DISCRETE EVENT MODEL VERIFICATION USING SYSTEM MORPHISM

KI JUNG HONG and TAG GON KIM

Dept of Electrical Engineering & Computer Science,
Korea Advanced Institute of Science and Technology,
Taejon, KOREA

Abstract

Model verification is to check the correctness of the model implementation comparing with a model specification. The specification is modeling and description properties of a real system. The implementation is the ready to simulation model described from model specification. Our proposed frame work is system morphism based approach, which has various levels of morphism depending on system description. I/O function morphism among various morphism levels is chosen to apply directly the implementation. We shows examples to prove the correctness of I/O function morphism based approach.

Keywords

Automatic model verification, DEVS, System morphism, Discrete event system.

1 Introduction

Simulation model has been more complicated as real system continues to grow complex. It becomes important for the credibility of large complicated model to get more accurate result. So, automatic model verification is highly desirable [12].

Discrete event system has three entities such as real system, model and simulator as like figure 1. There exists validation relation between a model and a real system, which checks whether the model, describing the real system, is valid or not. And there exists verification relation between a model and a simulator. The model is called as specification, and the simulator is called as implementation. The model verification checks whether the implementation run as intended depending on available the specification [2]. The description method of the implementation is using simulation language or general purpose language such as C++, Java and etc. The description method of the specification is using informal modeling or formal modeling. Informal modeling is based on three worldviews of discrete event model, and formal modeling is based on DEVS, CCS, Timed Petri-net and etc.

In this paper, we propose timed state reachability graph(TSRG) with system morphism. The specification is described by using DEVS intermediate format(DEVSIF).

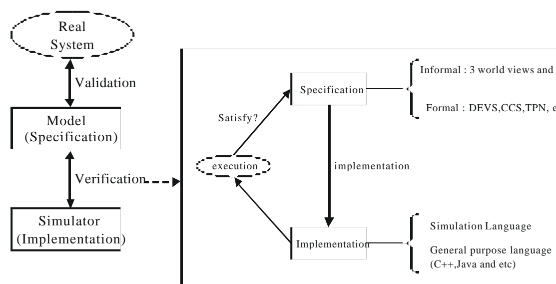

Fig. 1: Three entities of discrete event system

2 Related work

Formal model check and software test are related work. The formal model check has two kinds of approach such as single language approach and dual language approach. Single language approach is that specification and implementation descriptions are single formal modeling language such DEVS, Timed Petri-net, CCS and etc. Dual language approach is that specification description is assertional description by using temporal logic or other logic, and implementation description is behavioral description by using formal language, i.e. CCS and other process algebras. The advantage of these approaches has full coverage of model check. On the other side, the disadvantage of these approaches cannot directly apply to check the correctness of a real implementation [6][7][9][10]. Software test has three kinds of approach in viewpoint of major categories such as black-box test, white-box test and gray-box test [8]. In black-box approach, also known as functional testing, which is the only information used to develop test cases is the specification. That is the implementation is treated as a black box. The test steps are as follows: first, identify the functions the software is expected to perform, second, develop test data to check whether the functions are is expected to perform, And lastly, rely on an oracle to determine the correct response to the test data. Black-box test can directly apply to real implementation but disadvantage to function testing are that many of the

Copyright © 2001

test cases may not confirm the full coverage of implementation. In white-box testing, also known as structural testing, the test steps are as follows: first, determine a testing strategy or coverage goal and second, construct test cases to implement the strategy. Advantage is test case redundancy is minimized, but disadvantage is the same as white-box test. Gray-box is mixed with white-box and black-box to get more efficient test result.

3 Proposed Approach

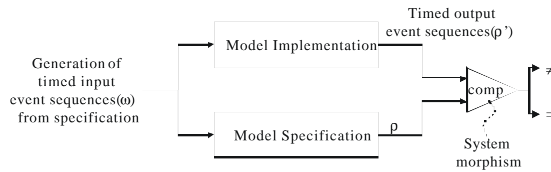

Fig. 2:How to check the correctness of implementation through specification

The objective of the proposed approach is to check the correctness of implementation by using all possible input/output events sequences with timed behavior through specification. Implementation. If a system has the requirement of all input and output events sequences with timed behavior, the proposed approach checks to satisfy this requirement in the system, called as implementation. Output event sequences with timed behavior, called as ρ, is compared by system morphism, which can solve the coverage problem for given all possible ω/ρ. And this approach supports to check whether the implementation satisfies the specification on execution, or not. The implementation is build by any language, but the specification is described by DEVS formalism in this approach.

4 System morphism

Figure 3 shows that system morphism can apply a $system_A$ and a $system_B$. If there exists system morphism from the $system_A$ to the $system_B$, there exist morphism functions g and k to satisfy $f_A(i) = k(f_B(g(i)))$ for $\forall i \in I_A$. And conversely, if there exists system morphism from the $system_B$ to the $system_A$, there exist morphism function g' and k' to satisfy $f_B(i') = k'(f_A(g'(i')))$ for $\forall i' \in I_B$. This property of system morphism is called as system relation preservation [2].

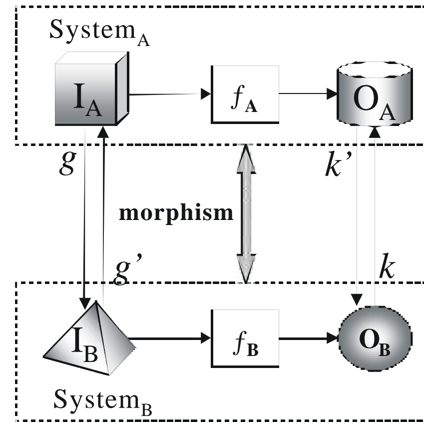

Fig. 3:System morphism diagram

System morphism has various levels of system description for a model, which contain internal information quantities depending on description levels. If a networked model description contains structured information of the model and a system model description contains internal state transition information of the model without structure information, the networked description has more detail internal information than the system description. In case of what system morphism applies between implementation and specification, the internal information in a model description is restricted such as input event, output event, next event scheduling time, initial state and the number of states of implementation. So, we choose I/O function morphism in various system morphism levels. I/O function system is defined as:

$$IOF = < T, X, Y, \Omega, F >$$

T :Time base

X :Input events set

Y: Output events set

Ω: Timed input event segments set

$F: \Omega \rightarrow \{\boldsymbol{r}\} : (x, t_1) \rightarrow (y, t_2)$

$: x \in X, y \in Y, t_1, t_2 \in T$

:I/O function

$\boldsymbol{r}$:timed output event segment

And I/O function morphism from specification to implementation is there exists morphism function g and k to satisfy $F_{impl}(\omega) = k(F_{spec}(g(\omega)))$ for $\forall \omega \in \Omega_{spec}$.

5 DEVS Formalism : A Brief Introduction

Copyright © 2001

The DEVS formalism specifies a model in a hierarchical, modular form. A discrete event system consists of entities whose dynamics are described as a set of procedure rules. Such rules control the interactions among the communicating entities. The communicating entities and the procedure rules can be decomposed into the smaller ones with the modeling semantics. These decomposed components are directly mapped to the atomic models, from which larger ones are built. A basic model, called an atomic model, is not further decomposed specification of the dynamic behavior of a component. Formally, an atomic model AM is specified as [1]:

$$AM = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X:Input events set

S: Sequential states set

Y: Output events set

$\delta_{int}: S \rightarrow S$: internal transition function

$\delta_{ext}: Q \times X \rightarrow S$:external transition function

$\lambda : S \rightarrow Y$:output function

$ta : S \rightarrow \Re^{+}_{(0,\infty)}$

$Q : \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$

:total state of AM

The second form of the DEVS model, called a coupled model(or coupled DEVS), is a specification of the hierarchical model structure. It describes how to couple component models together to form a new model. This new model can be employed as another component in a larger coupled model, thereby giving rise to the construction of complex models in a hierarchical fashion. Formally, a coupled model is defined as[1] :

$$DN = < X, Y, \{M_i\}, EIC, EOC, IC, SELECT >$$

X :Input events set

Y: Output events set

$\{M_i\}$:DEVS components set

$EIC \subseteq X \times \bigcup_i X_i$:external input coupling relation

$EOC \subseteq \bigcup_i Y_i \times Y \rightarrow S$:external output coupling relation

$IC \subseteq \bigcup_i Y_i \times \bigcup_i X_i$:internal coupling relation

$SELECT : 2^{\{M_i\}} - \varnothing \rightarrow \{M_i\}$:tie breaking selector

A detailed discussion about the DEVS formalism and modeling is found in [1].

6 Timed state rechability graph

Timed state reachability graph(TSRG) is all ω/ρ pairs generation tools. TSRG is defined as follows:

$$TSRG = < X, Y, N, T, E >$$

X:Input events set

Y: Output events set

N: States set : Node

$T: N \rightarrow \Re^{+}_{(0,\infty)} \times \Re^{+}_{(0,\infty)}$:Time function

$E: N \times (X \cup Y) \times Boolean \times N$: Edge

Boolean : CONTINUE

And TSRG has following definitions:

Definition 1 : Equivalent Node : $s1 \equiv s2$ is $T(s1) = T(s2) \wedge \forall ev \in X \cup Y, \forall (s1, ev, c_1, s_{12}), (s2, ev, c_2, s_{22}) \in E, c_1 = c_2$.

Definition 2 Minimized TSRG is $\forall s_1, s_2 \in N, \neg(s_1 \equiv s_2)$.

Definition 3 Reachable $s_1, s_2 \in N$, $s_1 \rightarrow_r s_2$ is $((s_1,_,_,s_2) \in E) \vee (((s_1,_,_,s_3) \in E) \wedge ((s_4,_,_,s_2) \in E) \wedge (s_3 \rightarrow_r s_4))$.

Definition 4 Loop at s_1, $\Xi(s_1)$ is $s_1, s_2 \in N$, $\exists e_1 = (s_1,_,_,s_2) \in E \wedge s_2 \rightarrow_r s_1$.

Definition 5 I/O Sequences from s_0, $\Gamma(s_0)$ is $\Gamma(s_0, \nu = \{s_0\}) = \{(e_1, T(s_0)) + \Gamma(s_0, \nu \cup \{s_1\}) | s_0, s_2 \in N, \forall e_1 = (s_1,_,_,s_2) \in E \wedge s_1 \notin \nu\} \cup \{(e_1, T(s_0)) | s_0, s_2 \in N, \forall e_1 = (s_1,_,_,s_2) \in E \wedge s_1 \in \nu\}$

$(e_1, T(s_0)) + \{(e_2, T(s_1)), (e_3, T(s_1))\} = \{(e_1, T(s_0)) (e_2, T(s_1)), (e_1, T(s_0)) (e_3, T(s_1))\}$

Theorem 1 : Coverage of ω/ρ in specification : $\Gamma(s_0)$ has all edges of SRG if and only if $\forall s_1 \in N$, $s_0 \rightarrow_r s_1$.

Proof: by definition of $\Gamma(s_0)$.

This theorem means that if any node from s_0 in TSRG is reachable, $\Gamma(s_0)$ has all edges in TSRG.

Specification and implementation has next following preconditions:

- Deterministic model implementation and specification.
- Message lossless model.
- TSRG from specification is $\forall s_1, s_2 \in N$, $s_1 \rightarrow_r s_2$ and minimized. It means specification is deadlock free and livelock free.

Theorem 2 : Number of states : If TSRG from specification is minimized and M_{impl} accepts M_{spec}, $|S_{impl}| \ge |S_{spec}|$.

Proof: If $|S_{impl}| < |S_{spec}|$, it means that TSRG from specification has equivalence state. It doesn't satisfy that TSRG from specification is minimized. So, it results in $|S_{impl}| \ge |S_{spec}|$.

If Implementation accepts specification and the number of states of implementation is great than or equal to specification's one, implementation has additional equivalence states or hidden states comparing to the states of specification. Hidden state

Copyright © 2001

is defined as figure 4. When the original state s_i of specification has time range $[0,t_i]$, the summation of time ranges of hidden states of implementation should be equal to $[0,t_i]$ and those hidden states' transition attributes should be equal to the original's one.

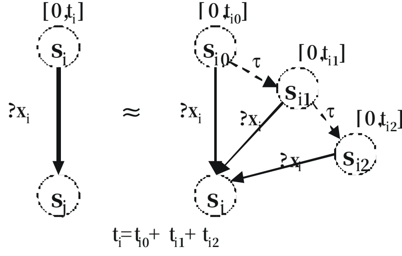

Fig. 4:Hidden State

And equivalence state is defined as figure 5. The equivalence state's definition is the same as equivalence node's definition of TSRG.

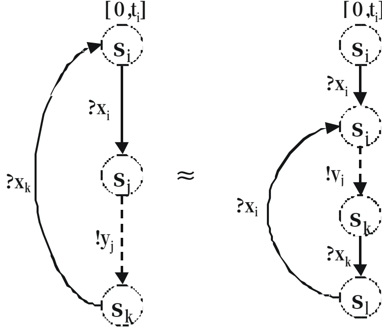

Fig. 5:Equivalence State

Theorem 3: Number of states: If M_{impl} is minimized and accepts M_{spec}, $|S_{impl}|-|S_{spec}|$ = |hidden states|

Theorem 4:Existence of loop: If $\forall s_1,s_2 \in N$, $s_1 \rightarrow s_2$, , $\Xi(s)$ for $\forall \gamma \in \Gamma(s_0)$, $(s,T(s)) \in \gamma$.

Proof: By definition of $\Gamma(s_0)$.

Theorem 5: Coverage of minimized implementation: If M_{impl} is minimized, all time i/o sequences from s_0 is $\Gamma(s_0)$.

Theorem 6: Coverage of non-minimized implementation: If M_{impl} is not minimized, n = $|S_{impl}|$ - $|S_{spec}|$ - |hidden states| and m is minimum length of $\Gamma(s_0)$, all timed I/O sequences from s_0 is $\Sigma^{\lceil n/m \rceil+1}\Gamma(s_0)$.

And specification use DEVS for modeling formalism. So, we introduce transformation rules from DEVS model to TSRG as follows:

TSRG.X = AM.X

TSRG.Y = AM.Y

TSRG.N = AM.S

TSRG.T = $\{s \rightarrow (0,ta(s)) \mid \forall s \in S \wedge \boldsymbol{d}_{int}(s) \neq \phi \wedge (\exists x \in X, \boldsymbol{d}_{ext}(s,0,x) \neq \phi)\} \cup \{s \rightarrow (ta(s),ta(s)) \mid \forall s \in S \wedge \boldsymbol{d}_{int}(s) \neq \phi \wedge (\exists x \in X, \boldsymbol{d}_{ext}(s,0,x) = \phi)\} \cup \{s \rightarrow (0,\text{¥}) \mid \forall s \in S \wedge \boldsymbol{d}_{int}(s) = \phi \wedge (\exists x \in X, \boldsymbol{d}_{ext}(s,0,x) \neq \phi)\}$

TSRG.E = $\{(s,\boldsymbol{l}(s),false, \boldsymbol{d}_{int}(s)) \mid \forall s \in S \wedge \boldsymbol{d}_{int}(s) \neq \phi\} \cup \{(s,x,true,\boldsymbol{d}_{ext}(s,0,x)) \mid \forall s \in S \wedge (\forall x \in X, \boldsymbol{d}_{ext}(s,0,x) \neq \phi \wedge e = ta(\boldsymbol{d}_{ext}(s,0,x)) - ta(\boldsymbol{d}_{ext}(s,e,x)))\} \cup \{(s,x,false,\boldsymbol{d}_{ext}(s,0,x)) \mid \forall s \in S \wedge (\forall x \in X, \boldsymbol{d}_{ext}(s,0,x) \neq \phi \wedge 0 = ta(\boldsymbol{d}_{ext}(s,0,x)) - ta(\boldsymbol{d}_{ext}(s,e,x)))\}$

7 Overview of implementation

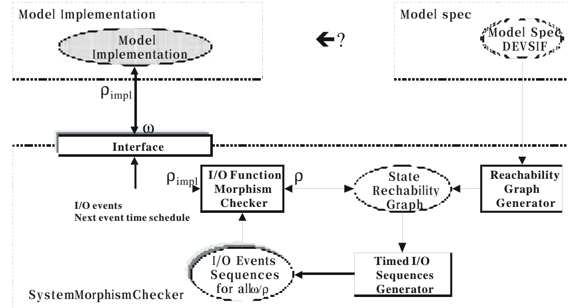

Fig. 6:Overview of Implementation

DEVSIF, called DEVS Intermediate Format, is used to describe specification as DEVS model. This language, DEVSIF, supports all semantics of DEVS formalism. Reachability Graph Generator generates TSRG from DEVSIF. And timed I/O sequences generator find all ω/ρ pairs as using γ. Interface operates with implementation to exchange input/output events and next event time schedule. Next event time schedule notices to solve time coverage of I/O function morphism checker.

Complexity analysis of I/O function morphism checker(IOFMC) has time complexity and space complexity. Time complexity of minimized implementation is $O(|\Gamma(s_0)| \times (1+n-m))$ for n = $|S_{impl}|$ and m = $|S_{spec}|$. Time complexity of non-minimized implementation is $O(|\Gamma(s_0)|^{(1+\lceil k/l \rceil)})$ for l = $|S_{impl}|$-$|S_{spec}|$-|hidden states|, k = minimum length of $\Gamma(s_0)$. Space complexity is O(NE) for N = the number of nodes in TSRG and E = the number of edges in TSRG. In usual, unbounded buffer results in state explosion, because unbounded buffer has infinite

Copyright ã 2001

states. To avoid state explosion, buffer size should be bounded in constant size, generally, one.

IOFMC's actual operation is following sequences. First, IOFMC tries to find hidden states in implementation by using all ω/ρ pairs, called as γ. If IOFMC find the candidate of hidden state in implementation, implementation's next event scheduling time is less than expecting time range of ρ from specification. Implementation should accept the remain time range of specification for γ finding the candidate of hidden state. Otherwise, it means that implementation don't satisfy the requirements of specification. Next, IOFMC tries to check the correctness of implementation through all γ from specification. Through all sequences of IOFMC, if implementation accepts all, it means implementation satisfy the requirements of specification.

World View		DEVS <X, Y, S, δ_{ext}, δ_{int}, ta, λ>
Event-scheduling	Input event	$x \in X$
	Output event	$y = \lambda(s) \in Y$
	Event routine	$\delta_{ext}(s,e,x)$, $\delta_{int}(s)$
	Schedule	ta
Process-interaction	Input, output event	$x \in X$, $y \in Y$
	Begin event	$\delta_{ext}(s,e,x)$
	End event	$\delta_{int}(s)$, $y = \lambda(s) \in Y$
	Interaction	$Z_{i,j} : Y_i \rightarrow X_j$
Object-oriented	Input, output message	$x \in X$, $y \in Y$
	Instance/state variable	S
	Method/state transition	$\delta_{ext}(s,e,x)$, $\delta_{int}(s)$

Table. 1:Transformation ru les from 3 worlds view to DEVS formalism

If original specification is 3 worlds view based informal description, it should be translated into DEVSIF by using table1's rules. 3 worlds view are event-oriented approach, process-oriented approach and object-oriented approach. Event-oriented approach is using event list for scheduling, which is list of event routines associated with times. Process-oriented approach is using process or thread for scheduling, which is entity life cycle. This approach is most close to the real implementation. Object-oriented approach is using object for scheduling unit, which is composed of state and state transition, called as member function.

8 Example of proving the correctness of proposed approach

First example is to show the verification process and results for event oriented and process oriented implementation respectively. Discrete event model verification environment is next following as:

- Worldview: Event-oriented/Process oriented.
- Simulation Engine: C++Sim(public domain)[11]
- Implementation method: C++, $|S_{impl}| = |S_{spec}|$, Event-oriented and Process oriented respectively.
- Specification: DEVSIF of assembly process as figure 7.

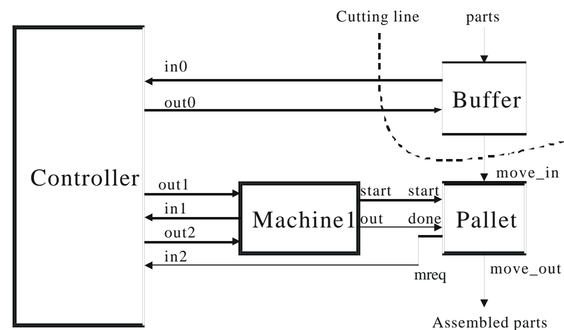

Fig. 7:Example of Assembly process

Assembly process has 3 components, which are controller, machine and pallet. Pallet carries with parts from buffer to machine1 to assemble parts, machine1 gets back parts to pallet after assembling parts by order from controller and controller controls pallet and machine without trouble.

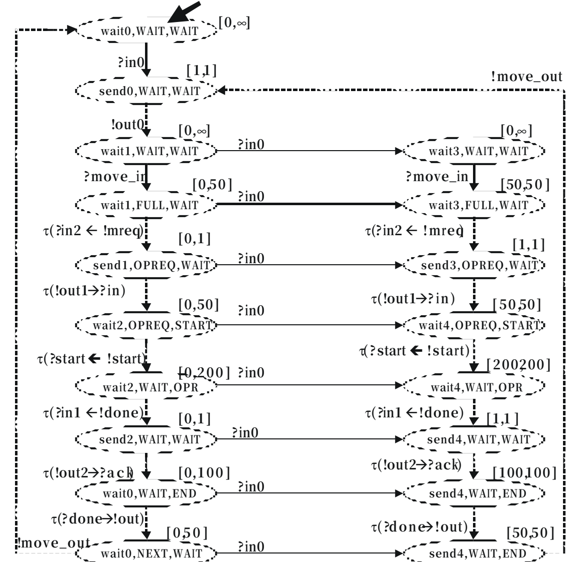

Fig. 8:TSRG of Assembly process

TSRG from specification of assembly process is generated as figure 8. This TSRG have 9 γ-sequences as followings:

1. (?in0(0,¥),!out0(1,1),?move_in(0,50),!move_out (452, 452))*

Copyright ã 2001

2. ?in0(0,∞),(!out0(1,1),?in0(0,0),?move_in(0,50), !move_out(452,452))*
3. ?in0(0,∞),(!out0(1,1),?move_in (0,50),?in0(0,50),!move_out(452,452))*
4. ?in0(0,∞),(!out0(1,1),?move_in (0,50),?in0(50,51), !move_out (402,402))*
5. ?in0(0,∞),(!out0(1,1),?move_in (0,50),?in0(51,101), ! move_out(401,401))*
6. ?in0(0,∞),(!out0(1,1),?move_in (0,50),?in0(101,301), ! move_out(351,351))*
7. ?in0(0,∞),(!out0(1,1),?move_in (0,50),?in0(301,302), ! move_out(151,151))*
8. ?in0(0,∞),(!out0(1,1),?move_in (0,50),?in0(302,402), ! move_out(150,150))*
9. ?in0(0,∞),(!out0(1,1),?move_in (0,50),?in0(402,452), ! move_out(50,50))*

Verification execution time is about 1sec for nine γ-sequences. If the implementation is not minimized, by time complexity analysis, the execution time is $9^{\lceil n/4 \rceil}$ sec for n = the number of equivalence nodes. This complexity indicates the minimization of the implementation is highly desirable. In usual, the implementation, without intention making equivalence nodes, has no equivalence nodes.

Next example is C++ implementation through C++ code generator from SDL. SDL description of implementation is as follows:

```
system AssemblyProcess;
signal
  in0,
  in1,
  in2,
  out0,
  out1,
  out2,
  mstart,
  done,
  move_in,
  move_out;
channel C1
  from env to Controller with in0;
  from Controller to env with out0;
endchannel C1;
channel C2
  from Controller to Machine1 with out1,out2;
  from Machine1 to Controller with in1;
endchannel C2;
…block Controller referenced;
block Machine1 referenced;
block Pallet1 referenced;
endsystem AssemblyProcess;

process PPallet;
timer  TPoll;
start;
   nextstate WAIT;
state WAIT;
  input done;
           set(now+50,TPoll);
   nextstate NEXT;
  input move_in;
           set(now+50,TPoll);
   nextstate FULL;
endstate;
state NEXT;
  input TPoll;
           output move_out;
   nextstate WAIT;
endstate;
state FULL;
  input TPoll;
           output in2;
   nextstate OPREQ;
endstate;
…endprocess PPallet;
```

This SDL model's first part indicates structured information of the implementation and the second parts is processes, which describe the I/O behavior of the implementation. This SDL model has originally no information about the time unit of a timer, but in this model, we assume that the time unit of the timer is the same as real model's one. Results of this example are the same as first example's one.

9 Conclusions

To solve the coverage problem of model verification, we introduce system morphism based automatic model verification methodology, which uses IOFM. IOFM is appropriate methodology for the check of the correctness of the implementation for a specification and requires to find all possible I/O sequences. So, TSRG is introduced to generate all possible I/O sequences. In analysis of IOFM complexity, the time complexity is reasonable for well implemented model, and the space complexity is depending on there exists the state explosion of the specification or not. To avoid the state explosion, buffer or queue should have a bounded size such as one or two.

In future works, we try to apply other more complicated example for proving practicality. If specification uses other modeling formalism, such as Timed Petri-net, CCS and etc, but not DEVS, a translator is needed to translate other formalism into DEVSIF.

References

1. B.P. Zeigler, *Multifacetted Modelling and Discrete-Event Simulation*, Orlando, FL, Academic Press, New York, 1984.
2. B.P. Zeigler, H. Praehofer and T.G. Kim, *Theory of Modeling and Simulation, 2nd ed*, Academic Press, 2000.
3. G.P. Hong and T.G. Kim, *A framework for verifying discrete event models within a devs-based system development methodology*, Trans of the S.C.S International, vol.13, pp. 19-34, 1996.

4. T.G. Kim, *DEVSIM++ Users Manual*, SMSLab Dept. of EE., KAIST, 1994, http://sim.kaist.ac.kr/.
5. R. Milner, *Communication and Concurrency*, Prentices Hall, 1989.
6. S.D.Brookes, C.A.R. Hoare and A.D. Roscoe, *A Theory of Communicating Sequential Processes*, Journal of the ACM, Vol.31, No.3, 1984.
7. J.L. Peterson, *Petri net theory and the modeling of systems*, Prentice Hall, 1981.
8. J.E. Heiser, *An Overview of Software Testing*, AUTOTESTCON,97, IEEE Autotestcon Proceedings, p204-211, 1997.
9. K.L.McMillan, *The SMV system manual*, Carnegie-Mellon University, 1992.
10. F. Moller and A. Rabinovich, *On the expressive power of CTL*, Logic in Computer Science, 1999. Proceedings. 14th Symposium on, 1999.
11. M.C. Little and D.L. McCue, *Construction and Use of a Simulation Package in C++*, Dept of CS, Univ of Newcastle upon Tyne, 1998, ftp://arjuna.ncl.ac.uk.
12. J. Banks, D. Gerstein and S.P. Searles, *Modeling process, validation, and verification of complex simulations: A survey*, S.C.S Methodology and Validation, simulation series vol. 19, No 1,pp13-18,1988.

Copyright ã 2001