A Formal Software Requirements Specification Method for Digital Plants Protection Systems

Junbeom Yoo
KAIST

Sungdeok Cha
KAIST

CS/TR-2003-191

June 1, 2003

K A I S T
Department of Computer Science

A Formal Software Requirements Specification Method for Digital Plants Protection Systems

Junbeom Yoo
KAIST

Sungdeok Cha
KAIST

Abstract

This article describes NuSCR, a formal software requirements specification method for digital plant protection system in nuclear power plants. NuSCR improves the readability and specifiability by supplying different notations on the basis of the typical operation categories. The characteristics of the software process controller in nuclear power plants, s.t. periodic sequential processing and classifiable operations, makes this possible. We introduce the syntax and semantics of NuSCR in the paper. An ATWS mitigation system in Korean nuclear power plant is used as a case study to illustrate the usefulness of NuSCR.

1 Introduction

Software safety is an important property for safety critical systems, especially those in aerospace, satellite and nuclear power plants, whose failure could result in danger to human life, property or environment. It is recently becoming more important due to the increase in the complexity and size of safety critical systems [9]. Formal software requirements specification is known as a means to increase the safety of such safety critical systems in the early phase of software development process. It guides the developer to specify all requirements explicitly without any assumptions or omissions. Also many recognized formal verification methods, such as model checking [2] [3] and mechanized theorem proving [21], can be applied to the formal software requirements specification.

In the area of nuclear power plant systems, the formal specification of software requirements becomes more important with the replacement of existing analog systems by digital systems composed of software process controllers [12]. Nowadays, software requirements and design specifications which are suitable for the characteristics of nuclear power plants system, are becoming new research issues by many researcher as the Practical Formal Specification(PFS) project in aerospace applications[11].

Typical characteristics of digital protection controllers in nuclear power plant systems are as follows. First, numerous inputs are calculated by the software process controllers. To maintain the system to be safe, all the status of reactors and peripherals, i.e. turbines, steam generators, and other subsystems, should keep being observed. Second, the software operates sequentially, s.t. receives software inputs, calculates with them, and then emits software outputs. It repeats the sequential operation periodically at every predefined time interval. Last of all, all the possible operations of the software process controller can be classified into three categories. They are function-based, state-based, and timing-based operations. Function-based operations are the functions that gets inputs, calculates with inputs only, and then emits an output. State-based operations are the operations that require the history information additionally. Timing-based operations are the ones which require timing constraints in addition to the history information.

NuSCR is a formal software requirements specification extended from SCR(Software Cost Reduction) [5] to easily specify the functional requirements of safety critical software, especially those of nuclear power plants. It is based on *Parnas' Four-Variable Model* [13] and uses FOD(Function Overview Diagram) for the overview of data flows in the same way as [22]. [22] is a variant of SCR, which was proposed by AECL(Atomic Energy of Canada Limited) and was used for the formal software requirements specification for SDS2(ShutDown System 2) in Wolsong nuclear power plant in Korea. NuSCR improves the readability and specifying ability by supplying different notations on the basis of their typical operation categories. The characteristics of software process controllers in nuclear power plants, s.t. periodic sequential processing and classifiable operations, makes this possible.

In the approach of AECL, the state-based operations such as *trip set point hysterisis* are specified by functions, although they have originally state-based features. It is because that the basic specifying concept of AECL approach is to specify all aspects by functions. Timing-based operations such as *delay timer* are also specified by special timer functions, which are too hard to define and understand. In NuSCR, we adopts FSM(Finite State Machine) for specifying state-based parts, and a kind of TTS(Timed Transition System)[18] for timing-related parts in software requirements.

NuSCR formal software requirements specifications can be verified by theorem prover PVS [15] with our approach [7] developed for SCR. Using PVS, we can verify the structural properties such as input/output completeness, consistency, and circular dependencies in NuSCR specification. NuSCR specifications can also be verified by model checker such as the SMV [10], based on the formal semantics of NuSCR presented in this paper. We are developing an automatic translator that translates NuSCR specification into SMV inputs.

The remainder of the paper is organized as follows: Section 2 reviews SCR and the variant proposed by AECL. Section 3 introduces the specification constructs in NuSCR. In Section 4, we represent the formal semantics of NuSCR software requirement specifications. We then briefly introduce NuSCR requirements specification for AMS(ATWS Mitigation System) as a case study, and describe the software development environment in progress in Section 5. Conclusion and future work direction are in Section 6.

2 Formal Requirements Specification Approaches

Some formal requirements specification methods such as Z [16], VDM [6], and Larch [4] focus on specifying the behavior of sequential systems. These approaches use rich mathematical structures like sets, relations, and functions to describe states and use pre-conditions and post-conditions for state transitions. However, these approaches are too expressive to specify nuclear power plants software concisely.

SCR [5] was introduced more than twenty years ago to specify the software requirements of real-time embedded systems. Recently it has been extended to incorporate both functional and non-functional(e.g. timing and accuracy) requirements [14], [1]. As it was designed to be used by engineers, the SCR methods has been successfully applied to a variety of practical systems, such as the A-7 Operational Flight Program [17], submarine communication system, and safety-critical component of Darlington nuclear power plant in Canada [1].

The approach [1] applied to the Darlington nuclear power plant by AECL is the first attempt as the formal software requirements specification for nuclear power plants system and it was also applied to SDS2(ShutDown System 2) in Wolsong nuclear power plant in Korea [23]. The approach is based on SCR and has some extensions from SCR. At first, to specify the software more concisely, it combined the three tables of SCR, the mode transition table, event table, and condition table, into a table called SDT(Structured Decision Table). It uses FOD(Function Overview Diagram) which is similar to DFD(Data Flow Diagram) for the overview of the system. Finally, it provides sophisticated functions for describing precision and tolerance to describe timing constraints.

The characteristics of AECL approach is as follows: (i) SDTs and FODs are familiar notations for domain engineers and developers. (ii) However, SDTs are too complicated. There usually are too many columns and rows to understand. The complexity of SDTs comes from the basic idea of the AECL approach. Although they originally have state-based features, the function form of them makes unnecessary complication arise. (iii) Managements of time-related feature such as *timers* are too complicated to define and understand. They use the special timing functions for specifying time-related requirements. However the definition of them is too hard to be known by common domain engineers by intuition.

3 NuSCR Software Requirements Specification Constructs

NuSCR basically uses four constructs, *monitored variable*, *input variable*, *output variable*, and *controlled variable* according to *Parnas' Four-Variable Model*[10]. In addition, to specify the relations of *Parnas' Four-Variable Model* in practical and domain dependent manners, we introduce three other basic constructs, *function variable*, *history variable*, and *timed history variable.* These three constructs can be defined as SDT, FSM, and TTS respectively. The relationship of all constructs is represented by FOD.

Naming Convention NuSCR uses the prefix naming convention as follows to distinguish each construct efficiently. Two prefixes, "*g_*" and "*k_*", are introduced for the convenience of specification:

- *m_* : monitored variable
- *i_* : input variable
- *f_* : function variable
- *h_* : history variable
- *th_* : timed history variable
- *g_* : set of function variable, history variable, or timed history variable
- *k_* : predefined constant
- *o_* : output variable
- *c_* : controlled variable

System Entities System entities constructing NuSCR software requirements specification are defined as follows:

- V_I : a set of system input variables
- V_F : a set of function variables
- V_H : a set of history variables
- V_{TH} : a set of timed history variables
- V_O : a set of system output variables
- V_{SE} :

– a set of system entities

– $V_I \cup V_F \cup V_H \cup V_{TH} \cup V_O$

- D_{SE} : a set of all possible domain for every r in V_{SE}
- σ : a valuation function that maps V_{SE} into D_{SE}

  – $\sigma[d/v]$ means that (let $v \in V_{SE}$, $V_v = V_{SE}$ - $\{v\}$, $d \in D_{SE}$) $\sigma^{'}[v] = d$ and $\sigma^{'}[V_v] = \sigma[V_v]$

  – $\sigma(f(v)) = f(\sigma[v]) = f(v)(\sigma)$

Condition Statements Condition statements are the predicates on the value of all entities in *SE*. The condition statements in NuSCR are defined as BNF form as follows:

Let $r \in V_{SE}$, $v_r \in D_{SE}$, $a, b \in N$, and $\otimes \in \{=, \neq, \leq, <, \geq, >\}$,
$simple\_condition := r \otimes v_r \mid r \otimes r \mid TRUE \mid FALSE$
$complex\_condition := simple\_condition \wedge simple\_condition$
$\mid simple\_condition \vee simple\_condition \mid \neg simple\_condition \mid simple\_condition$
$timed\_condition := [a, b] complex\_condition$

As the above definition, *timed_condition* is a *complex_condition* appended by the timing constraints $[a, b]$ which means a duration of time a and b. *timed_condition* is used in defining timed history variables, and *complex_condition* is used in defining both function variables and history variables.

Assignment Statements Assignment statements mean the valuation of entities in *SE*. The assignment statements in NuSCR are defined as BNF form as follows:

Let $r \in V_{SE}$, $v_r \in D_{SE}$, $a, b \in N$, and $\oplus \in \{+, -, *, \div\}$
$assignment := (r := v_r) \mid (r := r) \mid (r := r \oplus r) \mid (r := r \oplus v_r)$

Function Variable Function variables are used for specifying the mathematical functional behavior of a system. They are defined as SDTs. SDT is a kind of Condition/Action table, which represents the actions(assignment statements) performed if their guiding conditions(condition statements) are satisfied. Tabular notations such as SDTs have the merit of being familiar to engineers and developers. Conditions in SDT are the *complex_conditions* with the inputs of the function variable. Actions are the *assignment* to the function variable itself.

Conditions		
th_CH1_TICP_TRIP_Status = k_CH1_TICP_TRIP	T	F
th_CH1_TICP_TRIP_Status = k_CH1_TICP_NORMAL	F	T
Actions		
f_OC1_TICP_BL_I := f_OC1_TICP_I * 100	X	
f_OC1_TICP_BL_I := f_OC1_TICP_I * 10		X

Figure 1: Structured decision table for *f_OC1_TICP_BL_I*

⟨Fig. 1⟩ is an SDT defining function variable *f_OC1_TICP_BL_I*. It is excerpted from NuSCR software requirements specification for AMS[20]. Input entities for this variable are *th_CH1_TICP_TRIP_STATUS* and *f_OC1_TICP_I*. The detailed interpretation of such definition is as follows:

> If *th_CH1_TICP_TRIP_STATUS* is same as *k_CH1_TICP_TRIP* and not same as *k_CH1_TICP_NORMAL*, then the new value for *f_OC1_TICP_BP_I* is *f_OC1_TICP_I* multiplied by 100. Else if *th_CH1_TICP_TRIP_STATUS* is same as *k_CH1_TICP_NORMAL* and not same as *k_CH1_TICP_TRIP*, then the new value for *f_OC1_TICP_BP_I* is *f_OC1_TICP_I* multiplied by 10.

History Variable History variables are used for specifying the state-based behavior of a system. They are defined as FSMs. FSM consists of finite number of states, transitions between states, and labels on each transition. Labels are the Conditions/Actions statements which are same as that of SDTs. Conditions in FSM's transition labels are the *complex_conditions* with the inputs of the history variable. Actions are the *assignment* to the history variable itself. If the transition condition is satisfied in the current state, then the action is performed and the state transition occurs.

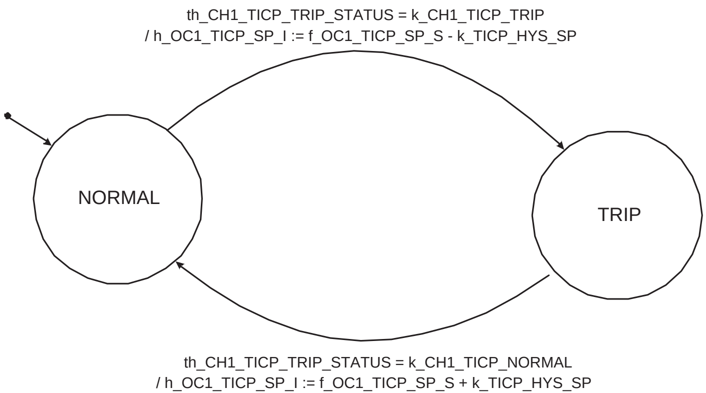


Figure 2: Finite state machine for *h_OC1_TICP_SP_I*

⟨Fig. 2⟩ is a FSM defining history variable *h_OC1_TICP_SP_I*. Input entity for this variable is *th_CH1_TICP_TRIP_STATUS* and the initial state is *NORMAL*. The detailed interpretation of such definition is as follows:

> In state *NORMAL*, if *th_CH1_TICP_TRIP_STATUS* is same as *k_CH1_TICP_TRIP*, then the new value for *h_OC1_TICP_SP_I* is *h_OC1_TICP_SP_S* minus *k_TICP_HYS_SP* and transition to state *TRIP* occurs. Also in state *TRIP*, if *th_CH1_TICP_TRIP_STATUS* is same as *k_CH1_TICP_NORMAL*, then the new value for *h_OC1_TICP_SP_I* is *h_OC1_TICP_SP_S* plus *k_TICP_HYS_SP* and transition to state *NORMAL* occurs.

Timed History Variable Timed history variables are used for specifying the time-related behavior of system. They are defined as a kind of TTS [18]. TTS is a FSM extended with the timing constrains $[a, b]$ in transition conditions. $[a, b]$ means the time duration between time a and b.

⟨Fig. 3⟩ is a TTS defining timed history variable *th_CH1_TICP_TRIP_STATUS*. Input entities for this variable are *f_OC1_TICP_I* and *h_OC1_TICP_SP_S* and the initial state is *NORMAL*. The detailed interpretation of such definition is as follows:

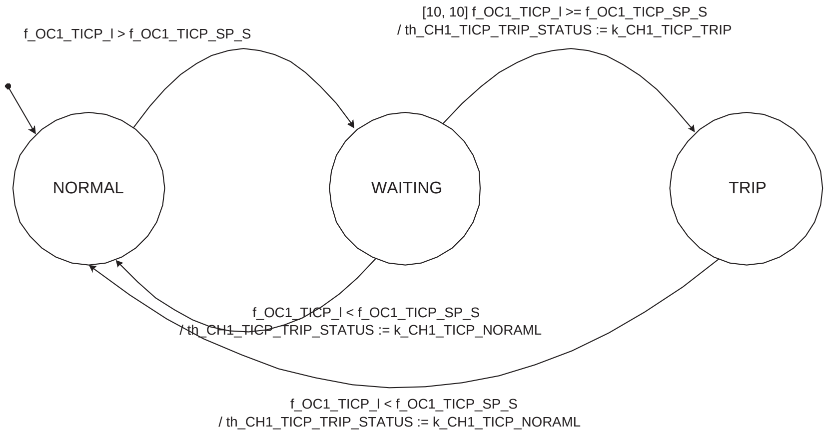


Figure 3: Timed transition system for *th_CH1_TICP_TRIP_STATUS*

In state *NORMAL*, if *f_OC1_TICP_I* is greater than *h_OC1_TICP_SP_S*, then immediate transition to state *WAITING* occurs and the output value is same as the previous one. If the condition is satisfied for 10 seconds from the point entering state *WAITING*, then the new value for *th_CH1_TICP_TRIP_STATUS* is *k_CH1_TICP_TRIP* and transition to state *TRIP* occurs. Else if the condition is not satisfied for 10 seconds from the point entering state *WAITING*, then the new value for *th_CH1_TICP_TRIP_STATUS* is *k_CH1_TICP_NORMAL* and transition to state *NORMAL* occurs. In state *TRIP*, if *f_OC1_TICP_I* is less than *h_OC1_TICP_SP_S*, then immediate transition to state *NORMAL* occurs and the new value for *th_CH1_TICP_TRIP_STATUS* is *k_CH1_TICP_NORMAL*.

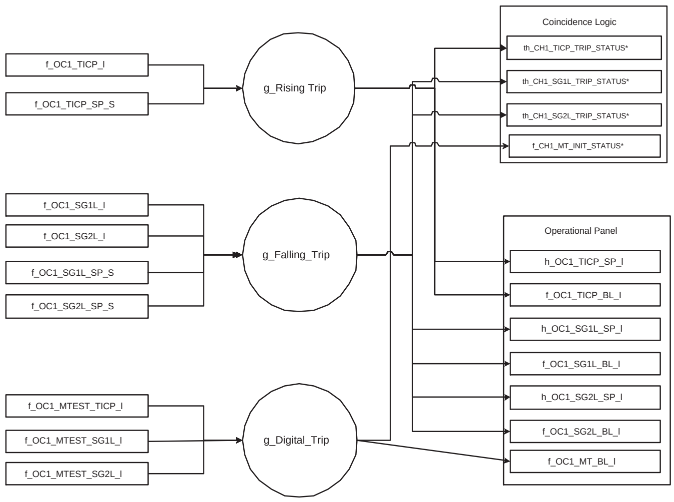
Function Overview Diagram FOD(Function Overview Diagram) is a kind of DFD, which describes the relationship between constructs in V_{SE} in NuSCR software requirements specification. Each construct in V_{SE} is represented by specific nodes, and the relationship between them is represented by unidirectional arrows. FOD is composed hierarchically and in this case the group nodes are used. The name of each node is followed by the prefix naming convention described above.
⟨Fig. 4⟩ is a FOD for group node *g_Bistable_Logic*. It is composed of three group nodes *g_Rising_Trip*, *g_Falling_Trip*, and *g_Digital_Trip*. As shown in (a), the group node *g_Rising_Trip* has *f_OC1_TICP_I* and *f_OC1_TICP_BP_S* as input entities. The output of *g_Rising_Trip* are *f_OC1_TICP_BL_I*, *th_CH1_TICP_TRIP_STATUS*, and *h_OC1_TICP_SP_I*. The refined FOD is represented in (b). It is composed of three nodes which have corresponding outputs repsectively. The relationship between these three nodes are represented by arrows.

4 Semantics for NuSCR

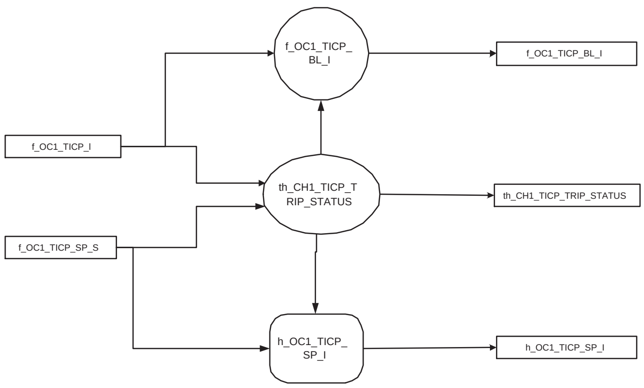
The behavior of software system specified by NuSCR can be defined based on the behavior of FOD.

Function Overview Diagram FOD in NuSCR is defined as a tuple:

$FOD = \langle\ N, T, I, O\ \rangle$



(a) g_Bistable_Logic



(b) g_Rising_Trip

Figure 4: Function overview diagram for *g_Rising_Trip*

- N
  - a set of all function nodes in FOD
  - all nodes in V_F and V_H and V_{TH} are defined as functions
- T
  - a set of transition (n_1, n_2) between all nodes n_1, n_2 in N
  - $\forall\ t = (n_1, n_2) \in T$, n_1 has a precedence on n_2
- I
  - a set of input values into FOD
  - It is mapped into input variables V_I of FOD
- O
  - a set of output values from FOD
  - It is mapped into output variables V_O of FOD

From the definition of FOD above, FOD can be defined as a function f_{FOD} from input values I to output values O. Also as all nodes in N have partial ordering according to the transition in T and all nodes are defined as functions, f_{FOD} can be represented as a function composition of all nodes in N according to the partial orders on their precedence.

$f_{FOD}(\sigma[I/V_I]) = \sigma[O/V_O]$
$f_{FOD} =$ (mathematical function composition by partial orders on their precedence)

Function Variable Node Function variable in NuSCR is represented by a function variable node in FOD. It is defined by SDT. Let I_{FV} be the set of input values from other nodes in FOD into the function variable node itself. Let O_{FV} be the set of output values from this node. They can be mapped into the set of variables, V_{FI} and V_{FO} respectively. Then *complex_conditions* in SDT are the predicate on V_{FI}, and actions are the *assignments* on V_{FO} which is the function variable itself.

SDT is defined as a set of a pair (p, a), where $p \in P$ and $a \in A$. P is a set of boolean predicates on V_{FI}, which is the conjunction of *complex_conditions* in condition statements and corresponding boolean conditions. A is a set of *assignments* to V_{FO} which is just the function variable itself.

SDT : a set of pair (p, a)

- $p \in P$ and $a \in A$
- $\forall\ (p, a)$ in SDT, $p(\sigma) = T$ then $a(\sigma) = \sigma[O_{FV}/V_{FO}] = \sigma'$

For example, SDT in ⟨Fig. 5⟩ can be defined as follows:

$SDT = \{(Cond_1 = T \wedge Cond_2 = F,\ Assign_1),\ (Cond_2 = T \wedge Cond_3 = F,\ Assign_2),\ (Cond_1 = T \wedge Cond_3 = T,\ Assign_3)\}$
From the definition of SDT above, a function variable node can be defined as a function f_{FV} with input values I_{FV} to output values O_{FV} as follows.

Conditions			
$Cond_1$	T	-	T
$Cond_2$	F	T	-
$Cond_3$	-	F	T
Actions			
$Assign_1$	X		
$Assign_2$		X	
$Assign_3$			X

Figure 5: Structured decision table for a function variable

$f_{FV}(I_{FV}) = \sigma[I_{FV}/V_{FI}] = O_{FV}$

History Variable Node History variable in NuSCR is represented by a history variable node in FOD. It is defined by FSM which is composed of states, transitions between states, and labels on transitions. Let I_{HV} be the set of input values from other nodes in FOD into the history variable node. Let O_{HV} be the set of output values from this node. They can be mapped into the set of variables, V_{HI} and V_{HO} respectively. Then *complex_conditions* in FSM are the predicate on V_{HI} and actions are the *assignments* on V_{HV} which is the history variable itself. FSM can be defined as a relation described below:

$FSM = \langle\ S_H,\ s_0,\ C,\ A,\ R\ \rangle$

- S_H : a set of states in history variable node
- s_0 : initial state in S_H
- C : a set of *complex_conditions*
- A : a set of *assignments*
- R :
  - $S_H \times C \times A \times S_H$ is a transition relation
  - $\forall\ r\ (s, c, a, s^{'})$ in R,
    s.t. $current\_state = s$ and $c(\sigma) = T$ then $a(\sigma) = \sigma[O_{HV}/V_{HO}] = \sigma^{'}$

current_state in the definition above means the variable in CS_H, which indicates the current state of the history node. It will be used in the definition of the overall NuSCR system. History variable in ⟨Fig. 6⟩ can be defined as a relation as follows:

$$
\begin{aligned}
&FSM = \langle S_H, s_0, C, A, R\rangle\\
&\quad S_H = \{S_1, S_2, S_3\}\\
&\quad s_0 = S_1\\
&\quad C = \{Cond_1, Cond_2, Cond_3, Cond_4\}\\
&\quad A = \{Assign_1, Assign_2, Assign_3, Assign_4\}\\
&\quad R = \{(S_1, Cond_1, Assign_1, S_2), (S_2, Cond_3, Assign_3, S_1), (S_2, Cond_2, Assign_2, S_3),\\
&\qquad (S_3, Cond_4, Assign_4, S_1)\}
\end{aligned}
$$

From the definition of FSM above, a history variable node can be defined as a function f_{HV} from input values I_{HV} to output values O_{HV} as follows.

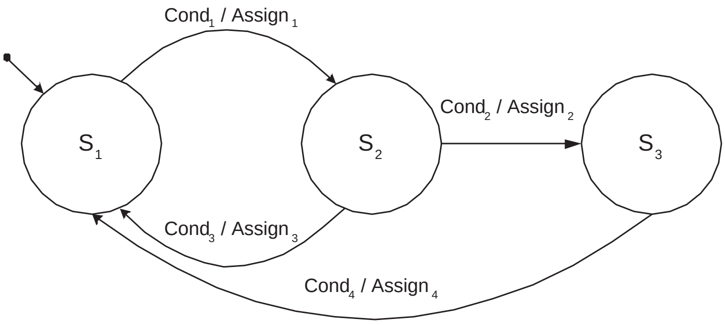


Figure 6: Finite state machine for history variable node

$f_{HV}(I_{HV}) = \sigma[I_{HV}/V_{HI}] = O_{HV}$

Timed History Variable Node Timed history variable in NuSCR is represented by a timed history variable node in FOD. It is defined by *TTS* which is a *FSM* extended with timing constrains $[a, b]$ in transition labels. a and b means the minimum and maximum delay in the transition respectively. Let I_{THV} be the set of input values from other nodes in FOD into the timed history variable node. Let O_{THV} be the set of output values from this node. They can be mapped into the set of variables, V_{THI} and V_{THO} respectively. Then *timed_conditions* are the predicate on V_{THI} and timing constrains $[a, b]$, and actions are the *assignment* on V_{THV} which is the history variable itself. TTS can be defined as a relation described below:

$TTS = \langle\ S_{TH},\ s_0,\ C,\ A,\ R\ \rangle$

- S_{TH} : a set of states in timed history variable node $\times$ lc, where lc is a local clock in LC
- s_0 : initial state in S_{TH}
- C : a set of *timed_conditions*
- A : a set of *assignments*
- R :
  - $S_{TH} \times C \times A \times S_{TH}$ is a transition relation
  - $\exists\ r\ (s, c, a, s^{'})$ in R,
    s.t. *current_state* $= s$ and $c(\sigma) = T$ then $a(\sigma) = \sigma[O_{THV}/V_{THO}] = \sigma^{'}$

The behavior of transition relations in TTS is a little different from that of FSM because of the timing constraints. For example, the transition from state S_1 to S_2 in ⟨Fig. 7⟩ has the transition labeled with "$[a, b]Cond_1/Assign_1$". The minimum delay a means that when the control of timed history node has resided at the location S_1 for at least a time units during which the guard $Cond_1$ has been continuously true, then the transition from S_1 to S_2 may occur. The maximum delay b means that whenever the state of history variable has resided at S_1 for b time units during which the guard $Cond_1$ has been continuously true, then the transition from S_1 to S_2 has to occur. The behavior of this transition can be described as follows. *"lc := lc + 1"* means the local time progress and *"lc := 0"* means the local clock initialization. *current_state* is a variable in CS_{TH}, which indicates the current state and the current local time.
For example, timed history variable in ⟨Fig. 7⟩ can be defined as a relation as follows:

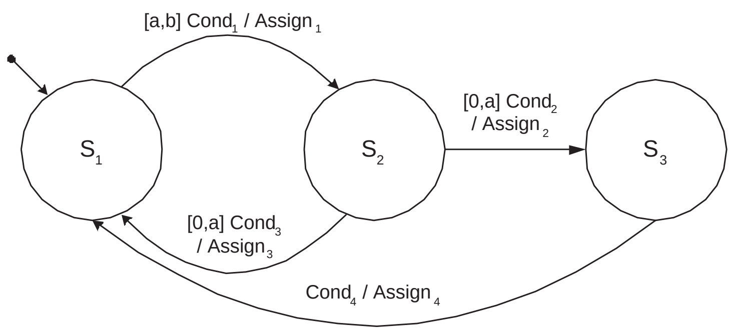


Figure 7: Timed transition system for a timed history variable node

$TTS = \langle S_{TH}, s_0, C, A, R \rangle$
$S_{TH} = \{(S_1, lc), (S_2, lc), (S_3, lc)\}$
$s_0 = (S_1, 0)$
$C = \{[a,b]Cond_1, [0,a]Cond_2, [0,a]Cond_3, Cond_4\}$
$A = \{Assign_1, Assign_2, Assign_3, Assign_4\}$
$R = \{((S_1, (a,b)), Cond_1, Assign_1, S_2), ((S_2, [0,a]), Cond_3, Assign_3, S_1), ((S_2, [0,a])Cond_2,$
$Assign_2, S_3), ((S_3, -), Cond_4, Assign_4, S_1)\}$

From the definition of TTS above, a timed history variable node can be defined as a function f_{THV} from input values I_{THV} to output values O_{THV} as follows.

$f_{THV}(I_{THV}) = \sigma[I_{THV}/V_{THI}] = O_{THV}$

Function f_{THV} generates an output(O_{THV}) whenever it gets inputs(I_{THV}) from other nodes in *FOD*. If no conditions are satisfied, then the value of O_{THV} in the previous scan cycle is preserved. However, although it gets no inputs I_{THV}, the transition condition can be satisfied as the local time proceeds. In nuclear power plants system, however, this situation can be avoided. It is because system scan cycle time d is always much more smaller than the time a or b in timing constraint $[a, b]$. (i.e. d is 50ms and delay time a is 5sec.) Of course, we need to adjust that a or b are the multiple of d.

NuSCR Software System NuSCR software system is defined as a tuple

$NSS = \langle\ S,\ S_0,\ R,\ d\ \rangle$ in which

- S
  - a set of system states
  - $\sigma[V_{SE} \times CS_H \times CS_{TH}]$
  - CS_H : a set of variables which indicate the current state of history nodes
  - CS_{TH} : a set of variables which indicate the current state and the current local time of time history variable nodes
- S_0 : initial state in S
- R : a set of transition relation $S \times I \longrightarrow S' \times O$

- d : system scan cycle time in which the system get the changed valuation function σ periodically

5 Case Study: AMS Example

In this section, we introduce NuSCR software requirements specification for AMS(ATWS Mitigation System) in Kori NPP Unit 1 in Korea [19], [20]. We also introduce our supporting tool for NuSCR specification.

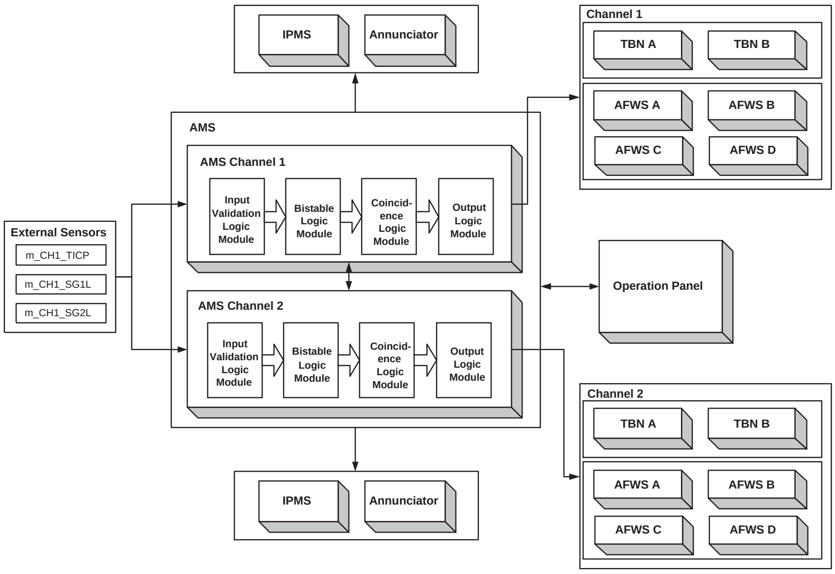


Figure 8: Overview of AMS Subsystem

AMS Description The AMS provides its protective action to mitigate the effects occurring followed by a failure of the reactor trip portion of the reactor trip system. It initiates a turbine trip and actuates AFWS(Auxiliary Feed Water System). There are two turbines and four AFWSs for each channel. The AMS also communicates some information with OP(Operational Panel), IPMS(In Plant Monitoring System), and Annunciators(i.e. alarms). The AMS consists of two identical channels containing *Input/Output Module*, *Bistable Logic Module*, and *Coincidence Logic Module*. The AMS and its related subsystems are described in ⟨Fig. 8⟩ and the description of its subsystems are as follows.

- *Input Validation Logic Module* : Input validation function converts the input parameters from raw input value to scaled value and validates the scaled input parameter.
- *Bistable Logic Module* : Bistable logic function determines if a trip condition exists based on measured parameters. Two types of bistable function are implemented, a fixed setpoint and a contact input. The fixed setpoint function compares a programmed constant with a digitized parameter to determine the trip state. The contact bistable function uses the state of the input to establish the trip state.
- *Coincidence Logic Module* : The coincidence logic function uses the bistable trip states generated internally from both channels to determine if a system initiation should occur. This function generates a turbine trip signal and a AFWS actuation signal whenever a coincidence of a low steam generator level trip or the manual trip has occurred.
- *Output Logic Module* : This logic shall obtain data from selected registers and send this information to the Annunciator, the In-Plant Monitoring System and the AMS OP.

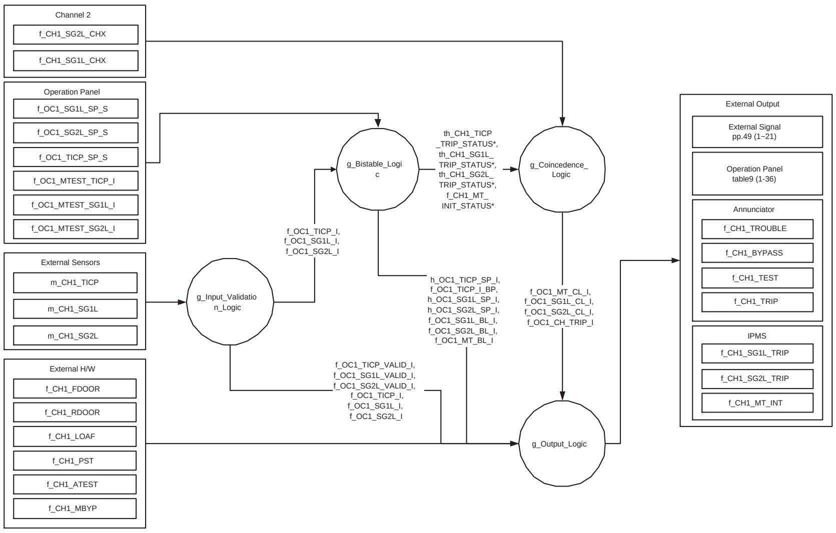


Figure 9: g_AMS_Overview in NuSCR specification

NuSCR Specification for AMS We produced NuSCR specification for a channel of AMS from [19] and [20]. ⟨Fig. 9⟩ depicts the *g_AMS_Overview* in NuSCR specifications, which is the root node of FOD. It shows the overall data flows of AMS systems. It has 17 inputs and 64 outputs. Inputs come from operation panel, external sensors, external hardware, and other channel. Outputs are for IPMS, annunicators, operation panel, turbines, and AFWSs. It is identical to ⟨Fig. 8⟩. The AMS consists of 4 subgroup nodes, *g_Input_validation_Logic*, *g_Bistable_Logic*, *g_Coincidence_Logic*, and *g_Output_Logic*.

The group node *g_Bistable_Logic* in ⟨Fig. 9⟩ is decomposed into three group nodes *g_Rising_Trip*, *g_Falling_Trip*, and *g_Digital_Trip* as depicted in ⟨Fig. 4(a)⟩, and the group node *g_Bistable_Logic* is decomposed as ⟨Fig. 4(b)⟩. Finally three nodes in *g_Rising_Trip* are defined using SDT, FSM, and TTS respectively as ⟨Fig. 1, 2, 3⟩. As short of space, we introduce the part of AMS, *g_Rising_Trip*, as a typical NuSCR specification.

NuSCR Specification Supporting Tools To be useful in developing practical systems, we provide a robust and well-engineered tool, NuEditor, for specifying the NuSCR specification. In NuEditor, simple properties s.t. completeness and consistency checking can be supported. Also it produces the adequate PVS inputs to verify the structural properties such as input/output completeness, consistency, and circular dependencies in NuSCR specification. It is based on our technique in [7]. We are now developing an automatic translating procedure from NuSCR specification into SMV inputs to verify further sophisticated properties. ⟨Fig. 10⟩ represents the NuEditor we are developing. With this tool, we are going to specify the whole system of RPS(Reactor Protection System), which is a core control process of nuclear power plant system, as a part of KNICS [8] project in Korea.

6 Conclusion and Future Work

Software safety is an important property for safety critical systems and formal requirements specification is known as a means to the safety in the early phase of software development process. Nowadays, in the area of nuclear power plants systems, the formal specification of software requirements is an urgent problem that needs to be solved right away with the replacement of existing

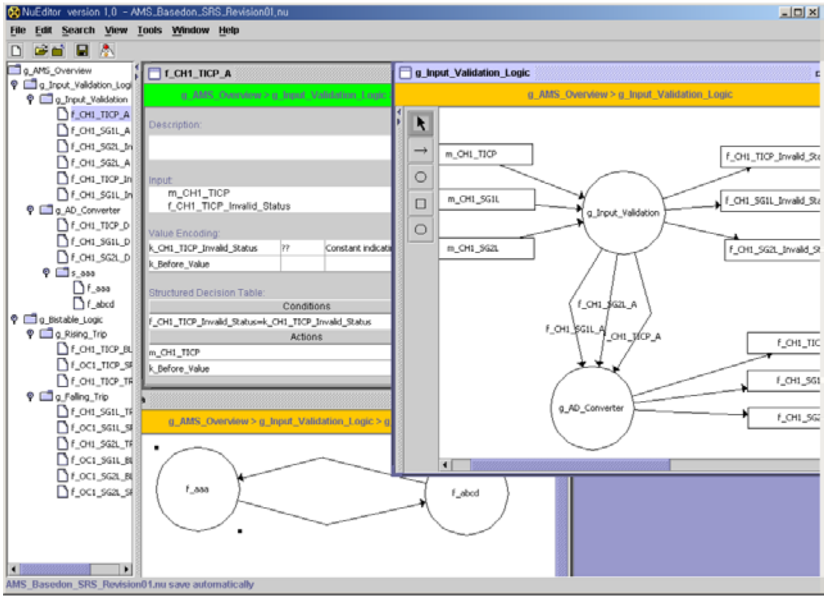


Figure 10: NuEditor

analog systems by digital systems composed of software process controllers.

In this paper, we introduce NuSCR, a formal software requirements specification method for digital protection system in nuclear power plants. NuSCR improves the readability and specifying ability by supplying different notations on the basis of the typical operation categories. The characteristics of the software process controller in nuclear power plants, s.t. periodic sequential processing and classifiable operations, makes this possible. We introduce the syntax and formal semantics of NuSCR to apply the recognized formal verification techniques to NuSCR specifications.

An ATWS mitigation system in Korean nuclear power plants is used as a case study to illustrate usefulness of our method. We also introduce the supporting tool, NuEditor, to be useful in developing practical systems. With this tool, we will specify the whole system of RPS(Reactor Protection System), which is a core control process of nuclear power plant system, as a part of KNICS [8] project in Korea. We are also developing an automatic translating procedure from NuSCR specification into SMV inputs to verify further sophisticated properties.

References

[1] D. Parnas A.J. Schouwen Van and J. Madey. Documentation of requirements for computer systems. In *RE'93: IEEE International Symposium on Requirements Engineering*, pages 198–207, 1993.

[2] E. A. Emerson Edmund M. Clarke and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Languages and Sysems*, 8(2):244–263, 1986.

[3] Orna Grumberg Edmund M. Clarke and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[4] J. Guttag and J. Horning. *Larch: Languages and Tools for Formal Specification*. Springer-Verlag, 1993.

[5] K. L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. Software Engineering*, SE-6(1):2–13, 1980.

[6] C. B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall International, 1986.

[7] T. Kim and S. Cha. Automated structural analysis of scr-style software requirements specifications using pvs. *Journal of Software Testing, Verification, and Reliability*, 11(3):143–163, 2001.

[8] KNICS. Korea nuclear instrumentation and control system research and development center. http://www.knics.re.kr/english/eindex.html.

[9] Nancy G. Leveson. *SAFEWARE, System safety and Computers.* Addison Wesley, 1995.

[10] K. M. McMillan. *Symbolic Model Checking.* Kluwer Academic, 1993.

[11] UK MoD. The procurement of safety critical software in defense equipment. Defense Standard 00-55, 1997.

[12] U.S. NRC. *Digital Instrumentation and Control Systems in Nuclear Power Plants: safety and reliability issues.* National Academy Press, 1997.

[13] D. Parnas and J. Madey. Functional documentation for computer systems engineering(version 2). CRL 237, Telecommunications Research Institute of Ontario(TRIO), McMaster Univ., Hamilton, Ontario, 1991.

[14] D.L. Parnas and J. Madey. Functional documentation for computer systems. *Science of Computer Programming*, 25(1):41–61, 1995.

[15] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 411–414, New Brunswick, NJ, USA, 1996. Springer Verlag.

[16] J. M. Spivey. *Introducing Z: a Specification Language and its Formal Semantics.* Cambridge Univerity Press, 1988.

[17] K.H. Britton R.A. Parker D.L. Parnas T.A. Alspaugh, S.R. Faulk and J.E. Shore. Software requirements for the A-7E aircraft. NRL 9194, Naval Research Laboratory, Washington, D.C., 1992.

[18] Zohar Manna Thomas A. Henzinger and Amir Pnueli. Timed transition systems. In *REX Workshop*, pages 226–251, 1991.

[19] Kori NPP Unit1. Functional requirements for ATWS mitigation systems for Kori NPP unit1. 1A110-IC-FR101, 2001.

[20] Kori NPP Unit1. Software requirements specification for ATWS mitigation systems for Kori NPP unit1. 1A110-IC-SX101, 2001.

[21] D. vam Dalen. *Logic and Structure.* Springer-Verlag, 3 edition, 1994.

[22] WolsongNPP2/3/4. Software work practice, procedure for the specification of software requirements for safety critical software. 00-68000-SWP-002, Sept. 1991.

[23] WolsongNPP2/3/4. Software requirements specification for shutdown systems 2 PDC. 86-68350-SRS-001, June 1993.