Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 74 (2005) 73–83

The Journal of Systems and Software

www.elsevier.com/locate/jss

A formal software requirements specification method for digital nuclear plant protection systems

Junbeom Yoo [a,*], Taihyo Kim [a], Sungdeok Cha [a], Jang-Soo Lee [b], Han Seong Son [b]

[a] *Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST) and AITrc/SPIC, 373-1, Kusong-dong, Yusong-gu, Taejon 305701, South Korea*

[b] *Korea Atomic Energy Research Institute (KAERI), MMIS team, 150, Deokjin-dong, Yusong-gu, Taejon, South Korea*

Received 18 March 2003; received in revised form 29 September 2003; accepted 5 October 2003
Available online 30 December 2003

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 providing graphical or tabular notations depending on the type of operations. NuSCR specifications can be formally analyzed for completeness, consistency, and against the properties specified in temporal logic. We introduce the syntax and semantics of NuSCR and demonstrate the effectiveness of the approach using reactor protection system, digital protection system being developed in Korea, as a case study.
© 2003 Elsevier Inc. All rights reserved.

Keywords: Formal specification; Software requirements; Digital plant protection system

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 (Leveson, 1995). Formal software requirements specification is known as a means to increase the safety of such systems in the early phase of software development process. It guides the developer to specify all requirements explicitly without any assumptions or omissions. In addition, formal specification can be verified using tools such as model checker (McMillan, 1993; Holzmann, 1997) or theorem prover (Owre et al., 1996).

Formal specification and verification of software requirements for instrumentation and control system for nuclear power plant has become the subject of extensive research as old and hardware-based analog systems are being replaced with software-based digital power plant protection system (NRC, 1997). Practical Formal Specification (PFS) project is a similar effort for aerospace applications (MoD, 1997).

Typical characteristics of digital protection controllers in nuclear power plant systems are as follows. First, numerous inputs, more than 80 for RPS which is being developed in Korea, 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 be kept 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.

[*] Corresponding author. Tel.: +82428695579; fax: +82428693510.
E-mail address: jbyoo@salmosa.kaist.ac.kr (J. Yoo).

0164-1212/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2003.10.018

NuSCR is a formal software requirements specification language which extends Software Cost Reduction (SCR) (Heninger, 1980) notation to specify functional requirements of safety critical software, especially nuclear power plants systems. It is based on *Parnas' Four-Variable Model* (Parnas and Madey, 1991) and uses Function Overview Diagram (FOD) for the overview of data flows in the same way as (WolsongNPP2/3/4, 1991). WolsongNPP2/3/4 (1991) is a variant of SCR, which was proposed by Atomic Energy of Canada Limited (AECL) and was used for the formal software requirements specification for ShutDown System 2 (SDS2) in Wolsong nuclear power plant in Korea.

In the approach of AECL, the state-based operations such as *trip set point hysteresis* are specified as functions. In fact, the AECL approach uses function to specify all aspects of requirements even though computation may require state-dependent information and timing constraints associated with it. Therefore, specification is sometimes awkward to specify and difficult to understand.

In the NuSCR approach, on the other hand, we use finite state machine (FSM) to specify state-dependent operations and timed transition system (TTS), a variant of automata (Henzinger et al., 1991) to specify timing-related requirements. NuSCR specification can be analyzed for structural correctness using PVS using techniques described in (Kim and Cha, 2001). 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 (McMillan, 1993), 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. We also compare the existing AECL approach with our proposed NuSCR to evaluate objectively their strengths and weakness. In Section 4, we represent the formal semantics of NuSCR software requirement specifications. We then briefly introduce NuSCR requirements specification for Reactor Protection System (RPS) as a case study, and describe the software development environment briefly 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 (Spivey, 1988), VDM (Jones, 1986), and Larch (Guttag and Horning, 1993) 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. While such notations are rich in expressiveness, application experts may find the notation to be difficult to write, read and review.

SCR (Heninger, 1980) was introduced in the early 80s 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 (Parnas and Madey, 1995; Van et al., 1993). 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 (Alspaugh et al., 1992), submarine communication system (Heitmeyer and McLean, 1983), and safety-critical component of Darlington nuclear power plant in Canada (Van et al., 1993).

The approach (Van et al., 1993) 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 ShutDown System 2 (SDS2) in Wolsong nuclear power plant in Korea (WolsongNPP2/3/4, 1993). The approach extends SCR in several ways. First, it combined the three tables of SCR (i.e., the mode transition table, event table, and condition table) into a table named Structured Decision Table (SDT). It uses Function Overview Diagram (FOD), similar to Data Flow Diagram (DFD), to graphically illustrate the overview of the system components and dependencies. Finally, it provides sophisticated functions for describing requirements related to precision and tolerance on timing constraints.

While most nuclear engineers and software engineers find the AECL notation to be easy to understand, there were some limitations. They are: (1) SDTs can be excessively complicated for engineers to review. AECL guidelines require all the conditions specified in each row of the SDT to be "atomic" (e.g., without using boolean operations). Therefore, in one case of Wolsung SDS2 specification, a SDT required 14 rows and 16 columns to fully specify functionality. However, mathematical formulas used in the SDTs are well-known as a whole in nuclear engineering community, and there is no need to separately verify the correctness of each term used in mathematical specification. (2) Management of time-related features 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. It also complicates FOD by adding additional notations for timing constraint.

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* (Parnas and Madey, 1991). 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_{SE} is a set of all system entities, defined as $V_{\mathrm{SE}} = V_{\mathrm{I}} \cup V_{\mathrm{F}} \cup V_{\mathrm{H}} \cup V_{\mathrm{TH}} \cup V_{\mathrm{O}}$
  - 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
- D_{SE}: a set of all possible valuation domain for every r in V_{SE}

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

Let $r \in V_{\mathrm{SE}}$, $v_r \in D_{\mathrm{SE}}$, $a, b \in N$, and $\otimes \in \{=, \neq, \leqslant, <, \geqslant, >\}$,

$$simple\_condition := r \otimes v_r | r \otimes r | TRUE | FALSE$$
$$complex\_condition := simple\_condition \wedge simple\_condition$$
$$| \; simple\_condition \vee simple\_condition |$$
$$\neg simple\_condition | simple\_condition$$
$$timed\_condition := [a, b] complex\_condition$$

It should be noted that timed condition is a complex condition which as timing constraints expressed as a duration. Timed condition is used to define timed history variables whereas complex condition is written as an expression containing function variables and history variables.

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

Let $r \in V_{\mathrm{SE}}$, $v_r \in D_{\mathrm{SE}}$, $a, b \in N$, and $\oplus \in \{+, -, *, \div\}$

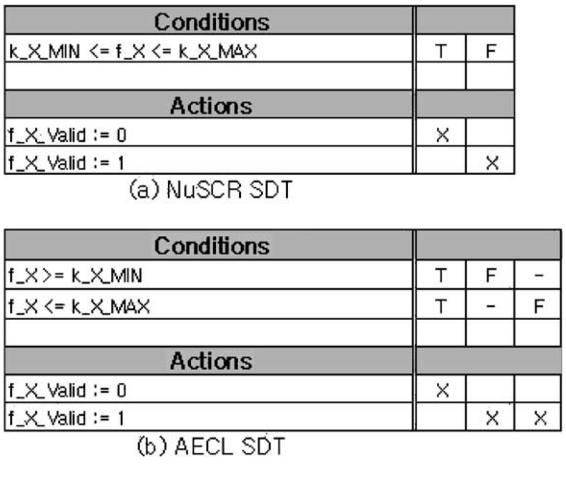
$$assignment := (r := v_r) | (r := r) | (r := r \oplus r) | (r := r \oplus v_r)$$

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.

Fig. 1 is an SDT defining function variable *f_X_Valid*. Two SDTs are equivalent in that they both specify "if *f_X* is smaller than *k_X_MAX* and larger than *k_X_MIN*, then the new value for the *f_X_Valid* is 0. Otherwise the output value of *f_X_Valid* is 1".

A key difference between SDTs used in NuSCR and AECL approaches is that the former allows conditions associated with each row to be arbitrarily complex boolean expression. SDTs used in NuSCR tend to be concise and provides superior readability.

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



Conditions		
k_X_MIN <= f_X <= k_X_MAX	T	F
Actions		
f_X_Valid := 0	X	
f_X_Valid := 1		X

(a) NuSCR SDT

Conditions			
f_X >= k_X_MIN	T	F	-
f_X <= k_X_MAX	T	-	F
Actions			
f_X_Valid := 0	X		
f_X_Valid := 1		X	X

(b) AECL SDT

Fig. 1. Structured decision tables for *f_X_Valid*.

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.

Fig. 2 is an FSM defining history variable *h_X_OB_Sta*. Input entities for this variable is *f_X* and *f_X_OB_Perm*, and the initial state is *Not_OB_State*.

Timed history variable. Timed history variables are used for specifying the time-related behavior of system.

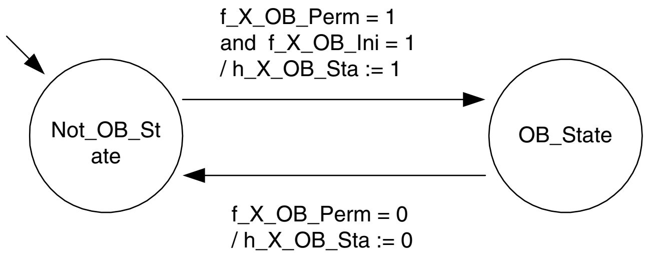
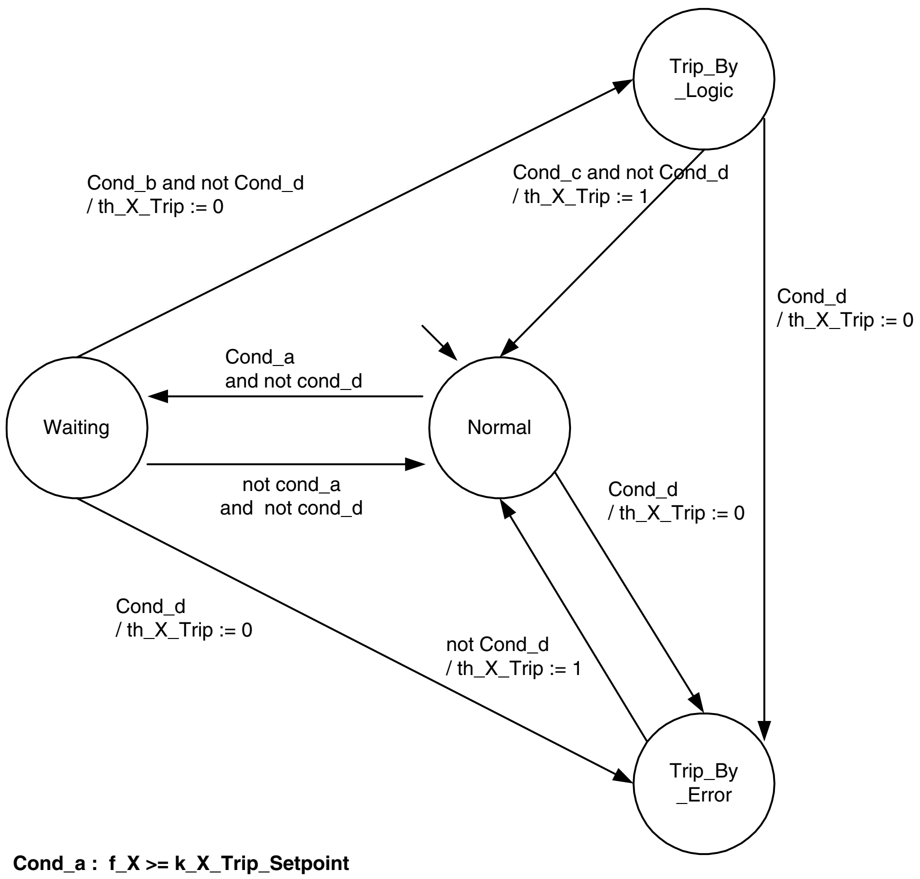


Fig. 2. Finite state machine for *h_X_OB_Sta*.

They are defined as a kind of TTS (Henzinger et al., 1991). TTS is an FSM extended with the timing constrains $[a, b]$ in transition conditions. $[a, b]$ means the time duration between time *a* and *b*.

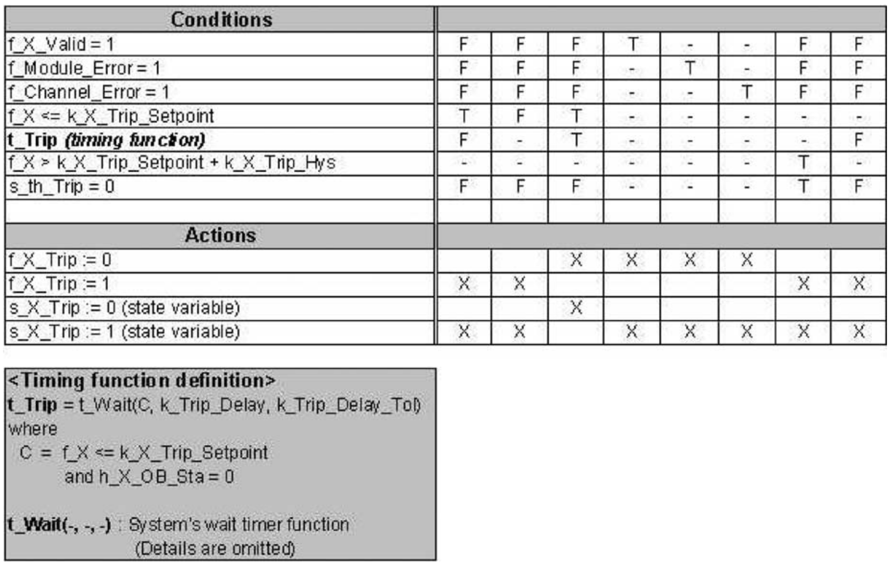
Fig. 3 is a TTS defining timed history variable *th_X_Trip*. Input entities for this variable are *f_X* and *h_X_OB_STA*, and the initial state is *NORMAL*. The detailed interpretation of such definition is as follows: "If *f_X* >= *k_X_Trip_Setpoint*, written as a macro named cond_a, and the condition defined in the macro cond_d is false, the system moves to the waiting state. (Other requirements are omitted.)" The TTS expression in *Cond_b*, [*k_Trip_Delay*, *k_Trip_Delay*] means that the condition has to remain true for *k_Trip_Delay* units.

Fig. 4 captures semantically equivalent notation using the AECL approach. The complete definition of timing functions are explained in (Van et al., 1993; WolsongNPP2/3/4, 1993). As depicted in above figures, in NuSCR, timing constraints are defined as TTS based on the transition of state as time passes. On the other hand, in the AECL approach, they enumerate each condition and mark whether they are effective or not. Timing functions, s.t. *t_trip* and *t_Wait*, are something



Cond_a : f_X >= k_X_Trip_Setpoint
Cond_b : *[k_Trip_Delay, k_Trip_Delay]* **(f_X >= k_X_Trip_Setpoint and h_X_OB_Sta = 0)**
Cond_c : f_X < k_X_Trip_Setpoint - k_X_Trip_Hys
Cond_d : f_X_Valid = 1 or f_Module_Error = 1 or f_Channel_Error = 1

Fig. 3. Timed transition system for *th_X_Trip*.



Conditions								
f_X_Valid = 1	F	F	F	T	-	-	F	F
f_Module_Error = 1	F	F	F	-	T	-	F	F
f_Channel_Error = 1	F	F	F	-	-	T	F	F
f_X <= k_X_Trip_Setpoint	T	F	T	-	-	-	-	-
t_Trip *(timing function)*	F	-	T	-	-	-	-	F
f_X > k_X_Trip_Setpoint + k_X_Trip_Hys	-	-	-	-	-	-	T	-
s_th_Trip = 0	F	F	F	-	-	-	T	F
Actions								
f_X_Trip := 0			X	X	X	X		
f_X_Trip := 1	X	X					X	X
s_X_Trip := 0 (state variable)			X					
s_X_Trip := 1 (state variable)	X	X		X	X	X	X	X

<Timing function definition>
t_Trip = t_Wait(C, k_Trip_Delay, k_Trip_Delay_Tol)
where
  C = f_X <= k_X_Trip_Setpoint
    and h_X_OB_Sta = 0

t_Wait(-, -, -) : System's wait timer function
    (Details are omitted)

Fig. 4. AECL SDT table for *f_X_Trip* and timing function *t_Trip*.

special to define and use, and also complicate the FOD. Timed-history variable, which is defined as one TTS node, makes FOD more concise to understand. Experience from the nuclear domain engineers in KNICS project in Korea (KNICS, 2001) says that the specification with NuSCR is more convenient and understandable.

Function overview diagram. Function overview diagram (FOD) 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 node, and the relationship between them is represented by unidirectional arrows. FOD is composed hierarchically and in this case the group nodes are used. Each node follows the naming convention mentioned earlier.

Fig. 5 is a FOD for *g_LOG_PWR* which is the logic for fixed set-point rising trip with operating bypass in RPS of digital plant protection system. Nodes *f_X_Valid*, *h_X_OB_Sta*, and *th_X_Trip* are defined in (Figs. 1–3) respectively. NuSCR distinguishes these timing and state variable related features using prefix naming convention (s.t. *th_*) and the shape of nodes.

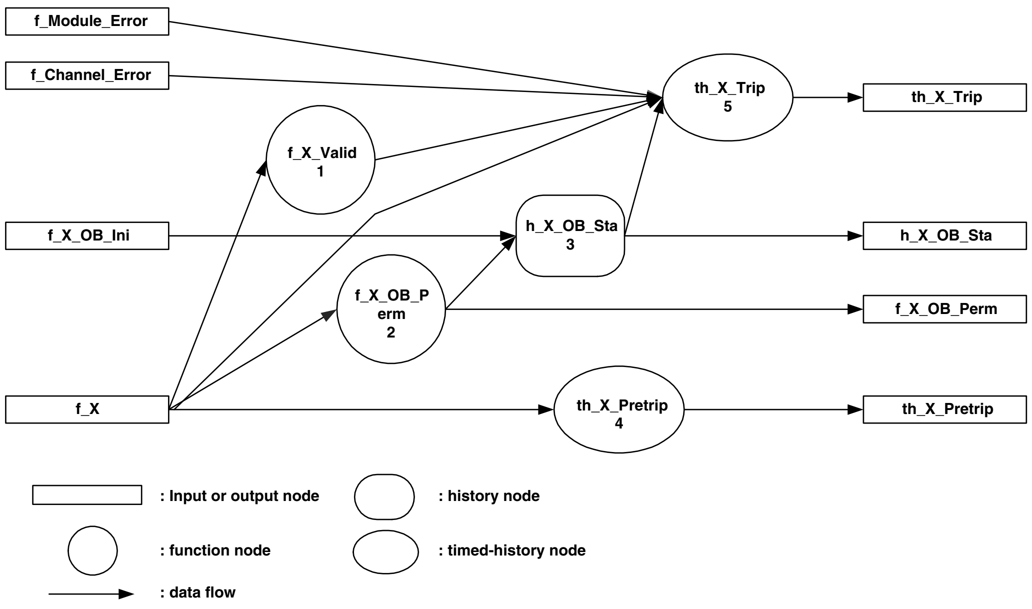


Fig. 5. Function overview diagram for *g_LOG_PWR*.

4. Semantics for NuSCR

To specify the meaning of software system written in NuSCR, we need to define an state valuation function σ (Tennent, 1976). If S is the set of all possible "states" of variables in V_{SE}, σ defines a correspondence between every variable and the value that is its current contents. Therefore it is convenient to model σ by function with domain V_{SE} and co-domain D_{SE}:

$$\sigma : S = V_{\mathrm{SE}} \rightarrow D_{\mathrm{SE}}$$

Then for any variable V in V_{SE}, $\sigma[V]$ is the contents of V in current state. The notation $\sigma[d/V]$ is used to update a state. It means the state σ' such that $\sigma'[V] = d$ and for all $V' \neq V$ in V_{SE}, $\sigma'[V'] = \sigma[V']$. That is, σ' is the same function as σ except at the argument V which is mapped into d.

The behavior of software system specified by NuSCR can be defined based on the behavior of FOD, which is a function from system input I to system output O. FOD is also based on the behavior of three basic variable nodes in FOD, which are the functions as follows:

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 *comple_conditions* in SDT are the predicate on V_{FI}, and actions are the *assignments* on V_{FO} which is the function variable itself. Therefore, a function variable node can be defined as a function f_{FV} with input values I_{FV} to output values O_{FV} as follows.

$$f_{\mathrm{FV}} : I_{\mathrm{FV}} \rightarrow O_{\mathrm{FV}}$$

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

SDT: a set of pair (p, a)

- $p \in \mathit{Predicate}$ and $a \in \mathit{Action}$
- if $p[I_{\mathrm{FV}}/V_{\mathrm{FI}}]\sigma = \mathit{TRUE}$ then $a(\sigma) = \sigma[O_{\mathrm{FV}}/V_{\mathrm{FO}}] = \sigma'[V_{\mathrm{FO}}]$

Conditions			
Cond$_1$	T	-	T
Cond$_2$	F	T	-
Cond$_3$	-	F	T
Actions			
Assign$_1$	X		
Assign$_2$		X	
Assign$_3$			X

Fig. 6. Structured decision table for a function variable.

For example, SDT in Fig. 6 can be defined as follows:

$$\begin{aligned} SDT = \{&(Cond_1 = T \wedge Cond_2 = F, Assign_1),\\ &(Cond_2 = T \wedge Cond_3 = F, Assign_2),\\ &(Cond_1 = T \wedge Cond_3 = T, Assign_3)\} \end{aligned}$$

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's transition labels are the predicate on V_{HI} and actions are *assignments* on V_{HV} which is the history variable itself. Therefore, a history variable node can be defined as a function f_{HV} from input values I_{HV} to output values O_{HV} as follows:

$$f_{\mathrm{HV}} : I_{HV} \rightarrow O_{\mathrm{HV}}$$

FSM can be defined as a relation described below:

$$FSM = \langle S_{\mathrm{H}}, s_0, C, A, R\rangle$$

- S_{H}: a set of all states in history variable node
- s_0: initial state in S_{H}
- C: a set of *complex_conditions*
- A: a set of *assignments*
- R:
  - a transition relation $S_{\mathrm{H}} \times C \times A \times S_{\mathrm{H}}$
  - $\exists r(s, c, a, s') \in R$ and $\exists \mathit{current\_state} \in CS_{\mathrm{H}}$, such that if $\mathit{current\_state} = s$ and $c[I_{\mathrm{HV}}/V_{\mathrm{HI}}]\sigma = \mathit{TRUE}$, then $a(\sigma) = \sigma[O_{\mathrm{HV}}/V_{\mathrm{HO}}] = \sigma'[V_{\mathrm{HO}}]$ and $\mathit{current\_state}' = s'$

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. 7) can be defined as a relation as follows:

$$\begin{aligned} FSM &= \langle S_{\mathrm{H}}, s_0, C, A, R\rangle\\ S_{\mathrm{H}} &= \{S_1, S_2, S_3\}\\ s_0 &= S_1 \end{aligned}$$

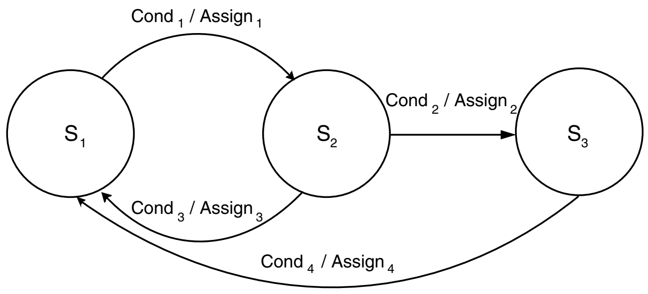


Fig. 7. Finite state machine for history variable node.

$C = \{Cond_1, Cond_2, Cond_3, Cond_4\}$
$A = \{Assign_1, Assign_2, Assign_3, Assign_4\}$
$R = \{(S_1, Cond_1, Assign_1, S_2),$
$(S_2, Cond_3, Assign_3, S_1), (S_2, Cond_2, Assign_2, S_3),$
$(S_3, Cond_4, Assign_4, S_1)\}$

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. Therefore, 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:

$$f_{\mathrm{THV}} : I_{\mathrm{THV}} \rightarrow O_{\mathrm{THV}}$$

TTS can be defined as a relation described below:

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

- 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* or *complex_conditions*
- A: a set of *assignments*
- R:
  - a transition relation $S_{\mathrm{TH}} \times C \times A \times S_{\mathrm{TH}}$
  - $\exists r(s, c, a, s') \in R$ and $\exists current\_state \in CS_{\mathrm{TH}}$, such that if $current\_state = s$ and $c[I_{\mathrm{THV}}/V_{\mathrm{THI}}]\sigma = TRUE$, then $a(\sigma) = \sigma[O_{\mathrm{THV}}/V_{\mathrm{THO}}] = \sigma'[V_{\mathrm{THO}}]$ and $current\_state' = s'$

current_state is a variable in CS_{TH}, which indicates the current state and the current local time. 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. 8 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.

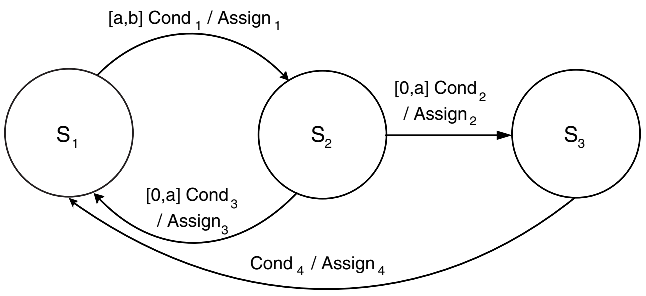


Fig. 8. Timed transition system for a timed history variable node.

```
if $current\_state = (S_1 \wedge lc < a)$
    then $next\_state := (current\_state, lc := lc + 1)$
else if $current\_state = (S_1 \wedge a \leqslant lc < b)$
    then $next\_state := (S_2, lc := 0)$ or $next\_state := (current\_state, lc := lc + 1)$
else if $current\_state = (S_1 \wedge lc = b)$
    then $next\_state := (S_2, lc := 0)$
```

For example, timed history variable in (Fig. 8) can be defined as a relation as follows:

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

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 50 ms and delay time a is 150 ms). Of course, we need to adjust that a or b are the multiple of d.

Function overview diagram. FOD in NuSCR describes the relationship between constructs in V_{SE}. Let I_{FOD} be the set of input values from out of FOD (i.e., environment or other FODs) into the FOD. Let O_{FOD} be the set of output values from FOD. They can be mapped into the set of variables, V_{FODI} and V_{FODO} respectively. Also as all nodes in FOD have partial orders according to their execution order and all nodes are defined as functions, f_{FOD} can be represented as a function composition of all nodes in FOD according to the partial orders on their precedence. Therefore, FOD can be defined as a function f_{FOD} from input values I_{FOD} to output values O_{FOD}.

$$f_{\mathrm{FOD}} = f_n \circ \cdots \circ f_2 \circ f_1$$

$$f_{\mathrm{FOD}} : I_{\mathrm{FOD}} \to O_{\mathrm{FOD}}$$

FOD can be defined as a tuple as follows:

$FOD = \langle N, T \rangle$

- N
  - a set of all nodes in FOD
  - all nodes in V_{F} and V_{H} and V_{TH} are defined as functions
  - V_{FODI} in V_{I} is a set of input variable nodes in FOD, which mapped as $\sigma[I_{\mathrm{FOD}}/V_{\mathrm{FODI}}] = \sigma'[V_{\mathrm{FODI}}]$
  - V_{FODO} in V_{O} is a set of output variable nodes in FOD, which mapped as $\sigma[O_{\mathrm{FOD}}/V_{\mathrm{FODO}}] = \sigma'[V_{\mathrm{FODO}}]$
- 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

For example, FOD in (Fig. 9) can be defined as follows:

$$\begin{aligned} &FOD = \langle N, T \rangle \\ &\quad N = \{I\_1, I\_2, I\_3, I\_4, f\_A, f\_B, h\_C, th\_D, th\_E, \\ &\qquad O\_1, O\_2, O\_3, O\_4\} \\ &\quad T = \{\{I\_1, th\_E\}, \{I\_2, th\_E\}, \{I\_3, h\_C\}, \{I\_4, f\_A\}, \\ &\quad \{I\_4, th\_E\}, \{I\_4, f\_B\}, \{I\_4, th\_D\}, \{f\_A, th\_E\}, \\ &\quad \{f\_B, h\_C\}, \{h\_C, th\_E\}, \{th\_E, O\_1\}, \{h\_C, O\_2\}, \\ &\quad \{f\_C, O\_C\}, \{th\_D, O\_4\}\} \end{aligned}$$

$$\begin{aligned} f_{\mathrm{FOD}} = {} & f_{th\_D} \circ f_{th\_E} \circ f_{h\_C} \circ f_{f\_B} \\ & \circ f_{f\_A} \text{ (other sequences are also possible)} \end{aligned}$$

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, which is defined as $\sigma[V_{\mathrm{SE}} \times CS_{\mathrm{H}} \times CS_{\mathrm{TH}}]$
- S_0: initial state in S
- R: a set of transition relation $S \times I \to S' \times O$, where I and O are system's input and output values respectively
- d: system scan cycle time in which the system get the changed valuation function σ periodically

NuSCR software system *NSS* uses the definitions of all three basic constructs and FOD. *NSS* gets inputs I from the out of system (i.e., environment), calculates with them, and then emits outputs O to the outside. In each time that NSS emits outputs, NSS changes its internal system states according to the behavior of NSS. The states of NSS is defined as $\sigma[V_{\mathrm{SE}} \times CS_{\mathrm{H}} \times CS_{\mathrm{TH}}]$, where CS_{H} and CS_{TH} mean the set of current state of history variable node and timed history variable node respectively. It means the current contents of all variables used in NSS. The behavior of NSS is defined based on a function f_{FOD} defined above. That is, between system states, there exists transition relation s.t. R, and it corresponds to $O = f_{\mathrm{FOD}}(I)$. NSS also operates periodically with system scan cycle time d. With every time interval d, it gets the changed valuation function σ for the inputs and outputs of NSS. This periodic behavior of NSS is an essential part for digital plant protection system in nuclear power plants, which requires the strict scan cycle time.

5. Case study: RPS example

In this section, we introduce NuSCR software requirements specification for Reactor Protection System (RPS), which is presently at developing in KINCS (KNICS, 2001), Korea.

KNICS RPS. Plant Protection System (PPS), which is presently being developed at KNICS in Korea, is composed of Reactor Protection System (RPS), Engineering

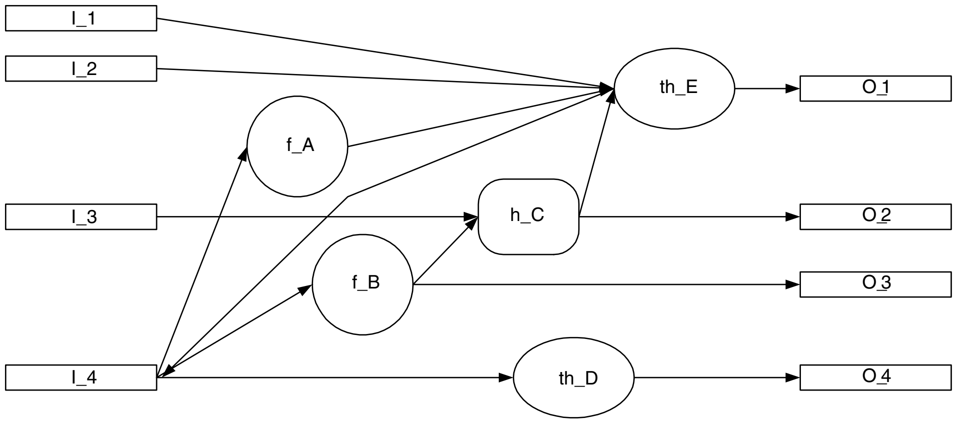


Fig. 9. Function overview diagram example.

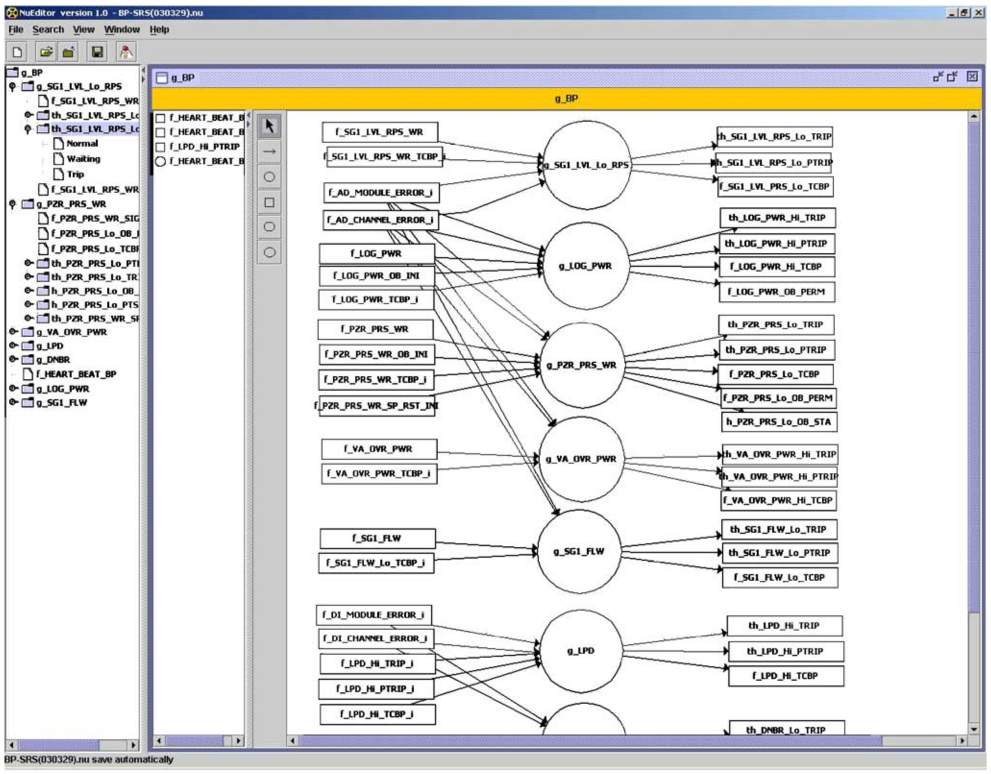
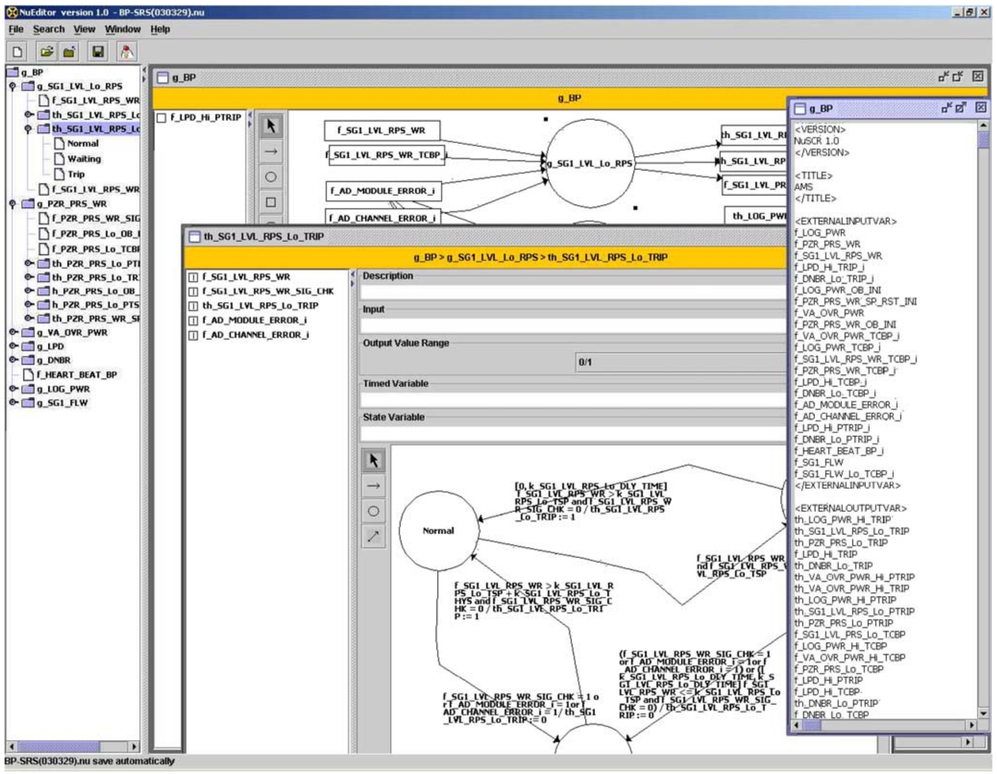


Fig. 10. FOD for *g_BP*.



```
[0, k_SG1_LVL_RPS_Lo_DLY_TIME]
f_SG1_LVL_RPS_WR > k_SG1_LVL_
RPS_Lo_TSP and f_SG1_LVL_RPS_W
R_SIG_CHK = 0 / th_SG1_LVL_RPS
_Lo_TRIP := 1

f_SG1_LVL_RPS_WR > k_SG1_LVL_R
PS_Lo_TSP + k_SG1_LVL_RPS_Lo_T
HYS and f_SG1_LVL_RPS_WR_SIG_C
HK = 0 / th_SG1_LVL_RPS_Lo_TRI
P := 1

f_SG1_LVL_RPS_WR
nd f_SG1_LVL_RPS_
VL_RPS_Lo_TSP

(f_SG1_LVL_RPS_WR_SIG_CHK = 1
or f_AD_MODULE_ERROR_i = 1 or f
_AD_CHANNEL_ERROR_i = 1) or ([
k_SG1_LVL_RPS_Lo_DLY_TIME, k_S
G1_LVL_RPS_Lo_DLY_TIME] f_SG1
LVL_RPS_WR <= k_SG1_LVL_RPS_Lo
_TSP and f_SG1_LVL_RPS_WR_SIG_
CHK = 0) / th_SG1_LVL_RPS_Lo_T
RIP := 0

f_SG1_LVL_RPS_WR_SIG_CHK = 1 o
r f_AD_MODULE_ERROR_i = 1 or f_
AD_CHANNEL_ERROR_i = 1 / th_SG1
_LVL_RPS_Lo_TRIP := 0
```

```
<VERSION>
NuSCR 1.0
</VERSION>

<TITLE>
AMS
</TITLE>

<EXTERNALINPUTVAR>
f_LOG_PWR
f_PZR_PRS_WR
f_SG1_LVL_RPS_WR
f_LPD_Hi_TRIP_i
f_DNBR_Lo_TRIP_i
f_LOG_PWR_OB_INI
f_PZR_PRS_WR_SP_RST_INI
f_VA_OVR_PWR
f_PZR_PRS_WR_OB_INI
f_VA_OVR_PWR_TCBP_i
f_LOG_PWR_TCBP_i
f_SG1_LVL_RPS_WR_TCBP_i
f_PZR_PRS_WR_TCBP_i
f_LPD_Hi_TCBP_i
f_DNBR_Lo_TCBP_i
f_AD_MODULE_ERROR_i
f_AD_CHANNEL_ERROR_i
f_LPD_Hi_PTRIP_i
f_DNBR_Lo_PTRIP_i
f_HEART_BEAT_BP_i
f_SG1_FLW
f_SG1_FLW_Lo_TCBP_i
</EXTERNALINPUTVAR>

<EXTERNALOUTPUTVAR>
th_LOG_PWR_Hi_TRIP
th_SG1_LVL_RPS_Lo_TRIP
th_PZR_PRS_Lo_TRIP
f_LPD_Hi_TRIP
th_DNBR_Lo_TRIP
th_VA_OVR_PWR_Hi_PTRIP
th_VA_OVR_PWR_Hi_TRIP
th_LOG_PWR_Hi_PTRIP
th_SG1_LVL_RPS_Lo_PTRIP
th_PZR_PRS_Lo_PTRIP
f_SG1_LVL_PRS_Lo_TCBP
f_LOG_PWR_Hi_TCBP
f_VA_OVR_PWR_Hi_TCBP
f_PZR_PRS_Lo_TCBP
f_LPD_Hi_PTRIP
f_LPD_Hi_TCBP
th_DNBR_Lo_PTRIP
f_DNBR_Lo_TCBP
```

Fig. 11. A screen-dump of NuEditor.

Safety Features-Component Control System (ESF-CCS), and Automatic test and Interface Processor (ATIP). RPS plays the role of protecting reactor, and ESF-CCS is used for reducing the influence of other accidents, i.e. Loss of Coolant Accident (LOCA). ATIP tests RPS and ESF-CCS automatically. In this section, we present the NuSCR formal software requirement specification for BP (Bistable Logic), which is the most important parts in RPS.

NuSCR specification for BP. Each BP gets 18 safety sensors input periodically, and performs the necessary comparison logic calculation according to each input to determine the safety state of the reactor. Each input is calculated independently as its trip calculation logic. Fig. 10 is a part of NuSCR specification for RPS BP(*g_BP*). This figure is a screen dump of NuEditor, which is the specification and analysis assistant tool for NuSCR. In Fig. 10, the quadrangle shaped nodes in the left mean the input variables for *g_BP*, and the right ones are the output from *g_BP*. The left part in NuEditor represents the hierarchy of FOD, which is at specifying level, and the editing window is located at the right.

g_LOG_PWR, which is the second nodes in the 18 independent modules consisting BP depicted in Fig. 10, is a fixed set-point rising trip with operating bypass logic. It means that the trip set-point is fixed and trip occurs if the input value falls above the set-point. The FOD of this trip logic module is depicted in Fig. 4 in Section 3. The full names of each variable are simplified to the concise ones for convenience.

NuSCR Specification Supporting Tool. 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 (Kim and Cha, 2001). We are now developing an automatic translating procedure from NuSCR specification into SMV inputs to verify further sophisticated properties. Fig. 11 represents the NuEditor, we are developing. With this tool, we have finished to specify the whole system of Reactor Protection System (RPS), which is a core control process of nuclear power plant system, as a part of KNICS 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 plant systems, the formal specification of software requirements is an urgent problem that needs to be solved right away with the replacement of existing 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 RPS system in digital nuclear power plants protection system, which is presently at developing in Korea 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 specified the whole system of RPS, which is a core control process of nuclear power plant system, as a part of KNICS (KNICS, 2001) project in Korea. We are also developing an automatic translating procedure from NuSCR specification into SMV inputs to verify further sophisticated properties.

References

Alspaugh, T., Faulk, S., Britton, K., Parker, R., Parnas, D., Shore, J., 1992. Software requirements for the A-7E aircraft. NRL 9194, Naval Research Laboratory, Washington, DC.

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

Heitmeyer, C., McLean, J., 1983. Automatic verification of finite-state concurrent systems using temporal logic specifications. IEEE Transactions on Software Engineering SE 9 (5), 580–589.

Heninger, K.L., 1980. Specifying software requirements for complex systems: new techniques and their application. IEEE Transactions on Software Engineering SE 6 (1), 2–13.

Henzinger, T.A., Manna, Z., Pnueli, A., 1991. Timed transition systems. In: REX Workshop. pp. 226–251.

Holzmann, G., 1997. The model checker spin. IEEE Transaction on Software Engineering 23 (5), 279–295.

Jones, C.B., 1986. Systematic Software Development Using VDM. Prentice-Hall International.

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

KNICS, 2001. Korea nuclear instrumentation and control system research and development center. Available from: <http://www.knics.re.kr/english/eindex.html>.

Leveson, N.G., 1995. SAFEWARE, System safety and Computers. Addison-Wesley.

McMillan, K.M., 1993. Symbolic Model Checking. Kluwer Academic.

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

NRC, US, 1997. Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues. National Academy Press.

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

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

Parnas, D., Madey, J., 1995. Functional documentation for computer systems. Science of Computer Programming 25 (1), 41–61.

Spivey, J.M., 1988. Introducing Z: A Specification Language and its Formal Semantics. Cambridge University Press.

Tennent, R., 1976. The denotational semantics of programming languages. Communication of the ACM 19 (8), 437–453.

Van, A.S., Parnas, D., Madey, J., 1993. Documentation of requirements for computer systems. In: RE'93: IEEE International Symposium on Requirements Engineering. pp. 198–207.

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

WolsongNPP2/3/4, June 1993. Software requirements specification for shut-down systems 2 PDC. 86-68350-SRS-001.