Development of Verification and Conformance Test Generation of Communication Protocol for Railway Signaling Systems

Jae-Ho Lee[1], Jong-Gyu Hwang[1], Mi-Seon Seo[2], Sung-Un Kim[2], Gwi-Tae Park [3]

[1] Train Control Research Corps, Korea Railroad Research Institute, Kyong gi-Do, Korea
(Tel.+82-31-460-5438, E-mail : jhlee1@krri.re.kr, jghwnag@krri.re.kr)
[2] Department of Telematics Engineering , Pukyong National University, Pusan, Korea
(Tel.+82-51-620-6476, E-mail : jljpms@mail1.pknu.ac.kr, kimsu@pknu.ac.kr)
[3] Department of Electrical Engineering, Korea University, Seoul, Korea
(Tel. +82-2-3290-1152, E-mail : gtpark@korea.ac.kr)

Abstract : Verification and testing are complementary techniques that are used to increase the level of confidence in the correct functioning of communication systems as prescribed by their specifications. This paper presents an experience of model checking for a formal railway signaling protocol specified in LTS (Labeled Transition System). This formal approach checks deadlock, livelock and reachability for the state and action to verify whether properties expressed in modal logic are true on specifications. We also propose a formal method for semi-automated test case generation for a railway signaling protocol described in I/O FSM (Input/Output Finite State Machine). This enables the generation of more complete and consistent test sequence for conformance testing. The above functions are implemented by C++ language and included within RSPVTE (Railway Signaling Protocol Verification and Testing Environment).

Keywords: Verification, Testing, Railway signaling system, LTS, I/O FSM, Formal method

1. INTRODUCTION

The primary objectives of protocol standardization are to allow systems developed by different vendors to work together, to exchange and handle information successfully. In recent years, the application of formal methods and software engineering methodologies to standardized protocol design has given rise to a new interdisciplinary called protocol engineering[1]. Actually the domain of protocol engineering includes all of the activities related to protocol development: protocol specification, validation and verification, synthesis, conversion, performance analysis, automatic implementation and conformance testing.

Verification and testing are complementary techniques that are used to increase the level of confidence in the correct functioning of communication systems as prescribed by their specifications. Verification can give certainty about satisfaction of a required property, but this certainty only applies to the model of the specification. On the other hand, conformance testing is the assessment of an implementation with its specification.

This paper presents an experience of model checking for a formal railway signaling protocol specified in LTS. It checks deadlock, livelock and reachability for the state and action. The implemented formal checker using modal mu-calculus is enable to verify whether properties (deadlock, livelock and reachability) expressed in modal logic are true on specifications. Throughout this paper, our interest also lies in a formal method for semi-automated test case generation for a railway protocol described in I/O FSM. We made use of several existing techniques to form an integrated framework for abstract test case generation from a railway protocol specification. A prototype of the proposed framework has been built with special attention to real application in order to generate the ETS (Executable Test Suite) in an automatic way.

This paper is organized as follows: Section 2 describes some key concepts and notations in protocol verification and conformance testing. Section 3 describes LTS and I/OFSM specifications of a Korean Railway signaling system. Section 4 gives the functional description of a model checker which is able to formally verify whether properties expressed in modal logic are true on LTS specifications, and then gives the description of the method to generate test sequences from an intermediate model (reference I/O FSM) of railway signaling protocol, and finally conclusions are given in Section 5.

2. PRELIMINARIES

2.1. Definition of LTS

The formalism of a LTS is used for modeling the behavior of processes, systems and components. LTS serves as a semantic model for a number of protocol specification languages, e.g. CCS (Calculus of Communication System), CSP (Communicating Sequential Processes), ACP (Asynchronous Communicating Processes) and LOTOS [2].

Definition 1: A labeled transition system is a 4-tuple $<S, L, T, s_o>$ with

- S is a (countable) non-empty set of states.
- L is a (countable) set of observable actions.
- $T \subseteq S \times (L \cup \{\tau\}) \times S$ is the transition relation.
- $s_o \in S$ is the initial state.

The labels in L represent the observable interactions of a system; the special label $\tau \notin L$ represents an unobservable internal action, and label ε represents null action.

2.2 Model Checking for Verification

Model checking is a verification technique that uses formulas of a modal or temporal logic to express properties of a system expressed in some other kind of specification language, and then match them against each other to decide whether the property holds for the system in question. Thus there are two languages involved in the verification process: the system specification language and the property specification language. We use finite state LTS to specify systems. It has been the most common specification paradigm in recent years.

We choose the modal mu-calculus as our property specification language. It is a reasonable compromise between

expressive power and complexing of model checking. Generally the major obstacle in applying finite state labeled transitions system checking to the correctness of large specifications is the combinatorial explosion of the state space arising when many loosely coupled parallel processes are considered. The problem also known as the state-explosion problem has been attacked from various sides.

To address this problem, the modal mu-calculus is originally due to Kozeņ [3] although in its current incarnation, modalities are parameterized by action names [4][5]. The modal mu-calculus is proposed as a highly expressive logic that can be used to specify safety and liveness properties of concurrent systems represented as labeled transition systems.

2.2.1 Modal mu-calculus

The modal mu-calculus can alternatively be viewed as the logic obtained by adding recursion to Hennessy-Milner logic [7]. More generally practitioners have found Hennessy-Milner logic useful to be enable to express temporal properties (such as liveness and safety) of concurrent systems. However as a logic it is not very expressive because formulas of the logic are not rich enough to express such temporal properties. So extra operators, external fixed points, are added in modal mu-calculus. The result is a very expressive temporal logic.

The modal mu-calculus, modal logic with external fixed points, is a very expressive propositional temporal logic. Syntactically, it consists of atomic propositions, $\wedge$ (conjunct), $\vee$ (disjunct), [ ] (necessarily), < > (possibly), μ (least fixed point), ν (greatest fixed point). This logic is often referred to as Lμ.

Formulas of the logic are defined as follows;

$$\Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K \rangle \Phi \mid \nu Z.\Phi \mid \mu Z.\Phi \quad (1)$$

Where tt is the atomic proposition that is true at every state, ff is the atomic proposition that is false at every state, Z ranges over a family of propositional variables, and K over subsets of A (set of action). The binder νZ is the greatest fixed point operator whereas μZ is the least fixed point operator.

The protocol has two properties in which it has a liveness with some good state and action, and a safety without deadlock and livelock. The modal mu-calculus is temporal logic with a logically powerful logic, in which it has both liveness and safety property.

2.2.2 Safety and Liveness

A safety property states that some bad feature is always precluded. Safety can either be ascribed to states, that bad states can never be reached, or to actions, that bad actions never happen. If the formula Φ captures the complement of those bad states, then $\nu Z.\Phi \wedge [-]Z$ expresses safety. Where [-] represents all actions.

A liveness property states that some good feature is eventually fulfilled. Again it can either be ascribed to states, that a good state is eventually reached, or to actions, that a good action eventually happens. If Φ captures the good states then $\mu Z.\Phi \vee (<->tt \wedge [-]Z)$ expresses liveness with respect to state. Where the presence of $(<->tt \wedge [-]Z)$ to ensure that Φ does become true.

2.3 Conformance of I/O FSM

In general, an I/O FSM is usually used for modeling the control part of communication protocols.

Definition 2: An I/O FSM is a 6-tuple <S, s_o, I, O, δ, t_r> where

- S is a set of finite states.
- $s_o \in S$ is the initial state.
- $I = \{i_1,..., i_n\}$ is a set of input symbols.
- $O = \{o_1, ..., o_n\}$ is a set of output symbols.
- δ is output function $S \times I \rightarrow O$.
- t_r is a transition function, $t_r \subseteq \{s\text{-}i/o \rightarrow s' \mid s, s' \in S \wedge i \in I \wedge o \in O\}$

2.3.1 Conformance Relation

Definition 3: Given I/O FSM model specifying a protocol description, the definition of conformance testing consists of three steps.

① The I/O FSM implementation is set to state s_i.

② Input is applied, and it is verified that output of the implementation I/O FSM is the same as the expected output of the specification I/O FSM.

③ The new state of the I/O FSM implementation is checked to verify that it is really s_j.

In order to generate automatically the optimal and efficient test cases for the given I/O FSM protocol specification, general methods for test generation are based on the specification S, which is in the form of strongly connected, minimal and deterministic I/O FSM. If these conditions are satisfied, the definition of conformity between specification I/O FSM and implementation I/O FSM is dedicated to the trace equivalence or observable equivalence.

2.3.2 Testing Framework

Given an I/O FSM representation of a specification, denoted henceforth as I/O FSM_s, and an implementation of this I/O FSM_s denoted as I/O FSM_I conforms to I/O FSM_s by testing I/O FSM_I as a black-box. In general the problem of conformance testing with the purpose of detecting faults in a black-box implementation is in general unsolvable unless it is dealt with in a restricted framework.

Solving this problem implies that we should

1) formally define the conformance relation between I/O FSM_I and I/O FSM_s;
2) generate from I/O FSM_s all sequences of test cases TS_I of inputs and its expected sequences of test cases TS_o of outputs;
3) apply TS_I to the input PCO (Points of Control and Observation) port of I/O FSM_I ;
4) observe all sequences TS_a of actual outputs at the output PCO port of I/O FSM_I;
5) compare TS_a with TS_o to determine the conformance of I/O FSM_I to I/O FSM_s.

2) to 5) are mainly related to the actual execution of the test sequence.

2.3.3 Automated Test Case Generation Method

Hence, for each transition, test case can be formally described as $\underline{R_i \bullet \text{shortest-path}(S_0\text{-}S_i) \bullet T_{ij}@UIO(S_j)}$, where R_i is the symbol that send testing object to initial state, shortest-path(S_0-S_i) is the shortest path from initial state S_0 to beginning state of TUT(Transition Under Test). And T_{ij} is the TUT that takes the specification I/O FSM_s from state s_i to state s_j, UIO(s_j) is a possible UIO(Unique Input/Output) sequence

for state s_j, which is a sequence of actions and @ is the concatenation symbol.

In fact, UIO sequences are used to test the resulting state after any transition of the I/O FSM.

3. RAILWAY SIGNALING PROTOCOL SPECIFICATION

In this section, as a reference model for verification and conformance test generation, we concentrate on a Korean railway signaling protocol between EIS (Electronic Interlocking System) and LDTS (Local Data Transmission System) systems. The LDTS located in the signaling equipment room receives the control commands from CTC (Centralized Traffic Control) system for the control of field signaling equipment such as EIS and others. This LDTS communicates with EIS to send the commands from CTC and receive the state information of field signaling equipment.

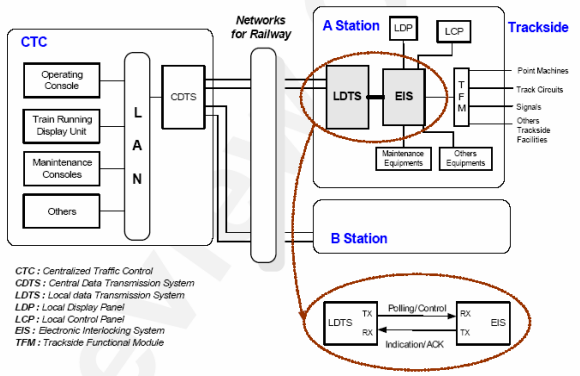


Figure 1. A Korean rail signaling systems

Figure 2 shows the frame format of transmission message between LDTS and EIS for the signaling system described above.

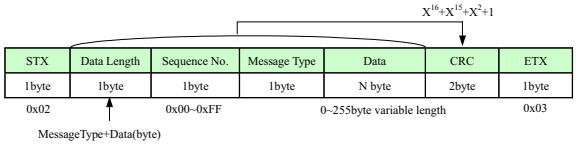


Figure 2. Frame format of transmission message

The 'STX(Start of Text)' field represents the start of the transmission message and the 'ETX(End of Text)' field represents the end of it. They are respectively marked 0x02 and 0x03. The 'Data Length' indicates the length from 'Message Type' to 'Data' in bytes. Also, the 'Sequence Number' used to show the transmitted order of message is marked from 0x00 to 0xFF, and it increases the number by one after successful transmission of a message frame. The 'Message Type' identifies the type of transmitting messages, and the 'Data' field contains variable transmitting data actually. The 'CRC' is used to check for error from 'Data Length' to 'Data' field, it has polynomial $X^{16}+X^{15}+X^2+1$.

Figure 3 specifies LTS for modeling the behavior of railway signaling process. This model consists of 6 states such as idle state (S_0), Master_Clock_Timer state (S_1), ack_await state (S_2), train_movement state (S_3), Polling_Timer state (S_4), and response_await state (S_5). This LTS model has 9 transitions as abbreviated as in Figure 3.

The corresponding I/O FSM model obtained from Figure 3 is describes in Figure 4. It has 9 states such as idle state, ack_await state, polling_detect state, system_initialized state, response_await state, retransmit state, message_detect state, abnormal_state, and restore state.

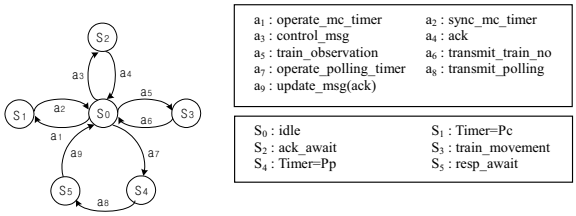
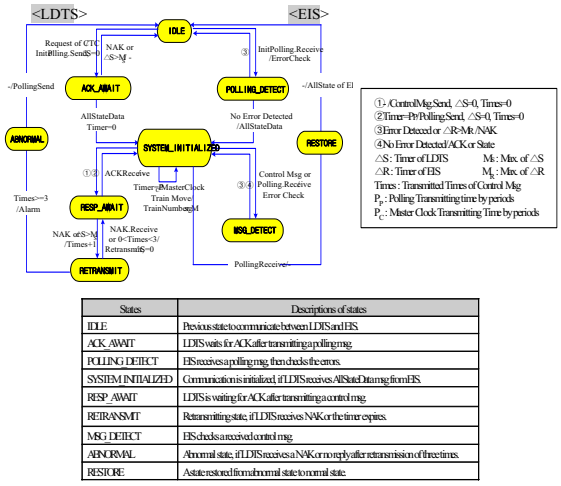


Figure 3. The LTS model generated from a Korean railway signaling protocol

The details of these states are respectively in the table of Figure4. This I/O FSM has four kinds of transitions like as abbreviated in Figure 4. The detail description of the actions of transitions is beyond of this paper.



States	Descriptions of states
IDLE	Previous state to communicate between LDTS and EIS.
ACK_AWAIT	LDTS waits for ACK after transmitting a polling msg.
POLLING DETECT	EIS receives a polling msg, then checks the errors.
SYSTEM_INITIALIZED	Communication is initialized, if LDTS receives AllStateData msg from EIS.
RESP_AWAIT	LDTS is waiting for ACK after transmitting a control msg.
RETRANSMIT	Retransmitting state, if LDTS receives NAK or the timer expires.
MSG DETECT	EIS checks a received control msg.
ABNORMAL	Abnormal state, if LDTS receives a NAK or no reply after retransmission of three times.
RESTORE	A state restored from abnormal state to normal state.

Figure 4. The I/O FSM obtained from the LTS shown in Fig. 3

4. VERIFICATION AND CONFORMANCE TESTING

4.1 LTS model checking by Modal mu-calculus

Consider the LTS model given in Figure 3; we wish to determine which states of this process satisfy the formula Lμ where {B1, B2}. Where Lμ is

$$\nu Z.\ (\mu Y.A \vee (\ <-> tt \wedge [-]Y)) \wedge [-]Z \text{ if, } A=\{S_0\}$$

$$B1 = \min\{Y = A \vee (< - > tt \wedge [-]Y)\}$$
$$B2 = \max\{Z = Y \wedge [-]Z)\} \quad (3)$$

Recall that states satisfying this formula have the safety property that along all paths emanating from S_0 state, it is always the case that A eventually holds.

Figure 5 contains the translation of this block set into block having only simple right-hand sides; it also shows the dependency graph corresponding to these new blocks. Note that variable X_1 corresponds to variable Y, while X_7 corresponds to Z.

We now trace through the steps of the algorithm as following. The first step requires a topological sorting of the block graph for B1 and B2. Figure 6 is edge-labeled directed graph G related to block B1 and B2. The only edge in this graph goes from B1 to B2, so B1 precedes B2 in the order.

$B_1 \equiv \min\{X_1 = X_2 \vee X_3$	$B_2 \equiv \max\{X_7 = X_1 \wedge X_8$
$X_2 = A$	$X_8 = [-]X_7\}$
$X_3 = X_4 \wedge X_5$	
$X_4 = [-]X_1$	
$X_5 = <->X_6$	
$X_6 = tt\}$	

Figure 5. Max block, min block

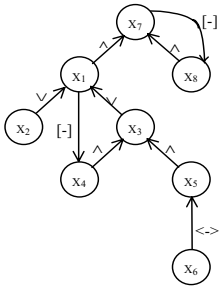


Figure 6. Edge-labeled directed graph G

The next step involves the initialization of the bit vectors and counters for each state, and the lists for each block. In what follows we represent state data structures using two arrays, one for the bits and one for the counters, with true bits being given as 1′s and false bits as 0′s. Unused counters are omitted. The results of the initialization phase are as follows.

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S_0	0	1	0	0	0	1	1	1
S_1	0	0	0	0	0	1	1	1
S_2	0	0	0	0	0	1	1	1
S_3	0	0	0	0	0	1	1	1
S_4	0	0	0	1	0	1	1	1
S_5	0	0	0	0	0	1	1	1

C	X_3	X_4
S_0	2	4
S_1	2	1
S_2	2	1
S_3	2	1
S_4	1	0
S_5	2	1

M[1]=<<S_0, X_2>, <S_4, X_4>, <S_0, X_6>, <S_1, X_6>, <S_2, X_6>, <S_3, X_6>, <S_4, X_6>, <S_5, X_6>>

M[2]=< >

Figure 7. Initialization of Bit-vector, counter and array M[i]

According to the description of update algorithm, the relevant bit vectors and counters are modified respectively. The data structures now look as follows.

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S_0	1	1	1	1	1	1	1	1
S_1	1	0	1	1	1	1	1	1
S_2	1	0	1	1	1	1	1	1
S_3	1	0	1	1	1	1	1	1
S_4	1	0	1	1	1	1	1	1
S_5	1	0	1	1	1	1	1	1

C	X_3	X_4
S_0	0	0
S_1	0	0
S_2	0	0
S_3	0	0
S_4	0	0
S_5	0	0

M[1]=< > M[2]=< >

Figure 8. Result of bit-vector, counter and array M[i]

Recall that our original question involves determining which state satisfied the formula X_5. By consulting the above bit vectors, we see that all states satisfy formula νZ. (μY.A ∨ (<-> tt ∧ [-]Y)) ∧ [-]Z, A={S_0}. Finally, the LTS model given in Figure 8 has no deadlock and livelock, it satisfies safety property.

4.1.2 Liveness Check

Now, we wish to determine which states of the process (the model in Figure 3) satisfy the liveness property, especially, state reachability. Where the formula Lμ is

$$\nu Z.\ (\mu Y.A \vee (<a_6> tt \wedge [a_5]Y)) \wedge [-]Z \quad \text{if, } A=\{S_3\} \tag{4}$$

$$B1 = \min\{Y = A \vee (< a_6 > tt \wedge [a_5]Y)\} \tag{5}$$

$$B2 = \max\{Z = Y \wedge [-]Z)\}$$

Recall that states satisfying this formula have the liveness property that a_6 action occured after a_5 action is eventually emanated from a special state(S_3).

The final values of the data structures are the following.

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S_0	0	0	0	1	0	1	0	0
S_1	1	0	1	1	0	1	0	0
S_2	1	0	1	1	0	1	0	0
S_3	1	1	1	1	1	1	0	0
S_4	1	0	1	1	0	1	0	0
S_5	1	0	1	1	0	1	0	0

C	X_3	X_4
S_0	1	0
S_1	0	0
S_2	0	0
S_3	0	0
S_4	0	0
S_5	0	0

M[1]=< >

M[2]=< >

Figure 9. Bit-vector, counter and array M[i]

To determine that the LTS model in Figure 3 satisfies the state reachability, we examine the formula X_2, X_4, X_5. If there is a state that X_2, X_4 and X_5 are simultaneously true, we call it satisfies the state reachability. Now, we can find a state S_3 satisfied above condition, so the LTS model given in Figure 3 satisfies liveness property, especially state reachability, and νZ. (μY.A ∨(<a_6> tt ∧ [a_5]Y)) ∧ [-]Z , A={S_3}.

4.2 Conformance Testing Generation using UIO Sequence

Theoretical research has made significant advances in the generation of test sequences from formal specifications and in the development of computer-aided test tools. However, these methods and tools are not too industrially related and do not quite address the problems facing tests in the industry. In practice, test sequence is manually generated by extracting test cases from natural language specification. These are largely based on the experience of testers who are often able to uncover bugs in the implementation. With the merits of test case generation from specifications based on FDT(Formal Description Technique), this paper describes a research result on semi-automatic conformance test case generation which is relevant to the industrial application. The reference I/O FSM is shown in Figure 4. For the simplicity, each transition label and state of the reference I/O FSM are abbreviated as shown in Table 1.

Table 1. Abbreviated I/O FSM transitions

Transition	Abbreviation
Request of CTC/InitPolling send, △S=0	A
NAK or △S>M_S/ -	B
Error detected or △R>M_R/NAK	C
InitPolling receive / Error check	D

AllStateData / Timer=0	E
No error detected/AllStateData	F
-/ControlMsgSend, △S=0, Times=0	G
Timer=P_P/Polling send, △S=0, Times=0	H
ACK receive/ -	I
No error detected/Ack or State	J
Control Msg or Polling receive/Error check	K
Timer=P_C/Master Clock	L
Train move/Train Number Msg	M
NAK or △S>M_S/Times+1	N
NAK receive or 0<Times<3/Retransmit, △S=0	O
Times>=3/ALARM	P
Polling receive/-	Q
- /Polling send	R
- /AllState of EIS	S

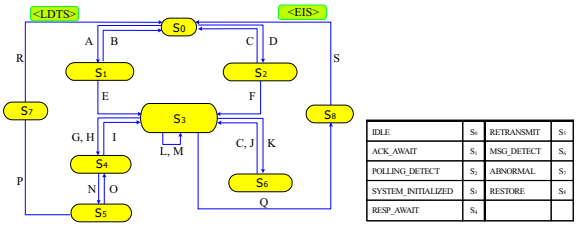


Figure 10. The abbreviated I/O FSM corresponding to the I/OFSM of Figure 4

We have adapted a procedure developed by Dabhura for UIO sequence generation [6]. In fact, UIO sequences are used to test the resulting state after any transition of the I/O FSM.

Table 2. UIO sequences for the I/O FSM in Figure 10

State	UIO Sequence for state
IDLE(S_0)	Request of CTC / Initpolling.Send, △S=0 (A)
ACK_AWAIT(S_1)	NAK or △S>M_S/ - (B)
POLLING_DETECT(S_2)	No error detected / AllStateData (F)
SYSTEM_INITIALIZED(S_3)	-/ControlMsgSend, △S=0, Times=0 (G)
RESP_AWAIT(S_4)	ACK Receive/ - (I)
RETRANSMIT(S_5)	NAKReceive or 0<Times<3/Retransmit, △S=0 (O)
MSG_DETECT(S_6)	No error detected/Ack or State (J)
ABNORMAL(S_7)	- /PollingSend (R)
RESTORE(S_8)	- /AllState of EIS (S)

The following test cases which are generated by the formula R_i•shortest-path(S_0-S_i) •T_{ij}@UIO(S_j), correspond to a resulting test sequence for all TUTs of the I/O FSM in Figure 10. The generated test cases are listed in Table 3.

Table 3. Test cases for the I/O FSM in Figure 10

TUT		Test case for TUT	TUT	Test case for TUT
A		Ri · A · B	J	Ri · A · E · K · J · G
B		Ri · A · B · A	K	Ri · A · E · K · J
C	(S_2→S_0)	Ri · D · C · A	L	Ri · A · E · L · G
	(S_6→S_3)	Ri · A · E · K · C · G	M	Ri · A · E · M · G
D		Ri · D · F	N	Ri · A · E · H · N · O
E		Ri · A · E · G	O	Ri · A · E · H · N · O · I
F		Ri · D · F · G	P	Ri · A · E · H · N · P · R
G		Ri · A · E · G · I	Q	Ri · A · E · Q · S
H		Ri · A · E · H · I	R	Ri · A · E · H · N · P · R · A
I		Ri · A · E · H · I · G	S	Ri · A · E · Q · S · A

These test cases can be used to verify the conformation relation. For example, we apply the input sequence of each test case for TUT to the input PCO port of I/O FSM_I, observe all sequences TS_a of actual outputs at the output PCO port of I/O FSM_I, and finally compare TS_a with TS_o to determine the conformance of I/O FSM_I to I/O FSM_s.

5. CONCLUSIONS

The proposed model checking for railway signaling protocols enables to formally verify whether properties such as safety and liveness expressed in modal logic are true on specifications. This model checking using modal mu-calculus is well suited to computer-aided verification [7]. On the other hand, with the merits of test case generation from specifications based on FSM, we have applied the obtained test cases to the Korean railway signaling systems in the real environment and approved that the result is very efficient to cover bugs in the implementation.

The above functions are implemented by C++ language and included within RSPVTE(Railway Signaling Protocol Verification and Testing Environment) in the M/S window environment. The whole process is shown in Figure 11.

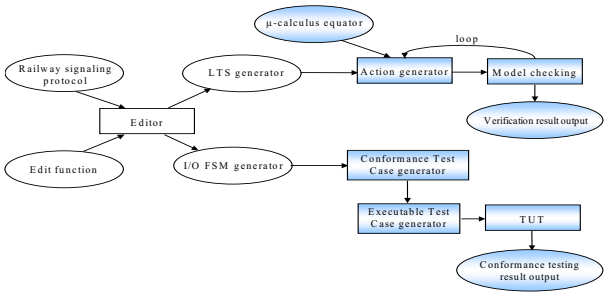


Figure 11. Components of the integrated environment

In the future, we have a plan to apply these techniques to more complex railway protocols in a real environment of railway signaling systems.

REFERENCES

[1] B. Sarikaya, "Principles of Protocol Engineering and Conformance Testing", Ellis Horwood, 1992.

[2] R. Milner, Communication and Concurrency, Prentice Hall International, 1989.

[3] D. Kozen, "Results on the Propositional μ-Calculus", Theoretical Computer Science 27, pp. 333-354, 1983.

[4] R. Cleaveland, B. Steffen, "A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus", Formal Methods in System Design, Feb. 1993.

[5] J.L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron, "Verification in Xesar of the Sliding Window Protocol", In Proceedings of the Seventh IFIP Symposium on Protocol Specification, Testing and Verification VII, pp. 235-248, 1987.

[6] K. K. Sabnani and A. T. Dahbura, "A Protocol Test Generation Procedure", Computer Networks and ISDN Systems, Vol. 15, No. 4, 1998.

[7] J. Jung, Y. Kim, Y. Chung, C. Jeong, S. Kim, "A Study on Implementation of Model Checking Tool for Verifying LTS Specifications", ICOIN98, Tokyo, Jan. 1998.