitative formal method (QFM) for building models corresponding to the behavior of the plant and the controller. In particular, we adapted research results in artificial intelligence, qualitative physics, to formally express qualitative terms. Behavioral models of the control software and the plant are expressed in notations known as the causal functional representation language[19] (CFRL) and the compositional modeling language[20] (CML). These languages allow one to express inherently qualitative state changes such as the temperature rising and the possible consequences in a manner where formal and causal reasoning can be performed. Outcome of the QFM process is called the state trajectory, which is analogous to the reachability graph of the Petri Net models. State trajectory can be automatically generated using the device modeling environment (DME) tool,[21] which contains useful information to assist safety engineers in building fault trees in a systematic manner.

In this paper, we extend the QFM process and describe how to generate fault trees. We refer to the extended technique as causal requirements safety analysis (CRSA). Although the process is not fully automated, much of the fault-tree skeleton can be automatically derived from information on causal relationship captured in the state trajectory and qualitative formal specification. We describe step-by-step procedures of applying CRSA to perform a safety analysis of Wolsong shutdown system (SDS) 2, which is an emergency SDS for a nuclear power plant currently in service in Korea.

The remainder of our paper is organized as follows. In Sec. II, we propose a requirements engineering process with QFM and CRSA and briefly review our QFM process.[4] The Wolsong SDS2 system is also briefly explained. In Sec. III, we discuss in detail the procedures of applying CRSA and building fault trees. Section IV concludes the paper.

II. THE REQUIREMENTS ENGINEERING PROCESS

An engineer who specifies and verifies the requirements of hybrid system software must possess knowledge of requirements engineering techniques as well as domain-specific principles. The challenge, however, is how to define a requirements engineering framework that allows designers to specify and verify the requirements without misunderstanding. To reduce the burden in understanding completely the physical process when eliciting, specifying, and verifying the requirements, we propose a requirements engineering process for hybrid systems that relies on the idea from qualitative reasoning[22] of artificial intelligence (see Fig. 1).

II.A. Software Requirements Specification (QFM1)

The SDS2 of Wolsong nuclear power plant in Fig. 2 has been designed to rapidly terminate the nuclear chain

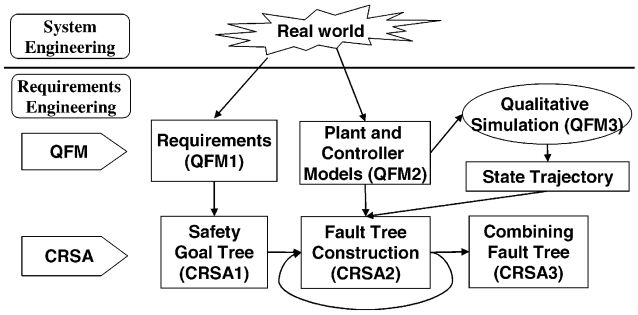

Fig. 1. The QFM and CRSA process.

reaction and keep the reactor in a safe state upon the detection of various unsafe operating conditions. To provide fault-tolerance capability, the trip decision for SDS2 is made using general coincidence two-out-of-three channel voting logic. Although there are several trip parameters defined for SDS2, we use a steam generator low level (SGLL) trip as an example whose behavior is informally described as follows.

SDS2 operates by deenergizing relays, after a trip condition is detected by the trip logic, that initiates the opening of quickly opening valves, which causes high-pressure helium to push neutron-absorbing liquid poison from the tank into the core via injection lines in the primary coolant. Steam generators function by transferring energy from the primary coolant to the turbines, where it is converted to electricity. As such, they serve as a heat sink for the reactor core. For proper performance, the steam generator water level must be held within predetermined operating bounds. Too low a level may portend inadequate heat removal from the core, while too high a level may degrade the steam quality.

In the first step of the QFM process, we specify the required system behaviors using CFRL (Ref. 19), which is a language for specifying a required function of a device. The required function F for a hybrid system can be specified with four-part Ef, Df, Cf, and Gf, where

Ef = name of a required function that *F* elaborates

Df = system of which *F* is a required function

Cf = context in which the function is performed

Gf = functional goal to be achieved.

The CFRL notation captures the system components and the goal properties, such as safety requirements, that

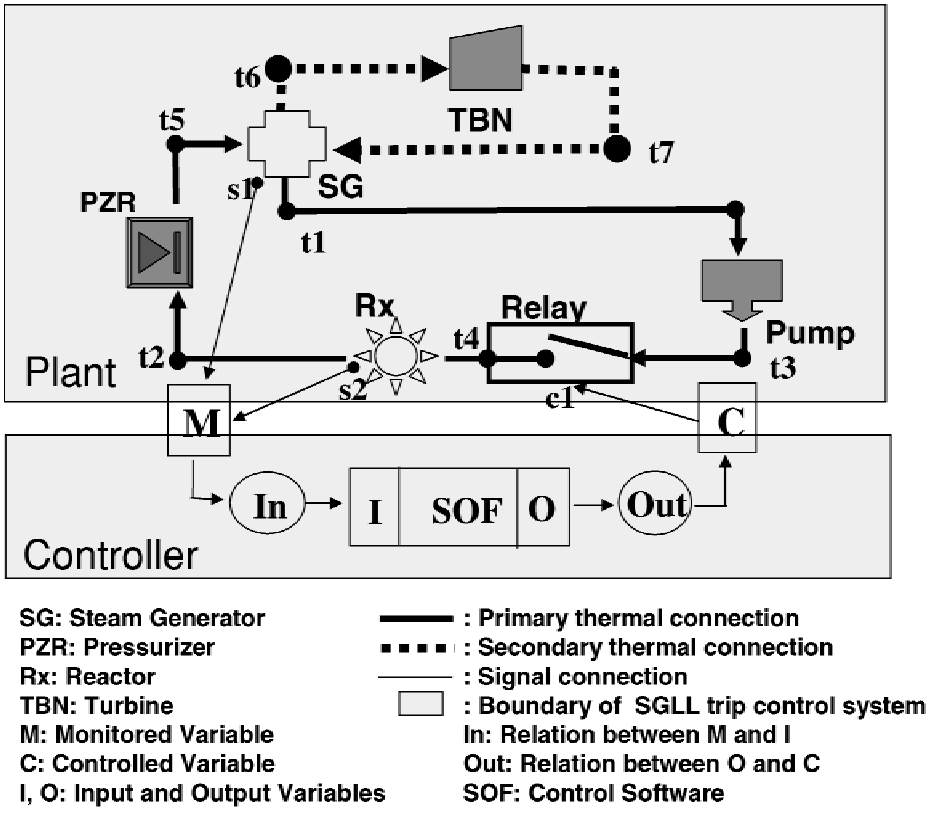

Fig. 2. Shutdown system 2.

```
Ef: F1   /* function name of the SDS2 system */          /* line 1 */
Df: Device: (?sglltcs SGLL-trip-control-system)          /* line 2 */
Cf: Objects: /* context of the system */
   (?pzr Pressurizer)
   (?pump Charging-pump)
   (?t-load Thermal-load-of-2nd-loop-with-turbine)
  Conditions:
   (Thermally-connected (Thermal-out ?sglltcs) (Thermal-in ?t-load))
   (Thermally-connected (Thermal-in ?sglltcs) (Thermal-out ?t-load))
Gf: (ALWAYS (AND      /* goal of the controller */
     (IMPLIES (AND ( (CAvgPower ?rx) 10 %FP)
          (> (Level ?sg) $LSP$) (Closed-p ?relay))
          CPD1)  /* subgoal 1: normal heat-sinked */
     (IMPLIES (AND ( (CAvgPower ?rx) 10 %FP)
          ((Level ?sg) $LSP$) (Closed-p ?relay))
          CPD2)  /* subgoal 2: trip when SGLL */
     (IMPLIES (AND (< (CAvgPower ?rx) 10 %FP) (Closed-p ?relay))
          CPD3))) /* subgoal 3: condition out trip signal */
```

Fig. 3. Requirements specification by CFRL.

the system must satisfy. For example, requirements for the SGLL trip control system of SDS2 in Fig. 2 include the system structure and the causal relations of the safety requirements.

The structural system description of Df consists of three parts: device, components, and conditions. Device and its component are described as a pair (?var, type) shown in Fig. 3, where var is the symbol to be used in the description of the function; the type is the class of the var. The conditions typically specify aspects of the system structure that are assumed in the description of the required function. The notion of a system function assumes some physical context in which the system is placed, and Cf is a specification of such context. The description of the required safety properties, Gf, is represented as an expression consisting of a causal process description (CPD), quantifiers, and Boolean connectives. There are two quantifiers, ALWAYS and SOMETIMES. Connectives are AND, OR, IMPLIES, and NOT. CPD, a directed graph shown in Fig. 4, is essentially a state machine in which each node describes a condition on a state, and each arc describes temporal and causal relations between states.

The condition in a CPD node is a logical sentence about the state of the world at some time using the variables defined in the Df and Cf portions of the function description. One or more nodes in each CPD are distinguished as the initial node(s). The initial nodes are indicated by a thick oval. The CPD of the CFRL can represent a causal and temporal relationship in the required behavior for the SGLL controller of SDS2.

The goal Gf of F1 means that when the reactor generates power as a heat source and the primary loop is closed, the steam generator must act as a heat sink. When the reactor starts to generate power and the reactivity of the reactor increases, the goal Gf of F1 specifies the desired behavior of the trip controller, i.e., shutdown and condition out. The CPD specifies the desired sequence of states abstractly. Unlike the total state trajectory of the plant and the controller (P and C), it only specifies the facts that must be true during the course of the trajectory and causal orderings among those facts.

II.B. Plant and Controller Specification (QFM2)

Behavioral models of the plant, controller, and their interface are written in CML (Ref. 20). The CML is a declarative model-building language for logically specifying the symbolic and mathematical properties of the structure and behavior of physical systems. The CML specifies abstract behavior of the plant using qualitative differential equations and causality relation. The physical situation is modeled as a general-purpose domain theory as a collection of model fragments, each of which represents a physical object or a conceptually distinct

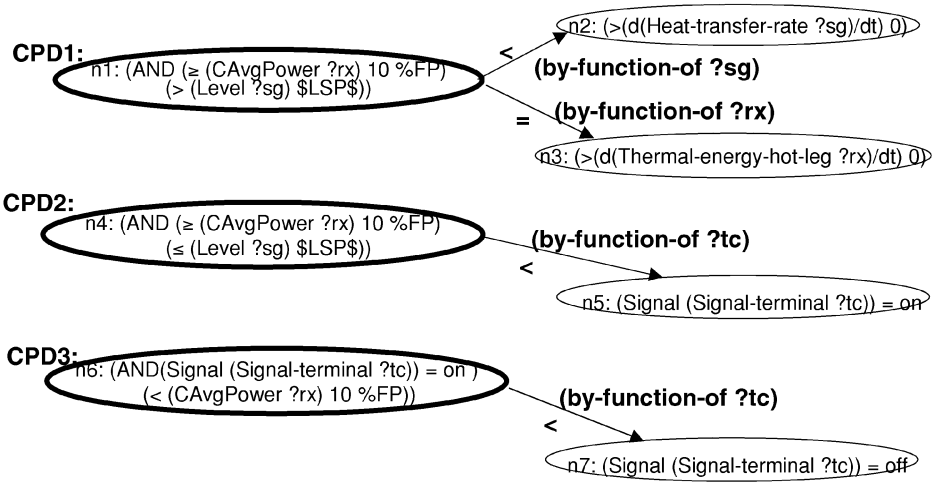

Fig. 4. Safety requirements by CPDs.

physical phenomenon. A model fragment specifies a set of conditions under which a phenomenon would occur and the consequences. The conditions are indicated as Pm and specify a set of instances of object classes that must exist, called participants, and a set of relations, called conditions, that must hold among those objects and their attributes. Their consequences are indicated as Cm for model fragments representing continuous phenomena and as Dm for model fragments representing discrete events.

The structural information of SDS2 as shown in Fig. 2 is described as a collection of model fragments representing components and their connections. These model fragments represent the static aspect of the situation, and they are always active. In addition, there are model fragments representing various behavioral aspects of the components. Figures 5 and 6 show the sets of model fragments specifying the plant and controller models of the shutdown system for the Wolsong nuclear power plants.

For example, the first model fragment definition (SG) in Fig. 5 defines the behavior of a steam generator. The thermal energy of the generated steam increases monotonically according to the thermal energy of the primary coolant. The steam generator has a normal heat transfer

Behavior of Steam Generator
Steam-Generator (SG)
Pm: (Steam-generator ?s)
Cm: (Thermal-energy-steam-out ?s) = M_+(Thermal-energy-primary-in ?s)
∧ (Heat-transfer-rate ?s) = M_+(Level ?s)
∧ (Level ?s) = IL - SO + FI + M_+(Thermal-energy-primary-in ?s)
SG-operating-in-low-reactor-power (SL)
Pm: (Steam-generator ?s) ∧ (Reactor ? r) ∧ (Level ?s) > LSP ∧ (CAvgPower ?r)
< 10 %FP
Cm: (Thermal-energy-feed-in ?s) = 0 /* no recirculation */
SG-operating-in-normal-reactor-power (SN)
Pm: (Steam-generator ?s) ∧ (Reactor ? r) ∧ (Level ?s) > LSP
∧ 10%FP =< (CAvgPower ?r) < 90 %FP
Cm: (Heat-transfer-rate ?s) = 35 %
SG-operating-in-high-reactor-power (SH)
Pm: (Steam-generator ?s) ∧ (Reactor ? r) ∧ (Level ?s) > LSP ∧ (CAvgPower ?r)
>= 90 %FP
Cm: (Thermal-energy-steam-out ?s) = $Max-S$

Behavior of Secondary Loop
Thermal-load (TL)
Pm: (Steam-generator ?s)
Cm: (Thermal-energy-feed-in ?s) = (Thermal-energy-steam-out ?s) - $Work$ - $Tloss2$

Behavior of Reactor
Reactor-thermal-energy-generating (RT)
Pm: (Reactor ? r) ∧ (In-closed-thermal-loop ?r)
Cm: (Thermal-energy-hot-leg ?r) = M_+(d(CAvgPower ?r)/dt)
- $Tloss1$ + (Thermal-energy-cold-leg ?r)
Reactor-in-open-loop (RO)
Pm: (Reactor ? r) ∧ ¬ (In-closed-thermal-loop ?r)
Cm: (Thermal-energy-hot-leg ?r) = 0

Behavior of Trip Relay
Relay-closed (TC)
Pm: (Steam-generator ?s) ∧ (Reactor ? r) ∧ (Trip-relay ?t) ∧ (Closed-p ?t)
Cm: (Thermal-energy-hot-leg ?r) + (Thermal-energy-primary-in ?s) = 0
Relay-opened (TO)
Pm: (Reactor ? r) ∧ (Trip-relay ?t) ∧ (Open-p ?t)
Cm: (Thermal-energy-hot-leg ?r) = 0
Relay-closing (CL)
Pm: (Trip-relay ?t) ∧ (Open-p ?t) ∧ ¬ (Signal-on (Sig-terminal ?t))
Dm: (Closed-p ?t)
Relay-opening (OP)
Pm: (Trip-relay ?t) ∧ (Closed-p ?t) ∧ (Signal-on (Sig-terminal ?t))
Dm: (Open-p ?t)

Fig. 5. Model fragments of the plant specified by the CML.

Behavior of Trip Controller
Trip-signal-on (TN)
Pm: (Reactor ? r) ∧ (Trip-controller ?tc) ∧ (Signal (Signal-terminal ?tc)) = off
∧ (LogPower ?r) >= 2%FP ∧ (CAvgPower ?r) >= 10 %FP
∧ (Level ?s) =< LSP
Dm: (Signal (Signal-terminal ?tc)) = on
Trip-condition-out (TT)
Pm: (Reactor ? r) ∧ (Trip-controller ?tc) ∧ (Signal (Signal-terminal ?tc)) = on
∧ (LogPower ?r) < 2%FP ∧ (CAvgPower ?r) < 10 %FP
Dm: (Signal (Signal-terminal ?tc)) = off

Fig. 6. Model fragments of the controller specified by the CML.

rate between 10% full power (FP) and 90% FP (SN), and the heat transfer rate is saturated when the power becomes >90% FP (SH). The quantity space of the steam generator operating region can be represented as {0, 10, 90, 120}. Here, 10 and 90 are the landmarks that represent the key changing points of the physical states.

II.C. Simulation of the System Behavior (QFM3)

We used a DME (Ref. 21) to produce the state trajectory, as shown in Table I, of the models for SDS2. To perform simulation, causal ordering among variables in the qualitative equations must be decided. Given the CML representations of the plant and controller models, DME simulates the behavior of the models as follows: generates the qualitative equation model, gives a causal ordering between variables from equations, and produces the state trajectory of the models. The ordering is total for the states in the state trajectory of Table I, while the order is partial for states in a CPD of Fig. 2. Note that state trajectory describes the values of state variables in terms of their magnitudes or direction of changes (e.g., rise or fall). In a state where their derivative is undefined, the sign is shown as x. Column 1 of the table shows the set of active models in each state. The set of all active model fragments in each state is actually much larger, but since most of them represent components, terminals, junctions, etc., and are active throughout the simulation, the table shows only the ones that change their activation status at some point. A model fragment that becomes activated in a given state is shown in bold. A model fragment with an underbar indicates that it becomes deactivated in the given state.

In this simulation, we assume that the thermal energy in the primary thermal connection (T1 and T5) is measured by temperature, the steam-out thermal energy (T6) is calculated by the electrical energy produced by the steam, the feedwater thermal energy (T7), by the flow rate and temperature, and the unit of compensated average power (CAP) of the reactor is the percent of full power. To simplify the simulation, we assume also that the low water level setpoint of the steam generator is constant (30%).

In the initial state of the simulation, we assumed that the water level of the steam generator is 50%, the reactor is in heatup mode, the trip relay is closed, and that the reactor power is between 0% FP and 120% FP. In the initial state S0, the thermal energy T1 and T5 increase at t1 and t5, respectively, in Fig. 2. When the primary coolant temperature reaches 292 (S1), the reactor enters the startup mode, and then the secondary thermal connection starts to produce steam (S2). At this point, the feedwater flow increases, the water level of the steam generator falls under the setpoint because of a fluctuation by the swell-and-shrink effects, and the trip signal occurs (S4). However, because the reactor power (CAP) at this state is <10%, the trip signal is conditioned out (S5). The reactor power increases continuously during the power operation mode.

When CAP reaches 10% FP, the steam generator enters the normal operating state (S6), and when it reaches 90% FP, the steam generator enters the high operating state (S8). From the high operating state, the thermal energy of the primary and secondary connections and the water level become steady state (S9). At this point, we assumed that the water level of the steam generator could be under the setpoint (30%) because of some accidents like loss-of-coolant accidents or loss of main feedwater. When the water level falls under 30%, the trip signal On occurs (S14), the trip relay opens (S15), and the reactor enters shutdown state (S16). After shutdown, the thermal energy of the primary connection increases because of the residual heat of the reactor (S17).

III. THE CRSA PROCESS

As noted earlier, the CRSA process is an extension of the QFM process proposed earlier,[4,18] and the overall process is shown in Fig. 1. The step-by-step procedure of the CRSA process is as follows.

III.A. Top Event Identification (CRSA1)

Fault-tree analysis begins by identifying the top-level events whose consequences are considered hazardous and therefore must be avoided. A safety goal tree, specified in the Gf section of the CFRL specification in the QFM1 process and shown in Fig. 3, is used to identify

TABLE 1

State Trajectory of SDS2

Active Models	State	Reactor	Relay	Signal	T1	T5	T6	T7	Level	CAP
SL, *TC*, *RH*	**S0**	Heatup	Close	Off	60-292 inc	60-326 inc	0-1000 std	0-840 std	50 std	0-120 std
SL, TC, RH	**S1**	↓	↓	↓	292 inc	↓	0 inc	0 inc	↓	↓
SL, TC, **RS**	**S2**	*Startup*	↓	↓	↓	292 inc	↓	↓	50 dec	↓
SL, TC, RS, **FL**	**S3**	↓	↓	↓	↓	↓	↓	15 inc	30 dec	2 inc
SL, TC, RS, **TN**, FL	**S4**	↓	↓	On	292 std	↓	50 inc	↓	30 inc	↓
SL, TC, RS, TN, **TT**, **FH**	**S5**	↓	↓	Off	↓	294 inc	↓	↓	↓	10 inc
SL, TC, RS, TT, FH	**S6**	↓	↓	↓	↓	↓	↓	↓	50 std	90 inc
SL, TC, **RN**, FH	**S7**	Normal	↓	↓	↓	326 std	1000 std	840 std	30–50 dec	90–100 inc
SL, **SN**, TC, RN, FH	**S8**	↓	↓	↓	↓	↓	↓	↓	30 dec	↓
SN, TC, FH, RN	**S9**	↓	↓	↓	↓	↓	↓	↓	20–30 dec	↓
SH, TC, FH, RN	**S10**	↓	↓	↓	↓	↓	↓	↓	↓	90–100 dec
SH, TC, FH, RN,	**S11**	↓	↓	↓	↓	↓	↓	↓	↓	90 dec
SH, TC, **FL**, RN	**S12**	↓	↓	↓	↓	↓	↓	↓	↓	↓
SH, TC, RN, FL, **TN**	**S13**	↓	↓	↓	↓	↓	↓	↓	↓	↓
SH, TC, RN, TN, FL, **OP**	**S14**	↓	↓	On	↓	↓	↓	↓	↓	↓
SH, RN, OP, **TO**	**S15**	↓	Open	↓	↓	↓	↓	↓	↓	↓
RN, **RO**, TO	**S16**	Shutdown	↓	↓	↓	↓	↓	↓	↓	↓
RO, TO	**S17**	Heat remove	↓	↓	292 inc	327 inc	x dec	x x	20 dec	0–90 dec

the candidates. Although notations, especially the use of gate symbols, are similar to fault trees, it does not state potential failure modes, causes, and the relationship among them. Rather, it describes conditions that must hold for the system to satisfy the goal stated in the root node. Some are hardware-related goals, and they are excluded because our focus is primarily on software. This step results in identifying software-related goals, and the negation of such goals, therefore, becomes the candidates of the fault-tree root node. The process of developing a safety goal tree is not automated and requires domain-specific knowledge to determine the goals that are to be satisfied by each component.

For designing a nuclear power plant, the final safety goal of the reactor protection system is to protect the public from radiation exposure over a setpoint by a design-basis event, such as core damage. Figure 7 shows the top-level goal tree of SDS2 produced from a system safety analysis. In this paper, we consider, for example, one of the goals to prevent core damage, which could be caused by an SGLL accident.

From the required system behaviors in Fig. 3, we can draw the safety goal tree according to the subgoals in Gf. Figure 8 shows the resulting safety goal tree to prevent the top event, the core damage of a nuclear power plant. There are three subgoals to meet the SGLL trip when required. We can extract the potential top events by negating the subgoals. Even each node could be a top event, we have identified that subgoal 1 is not handled by the embedded software in the controller because the conditions

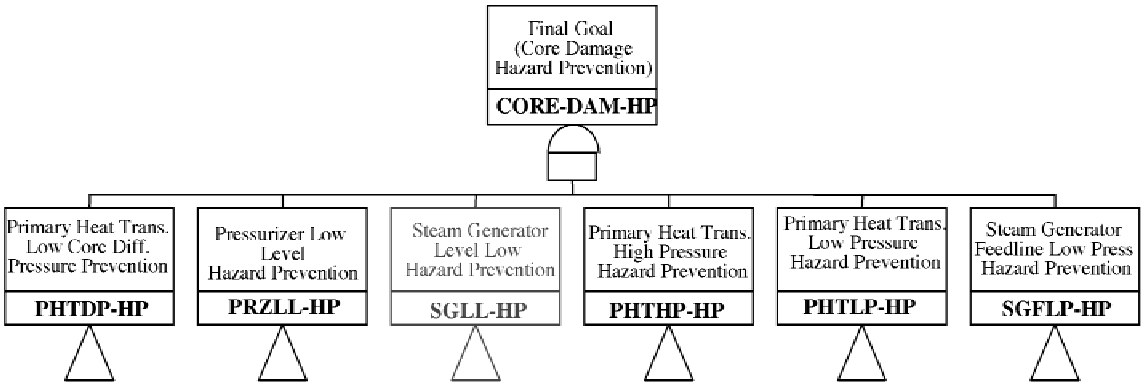

Fig. 7. The top-level goal tree for the SDS2 of Wolsong nuclear power plant (units 2, 3, and 4).

to meet subgoal 1 are determined by only the hardware components, such as the reactor, steam generator, and thermal load of the second loop. We can identify that in subgoals 2 and 3 of Fig. 8, the nodes TRIP-SIG-ON and TRIP-CON-OUT are determined by the software of the controller, so we can identify the two top events for software FTA as follows:

1. Top-event1: F-T-SIG-ON = ¬ TRIP-SIG-ON /∗ Trip control software fails to generate the trip on signal when SG-level is low ∗/

2. Top-event2: F-T-SIG-CO = ¬ TRIP-CON-OUT /∗ Trip control software fails to condition out the trip signal in low-power ∗/

III.B. Fault-Tree Construction (CRSA2)

Once top-level nodes of the fault tree are identified, causal ordering information included in the state trajectory as well as the causality information specified in the model fragments are used to investigate credible causes and the relationship among them. State trajectory is especially useful for this purpose because it contains causal, though qualitative, ordering information explaining the hybrid system behavior. The steps for constructing the fault tree are as follows:

1. Select the software top events in the state trajectory. For example, we can identify that top event 1, "Trip control software fails to generate the trip on signal when SG-level is low," means the failure of state S14 in the

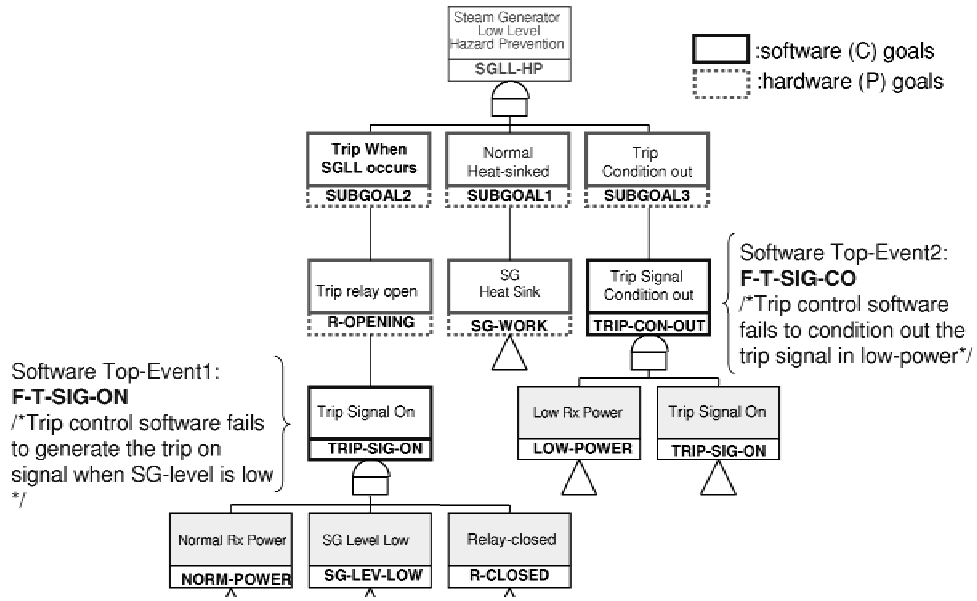

Fig. 8. CRSA1: safety goal tree.

state trajectory because the conditions of the software goal in Fig. 8 match the values of the same state variables in state S13.

2. Draw the essential fault tree at state level. We can draw the essential fault tree by following the behavior model fragments involved in making the trajectory. The essential fault tree (skeleton) represents the dynamic causalities between the plant and the control software. The set of all active model fragments in each state is actually much larger. Because, however, most of them represent the structural models, such as components, terminals, and junctions, and are active throughout the simulation, we follow only the behavior model fragments that change their activation status at the state in order to draw the essential fault tree that represents the dynamic causalities between the plant and the control software. Figure 9 shows the essential fault tree of top event 1. The safety analyst can draw the fault tree systematically from the behavior model fragments involved in the state trajectory. The bold names in the events of Figs. 9 and 10 represent the names of model fragments in Figs. 5 and 6. The main difference from the conventional fault tree is that there is a time sequence between parent and child events in the fault tree.

3. Decide the major causes of the top events. After we have found that the model fragment TN in state S13 is the immediate cause of top event 1, we can identify that the sequences of the model fragments, FL, NOT-FH, SH, Not-SN, SN, RN, . . . , RS, Not-RH are the major causes, because they are activated to make the system go into the state at that time. The shaded events in Fig. 9 represent the major causes of the top event and are drawn from the activated model fragments at the given states of the trajectory.

4. Search the immediate causes of the major events at condition level. We can find the immediate causes of the major event at condition level by negating the condition part of the model fragment. For example, the causes to fail the behavior of the trip-signal-on (TN) are the negated conditions in the model fragment TN in Fig. 6, $\neg((\text{Reactor ? r}) \wedge (\text{Trip-controller ?tc}) \wedge (\text{Signal (Signal-terminal ?tc)}) = \text{off} \wedge (\text{CAP ?r}) \geqq 10\ \%\text{FP} \wedge (\text{Level ?s}) \leqq \$\text{LSP}\$)$. Figure 10 shows the next refined fault tree for top event 1 using the causality information of the model fragment.

5. Repeat step 4 for all the major events, and connect the immediate causes to the next major event. Some human assistance is needed for the connection because there is no mechanical way to connect between the condition-level events and the next state-level events. For example, the event TN-FAIL in S13 of Fig. 10 has five condition-level events. An analyst can identify that there are three software-related events among them. He must determine how to connect one of them to the next state-level event. For the aforementioned example, it is relatively easy because there is only one condition-level event, L-NOT-LOW, which can be matched with the state-level event, FL-FAIL. The consequence part of the model fragment, FL, is that **Cm:** $(\text{Level ?sg}) = M\_(d(\text{Steam-out ?sg})/dt - d(\text{Feed-water-in ?sg})/dt)$. Because it means

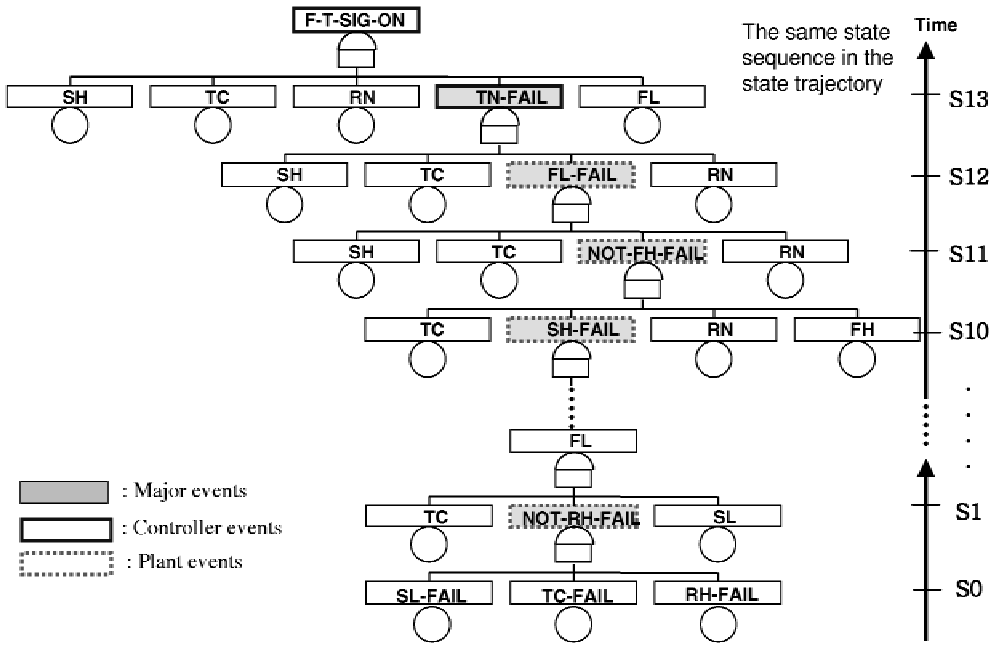

Fig. 9. Essential fault tree of software top event 1.

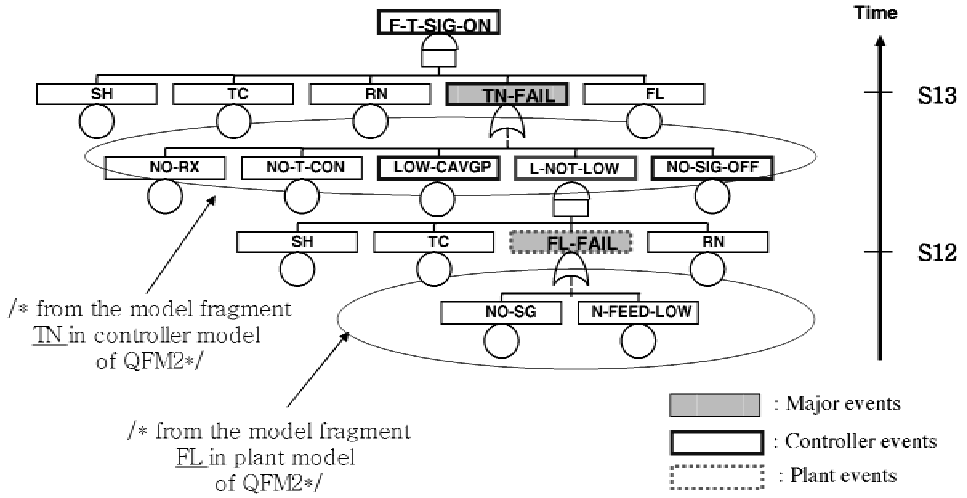

Fig. 10. Fault tree at condition level.

that the SG water level is the decreasing monotonic function of the difference between the change rates of Feedwater-in and Steam-out quantities, the analyst can identify the event, L-NOT-LOW, which can be selected to connect with the FL-FAIL event.

III.C. Fault-Tree Composition (CRSA3)

After the construction of fault trees on each top event, we have to combine these fault trees. We can combine these fault trees systematically by using the safety goal tree again. The negation of the safety goal tree can be the top-level fault tree. Then we can finalize the fault-tree composition by substituting the leaf node of the safety goal tree by the fault tree constructed in CRSA2. Figure 11 shows the top-level fault tree to combine the software fault trees of the SGLL trip controller.

Figure 12 shows the constructed fault tree for top event 1 on the essential level. Because the basic semantic of fault tree is the weakest precondition (wp) of Dijkstra,[23] we also apply Dijkstra's composition rules to compose

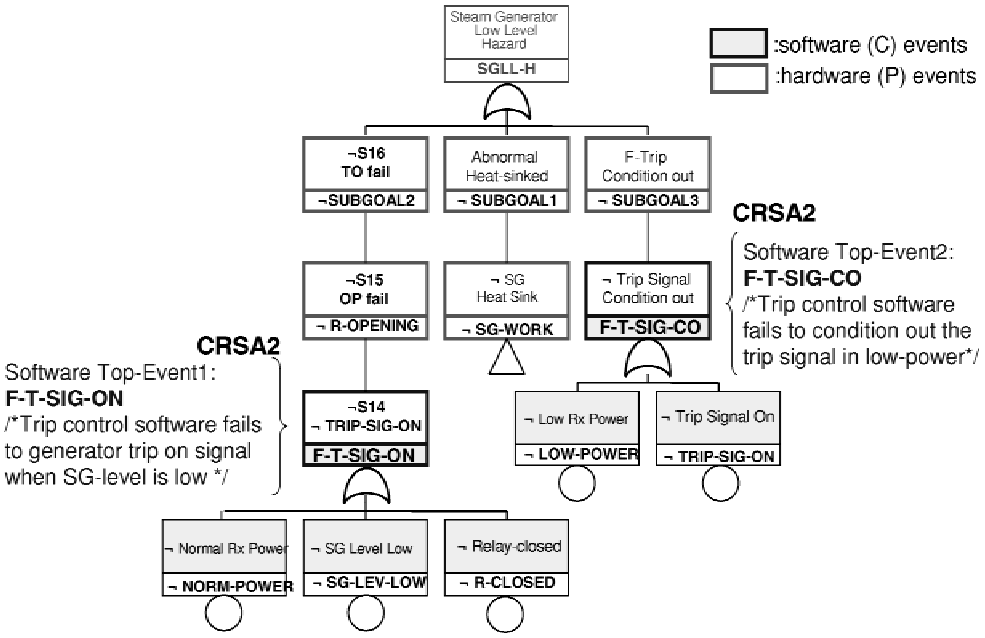

Fig. 11. Combined fault tree of the SGLL trip controller.

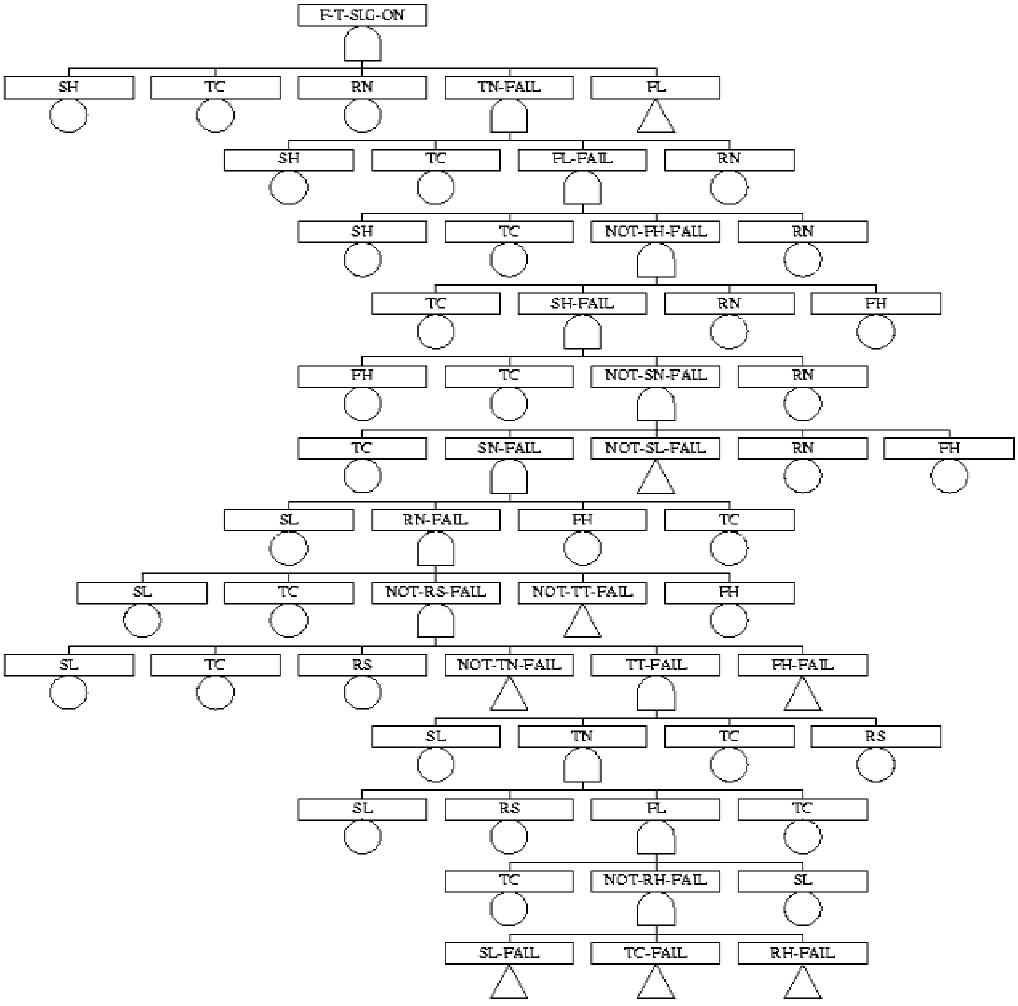

Fig. 12. Complete software fault tree on essential level for top event 1.

the fault trees. We can compose the fault trees step-by-step by applying the following rules of Dijkstra:

Rule 1. For any specification S and any post-conditions Q and R such that $Q \rightarrow R$ for all states
We also have $wp(S,Q) \rightarrow wp(S,R)$

Rule 2. For any specification S and any post-conditions Q and R we have $(wp(S,Q)$ and $wp(S,R)) = wp(S, Q$ and $R)$ for all states

Rule 3. For any specification S and any post-conditions Q and R we have $(wp(S,Q)$ or $wp(S,R)) \rightarrow wp(S, Q$ or $R)$ for all states

We have used the relationship between the semantic of the causality information by QFM and the semantic of FTA to integrate CRSA with QFM. That is the pre-postsemantic between events in the fault tree of CRSA and between states in QFM. Because CRSA is integrated with the formal specification method, the QFM, semantically, it can analyze the safety of software requirements systematically. CRSA is also compositional because it can construct fault trees on any level of abstraction according to the QFM specification.

III.D. Discussion

The system and hardware FTA methods have used the structural information. In the static FTA, analysts usually try to use the structural information of the program structure to find the causes of the upper events in the fault tree. The drawback of the static approach is that it is hard to find the causal relations between the physical hazard of the system and the hazardous behavior of the software. While there are only logical relations between nodes in the static FTA approach, CRSA has not only the logical but also the dynamic relations with time ordering

between the parent and the children nodes of the fault tree.

In the requirements phase of a software life cycle, the FTA methods usually use some models of the software, for example, the entity-relationship (E-R) diagram[10] and the statechart model.[13] A model-based FTA does not need to depend on the subjective knowledge of analysts. CRSA is also one of the model-based FTA methods. However, the model used in CRSA is different from others because it is the dynamic QFM model for hybrid system behavior. Because the QFM model has both the structural and the behavioral information of the continuous and discrete time, CRSA can analyze the hazard caused by dynamic causes, such as time-ordering errors in the software requirements for the hybrid system.

Software failures cannot be treated as random events, and probabilities for software failures cannot be derived using historical data.[2] We have instead developed the CRSA process to analyze the software failures in the requirements phase from the inconsistency between the logic of software and the physical hazard of system. CRSA can be used to guide software testing in the requirements phase.

IV. CONCLUSIONS

In this paper, we have presented a technique to systematically generate fault trees from qualitative specification available during early phases of requirements engineering. Our technique is well-suited to the development of hybrid real-time systems in that the behavioral models of the controller (software) as well as the plant (environment) behavior are included in the analysis.[5] Such comprehensive modeling and analysis is essential to safety engineers because safety property must be defined and verified within the overall system's context. Isolated analysis of software requirements alone would be inadequate.

Our approach extends the QFM framework we had previously proposed. While previous research demonstrated that qualitative physics formalism, developed in artificial intelligence research, was useful in modeling discrete and continuous behavior of hybrid systems, the technique described in this paper further extended the QFM process to include specific guidelines and procedures to conducting a safety analysis, systematic generation of fault trees in particular.

Conventional fault trees are also constructed based on systematic analysis on possible system states. However, it is difficult to extract the possible system states at the requirements analysis phase of the software development life cycle. It is also difficult to find the causal conditions of the state changes. Even though the state trajectory does not represent the complete set of procedures on all the possible system states, CRSA is a process that supports the more systematic construction of fault trees by understanding the system states and the conditions of the state changes as early as possible.

We have applied the proposed technique to partially specify and validate safety requirements of Wolsong SDS2, which is an emergency SDS for a nuclear power plant. Because CRSA was a fault-tree method based on the dynamic QFM models of a system, we could reduce the dependency on the subjective knowledge when constructing the fault trees by utilizing the causality information from the models.

The composed software fault trees will finally be integrated with the hardware and human fault trees, according to the information of the means-ends hierarchy.[24] The means-ends hierarchy can be used to provide the systematic framework of requirements engineering for a hybrid system. It will provide an evolutional framework from system to software, and it controls the complexity problems in interdisciplinary engineering by hierarchical and causal thinking. We are developing an expanded framework for a human-centered safety analysis framework by using the concepts from cognitive system engineering.[25]

ACKNOWLEDGMENTS

The work described here was supported in part by the Ministry of Science and Technology under the KNICS project and in part by AITrc of the Korea Advanced Institute of Science and Technology.

REFERENCES

1. W. E. VESELEY, *Fault Tree Handbook*, U.S. Nuclear Regulatory Commission (1981).

2. N. G. LEVESON and J. L. STOLZY, "Safety Analysis of Ada Programs Using Fault Trees," *IEEE Trans. Reliab.*, **5**, 479 (Dec. 1983).

3. N. G. LEVESON and P. R. HARVEY, "Analyzing Software Safety," *IEEE Trans. Software Eng.*, **5**, 569 (Sep. 1983).

4. J. S. LEE and S. D. CHA, "Qualitative Formal Method for Requirements Specification and Validation of Hybrid Real-Time Safety Systems," *IEE Proc. Software J.*, **14**, 1 (Feb. 2000).

5. J. S. OSTROFF, "Formal Methods for the Specification and Design of Real-Time Safety Critical Systems," *J. Sys. Software*, 33 (Apr. 1992).

6. "Software Requirements Specification for Shutdown System 2 PDC," Design Document No. 86-68350-SRS-001, Rev. 0, Wolsong NPP 2/3/4 (June 1993).

7. K. M. HANSEN, A. P. RAVN, and V. STAVRIDOU, "From Safety Analysis to Formal Specification," ProCos II Technical Report, Technical University of Denmark (1994).

8. D. HAREL, "Statecharts: A Visual Formalism for Complex Systems," *Sci. Comput. Program.*, **3**, 231 (June 1987).

9. N. G. LEVESON, *SAFEWARE: System Safety and Computers*, Addison-Wesley Publishing Company, New York (1995).

10. S. LIU and J. A. McDERMID, "Model-Oriented Approach to Safety Analysis Using Fault Trees and a Support System," *J. Syst. Software*, **35**, 151 (1996).

11. S. S. CHA, "A Safety-Critical Software Design and Verification Technique," PhD Dissertation, University of California Irvine (1991).

12. P. FENELON and J. A. McDERMID, "An Integrated Tool Set for Software Safety Analysis," *J. Syst. Software*, **21**, 279 (1993).

13. S. SUBRAMANIAN, R. V. VISHNUVAJJALA, R. MOJDEBAKHSH, W. T. TSAI, and L. A. ELLIOTT, "Framework for Designing Safe Software Systems," *Proc. COMPSAC'95*, Dallas, Texas, August 1995, p. 409.

14. N. G. LEVESON and J. L. STOLZY, "Safety Analysis Using Petri Nets," *IEEE Trans. Software Eng.*, **13**, 386 (1987).

15. S. S. CHA, N. G. LEVESON, and T. J. SHIMEALL, "Safety Verification in Murphy Using Fault Tree Analysis," *Proc. ICSE-10*, Singapore, p. 377, IEEE Computer Society Press (1988).

16. J. GORSKI and A. WARDZINSKI, "Formalizing Fault Trees," *Proc. Safety-Critical Systems Symp.*, United Kingdom, pp. 7 and 311 (Feb. 1995).

17. S. J. CLARKE and J. A. McDERMID, "Software Fault Trees and Weakest Preconditions: A Comparison and Analysis," *IEE Proc. Software J.*, 225 (1993).

18. J. S. LEE and S. D. CHA, "Behavior Verification of Hybrid Real-Time Requirements by Qualitative Formalism," *Proc. 4th RTCSA*, Taipei, Taiwan, October 1997, p. 127, IEEE Computer Society Press (Oct. 1997).

19. Y. IWASAKI, M. VESCOVI, R. FIKES, and B. A. CHANDRASEKARAN, "Causal Functional Representation Language with Behavior-Based Semantics," *Appl. Artificial Intell. J.*, **9**, 5 (1995).

20. B. FALKENHAINER, A. FARQUHAR, D. BOBROW, R. FIKES, K. FORBUS, T. GRUBER, Y. IWASAKI, and B. KUIPERS, "CML: A Compositional Modeling Language," KSL-94-16, Stanford University (1994).

21. Y. IWASAKI and C. M. LOW, "Model Generation and Simulation of Device Behavior with Continuous and Discrete Change," KSL-91-69, Stanford University (Nov. 1991).

22. D. G. BOBROW, *Qualitative Reasoning About Physical Systems*, MIT Press, Cambridge, Massachusetts (1985).

23. E. W. DIJKSTRA, *A Discipline of Programming*, Prentice Hall, Englewood Cliffs, New Jersey (1976).

24. J. RASMUSSEN, A. M. PEJTERSEN, and L. P. GOODSTEIN, *Cognitive Systems Engineering*, John Wiley & Sons, New York (1994).

25. J. S. LEE, K. H. CHA, and K. C. KWON, "Safety Assessment Framework for Software Based I&C Systems," *Trans. Am. Nucl. Soc.*, **88**, 39 (2003).

Jang-Soo Lee [BS, 1983, electronics, Kyungpook National University, Korea; MS, 1986, and PhD, 2002, computer science, Korea Advanced Institute of Science and Technology (KAIST), Korea] is a principal research scientist at Korea Atomic Energy Research Institute (KAERI) responsible for nuclear instrumentation and control. His research interests are safety analysis of software-based systems, software verification and validation, embedded system testing, formal methods, and digital instrumentation and control architecture.

Kee-Choon Kwon (BS, electronics, Kyungpook National University, Korea; MS, 1989, and PhD, 1999, computer science, KAIST) is a principal research scientist at KAERI responsible for nuclear instrumentation and control. His research interests are safety software verification and validation, real-time simulation, and nuclear instrumentation and control.

Sung-Deok Cha (BS, 1983, MS, 1986, and PhD, 1991, information and computer science, University of California, Irvine) is a professor at KAIST. His research interests are software engineering, software safety, computer security, formal specification and verification, and software fault tolerance.