A Modular Verification Framework Based on Finite & Deterministic DEVS

Moon Ho Hwang and Bernard P. Zeigler

Arizona Center for Integrative Modeling and Simulation,
Electrical and Computer Engineering Department,
The University of Arizona, Tucson, AZ 85721, USA
{mhhwang, zeigler}@ece.arizona.edu

Keywords: Discrete Event System Specification (DEVS), Finite and Deterministic DEVS, Modular Verification Framework, Illegal Behavior, Legal Behavior;

Abstract

In order to check if the system behavior satisfies the requirement set, this paper uses a class of DEVS, called finite & deterministic DEVS (FD-DEVS). Since the infinite state behavior of FD-DEVS networks can be abstracted as a finite reachable graph, this paper utilizes the reachable graph structure to check the emptiness of illegal behavior detected by checkers, called rejectors, as well as the non-emptiness of legal behavior generated by components under testing.

I. Introduction

Hierarchical and modular system modeling functionality has been strongly demanded [8] so Discrete Event System Specification (DEVS) has been evolved from a non-hierarchical formalism [13] to a hierarchical and modular one [14]. This formalism has been intensively researched from the view point of simulation and execution over past 30 years [15]. In spite of the advantage of DEVS's hierarchical and modular modeling capability, the verification method based on DEVS has been successful only recently, by abstracting the infinite state behavior of DEVS [6].

In order to check if the system behavior satisfies the requirement set, this paper uses a class of DEVS, called finite & deterministic DEVS. Since the infinite state behavior of FD-DEVS networks can be abstracted as a finite reachable graph[7], this paper utilizes the reachable graph structure to check the emptiness of illegal behavior detected by checkers, called rejectors, as well as the non-emptiness of legal behavior generated by components under testing.

This paper is organized as follows. Section II defines the time event segment and its concatenation operation. The formal definition of FD-DEVS is given from the view points of the atomic model as well as the coupled model in Section III. Section IV introduces the language of FD-DEVS as a set of event segments to reach an acceptance state. Thus we will check if the illegal langauge is empty as well as the legal language is not empty. Section V illustrates a verification example using an intersection traffic light control. Section VI summarizes contributions of our proposed framework. Finally, conclusions and further research directions are given in Section VII.

II. Timed Event Segment

Given an arbitrary event set Z and a time based $\mathbb{T} = \mathbb{R}_{[0,\infty]} \stackrel{def}{=} \{t | 0 \leq t \leq \infty\}$, a *timed event* is defined as a pair of $(t, \bar{a})$ such that $t \in \mathbb{T}$ and $\bar{a} \in Z^*$ is an event string, where Z^* is the *Kleene closure* of Z [5]. The Kleene closure of an event set Z is a set of all finite length of strings over Z, for example $Z = \{a, b\}$, then $Z^* = \{\epsilon, a, b, aa, ab, bb, aab, aba, \ldots\}$ where ϵ denotes the *empty string* or the *nonevent*. Therefore, an *event trajectory* $\omega : \mathbb{T} \rightarrow Z^*$ is used for describing all timed events over the time horizon from 0 to ∞. For example, the event trajectory $\omega = (t_1, b)(t_2, aab)$ is $\omega(t) = b$ for $t = t_1$; $\omega(t) = aab$ for $t = t_2$; $\omega(t) = \epsilon$ otherwise. If the number of events in an event trajectory ω is denoted by $|\omega|$, for $\omega = (t_1, b)(t_2, aab)$, $|\omega|$=4. Obviously, $|\epsilon|$=0.

We define the *all possible events over* Z at all times as $\Omega_Z = \mathbb{T} \times Z^*$. An *event segment* can be defined from a time interval $[t_l, t_u]$ [1] and the *domain function*, dom is a map from an event segment to its observation interval. An event segment in $\Omega_{Z[t_l,t_u]}$ can be written as $\omega_{[t_l,t_u]}$ or $\omega : [t_l, t_u] \rightarrow Z^*$ such that $dom(\omega_{[t_l,t_u]}) = [t_l, t_u]$. Sometimes we omit the observation interval such as $\omega \in \Omega_Z$ if $dom(\omega) = [0, \infty]$ or $dom(\omega)$ is not important.

Given two time events (t, ab) and (t, ba) at a time t, the concatenation of (t, ab) and (t, ba) is $(t, ab)(t, ba)$ and it can be also written as $(t, abba)$. A pair of con-

[1] For simplicity, we focus on the closed boundary interval only.

 ISBN 1-56555-304-7

tiguous segments $\omega_1 \in \Omega_{Z[t_1,t_2]}$ and $\omega_2 \in \Omega_{Z[t_2,t_3]}$, the concatenation of ω_1 and ω_2 is defined as a form of $\omega_1 \cdot \omega_2 : [t_1, t_3] \rightarrow Z^*$ such that $\omega_1 \cdot \omega_2(t) = \omega_1(t)$ for $t \in [t_1, t_2)$; $\omega_1(t) \cdot \omega_2(t)$ for $t = t_2$; $\omega_2(t)$ for $t \in (t_2, t_3]$; If there is no confusion, we will omit '$\cdot$' so $\omega_1\omega_2$ is the same as $\omega_1 \cdot \omega_2$.

The *empty segment* within $[t_l, t_u]$, denoted by $\epsilon_{[t_l,t_u]}$, is that $\omega(t) = \epsilon$ for $t \in [t_l, t_u]$. Since ϵ is the identity of concatenation of events [5], i.e., $\epsilon \cdot z = z$ where $z \in Z$, if $\omega = \epsilon_{[t_l,t]}(t, z)\epsilon_{[t,t_u]}$ where $t \in dom(\omega) = [t_l, t_u]$ and $z \in Z$, it can be simply written as $\omega = (t, z)$.

III. FD-DEVS

A. Atomic FD-DEVS

1) Definition of Atomic FD-DEVS: An *atomic FD-DEVS* is a 9-tuple,

$$M =< X, Y, S, s_0, \tau, \delta_x, \rho, \delta_\tau, \lambda >$$

where,

- X and Y are *finite sets of input and output events*, respectively such that $X \cap Y = \varnothing$.
- S is a *non-empty and finite states set*.
- $s_0 \in S$ is the *initial state*.
- $\tau : S \rightarrow \mathbb{Q}_{[0,\infty]}$ is the *time advance function* where $\mathbb{Q}_{[0,\infty]}$ denotes a set of non-negative rational numbers with infinity.
- $\delta_x : S \times X \rightarrow S$ is the *external transition function*.
- $\rho : S \times X \rightarrow \mathbb{B}$ is the *reschedule-indicating function* that returns 1 when a reschedule is needed; otherwise, returns 0.
- $\delta_\tau : S \rightarrow S$ is the *internal transition function*.
- $\lambda : S \rightarrow Y \cup \{\epsilon\}$ is the *internal output function*. □

Notice that δ_x δ_τ and λ are *partial functions* that can be defined for some elements in the domain.[2]

2) State Transition of Atomic FD-DEVS: Given $M =< X, Y, S, s_0, \tau, \delta_x, \rho, \delta_\tau, \lambda >$, the *total states set* considers the schedule t_s and its elapsed time e as well as state s such that

$$Q = \{(s, t_s, e) | s \in S, t_s \in \mathbb{Q}_{[0,\infty]}, 0 \leq e \leq t_s\}$$

From the total state set Q and the *total event set* $Z = X \cup Y \cup \{\epsilon\}$, the *total state transition function* $\delta : Q \times Z \rightarrow Q$ maps to other total state. For $(s, t_s, e) \in Q$ where $t_s \in \mathbb{Q}_{[0,\infty]}$ and $z \in Z$,

$$\delta((s, t_s, e), z) = (s', t'_s, e')$$

where

[2]In this paper $f(\ldots) \vdash$ denotes that $f(\ldots)$ is defined for any partial function $f(\ldots)$. For example, $\delta_x(s, x) \vdash$ means that $\delta_x(s, x)$ is defined.

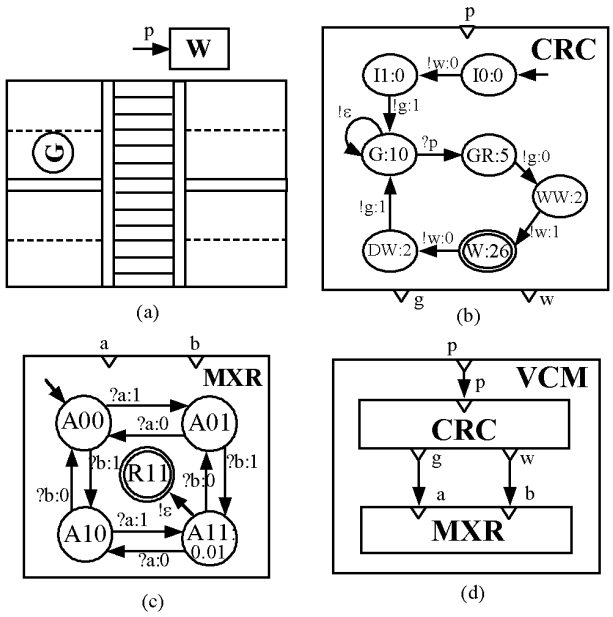

Fig. 1. (a) Lights Configuration (b) Controller FD-DEVS (c) Mutual Exclusion Rejector (d) The Verification Frame

[*External Transition*] For $z \in X$, $(s', t'_s, e') =$

$$\begin{cases} (\delta_x(s, x), \tau(\delta_x(s, x)), 0) & \text{for } \delta_x(s, z) \vdash, \rho(s, z) = 1 \\ (\delta_x(s, x), t_s, e) & \text{for } \delta_x(s, z) \vdash, \rho(s, z) = 0 \\ (s, t_s, e) & \text{otherwise} \end{cases}$$

[*Internal Transition*] For $z \in Y \cup \{\epsilon\}$, $(s', t'_s, e') =$

$$\begin{cases} (\delta_\tau(s), \tau(\delta_\tau(s)), 0) & \text{for } z = \lambda(s), e = t_s \\ \text{undefined} & \text{otherwise} \end{cases}$$

3) Execution of Atomic FD-DEVS: An *execution* of an atomic FD-DEVS M is a sequence of state changes from $q \in Q$ according to a sequence of I/O events. Formally, an execution is defined as $\Delta : Q \times \Omega_Z \rightarrow Q$: For $q = (s, t_s, e) \in Q$,$\omega, \omega' \in \Omega_{Z[0,t]}$, $t \in \mathbb{T}$ and $z \in Z^\epsilon$

$\Delta((s, t_s, e), \omega) =$

$$\begin{cases} \delta((s, t_s, e + t), z) & \text{for } \omega = \epsilon_{[0,t]}(t, z), \delta_1 \vdash \\ \delta(\Delta(q, \omega'_{[0,t]}), z) & \text{for } \omega = \omega'_{[0,t]}(t, z), \delta_2 \vdash \\ \text{undefined} & \text{otherwise} \end{cases}$$

where δ_1 and δ_2 means $\delta((s, t_s, e + t), z)$ and $\delta(\Delta(q, \omega'_{[0,t]}), z)$, respectively.

Example 1: Let's consider a cross road system in which there are two lights: Ⓖ for green and Ⓦ for walk, as shown in Figure 1(a). Figure 1(b) shows an atomic FD-DEVS, `CRC` that controls Ⓖ and Ⓦ lights. The formal model of `CRC` is: X={`p`}; Y={`g:0`, `g:1`, `w:0`, `w:1` } where ports `g` and `w` stand for

"green" and "walk", while values 0 and 1 for "off" and "on", respectively; S={I0,I1,G,GR,WW,W,DW } where I0 and I1 are initializing states, G stands for "green on", GR for "green to red",WW for "waiting for walk on", W for "walk on", DW for "don't walk"; τ(I0)=τ(I1)=0, τ(G)=10, τ(GR)=5, τ(WW)=2, τ(W)=26, τ(DW)=2; δ_τ(I0)=I1, δ_τ(I1)=G, δ_τ(G)=G, δ_τ(GR)=WW, δ_τ(WW)=W, δ_τ(W)=DW, δ_τ(DW)=G; λ(I0)=w:0, λ(I1)=g:1, λ(G)=ϵ, λ(GR)=g:0, λ(WW)=w:1, λ(W)=w:0, λ(DW)=g:1; δ_x(G,p)=GR; ρ_x(G,p)=1; [3] For $\omega_{[0,44.3]}$ = (0,w:0)(0,g:1)(27,p)(32,g:0)(34,w:1) $\epsilon_{[34,44.3]}$, $\Delta((\text{I0},0),\omega)$=(W,26,5.7). □

B. Coupled FD-DEVS

1) Definition of coupled FD-DEVS: A *coupled FD-DEVS* is a 6-tuple,

$$N =< X, Y, D, C_{xx}, C_{yx}, C_{yy} >$$

where

- X and Y are *finite sets of input and output events*, respectively such that $X \cap Y = \varnothing$.
- $D = \{M_i\}$ is the finite set of sub-component FD-DEVSs that are atomic FD-DEVSs. [4]
- $C_{xx} \subseteq X \times \bigcup_{M_i \in D} X_i$ (res. $C_{yx} \subseteq \bigcup_{M_i \in D} Y_i \times \bigcup_{M_i \in D} X_i$ and $C_{yy} = \bigcup_{M_i \in D} Y_i \rightarrow Y \cup \{\epsilon\}$) is the *external input (internal and external output) coupling relation.* ■

For example, VCM shown in Figure 1(d) is a coupled FD-DEVS VCM=$< X, Y, D, C_{xx}, C_{yx}, C_{yy} >$ where X={ p }; Y=$\varnothing$; D={CRC, MXR }; C_{xx}={ (VCM.p, CRC.p)}; C_{yx}={ (CRC.g, MXR.a), (CRC.w, MXR.b)}; C_{yy}=$\varnothing$;

2) State Transition of Coupled FD-DEVS: The *total state set* of N is defined as the combination of sub-components' total states such that

$$Q = \{(\ldots,(s_i, t_{si}, e_i),\ldots)|(s_i, t_{si}, e_i) \in Q_i, M_i \in D\}$$

We consider its state can change with a triggering event $z \in Z = X \bigcup_{M_i \in D} Y_i \cup \{\epsilon\}$. Thus the *state transition function* $\delta : Q \times Z \rightarrow Q$

$$\delta((\ldots,(s_i, t_{si}, e_i),\ldots), z) = (\ldots,(s'_i, t'_{si}, e'_i),\ldots)$$

can be categorized into two transitions according to the triggering events:

[*External Transition Triggering*] For $z \in X$,

$$(s'_i, t'_{si}, e'_i) = \begin{cases} \delta_i((s_i, t_{si}, e_i), x_i) & \text{for } (z, x_i) \in C_{xx} \\ (s_i, t_{si}, e_i) & \text{otherwise} \end{cases}$$

[3]To distinguish the internal transition from the external transition, we use ! in front of an output event, ? for an input event for each transition in all state transition diagrams of this paper.

[4]This restriction of only atomic FD-DEVS for sub-components is for the simple explanation. For analysis of hierarchical FD-DEVS networks, we first flatten them, then apply this explanation.

[*Internal Transition Triggering*] For $z \in \bigcup_{M_i \in D} Y_i \cup \{\epsilon\}$ and $\lambda_{i^*}(s_{i^*}) = z$,

$$(s'_i, t'_{si}, e'_i) = \begin{cases} \delta_i((s_i, t_{si}, t_{si}), z) & \text{for } M_i = M_{i^*} \\ \delta_i((s_i, t_{si}, e_i), x_i) & \text{for } (z, x_i) \in C_{yx} \\ (s_i, t_{si}, e_i) & \text{otherwise} \end{cases}$$

However, if $z \in \bigcup_{M_i \in D} Y_i \cup \{\epsilon\}$, $e \neq t_{si^*}$ or $\lambda_{i^*}(s_{i^*}) \neq z$, then (s'_i, t'_{si}, e'_i) is undefined because $\delta_i((s_i, t_{si}, e_i), z)$ is undefined.

3) Execution of Coupled FD-DEVS: Similar to the atomic FD-DEVS, an *execution* of a coupled FD-DEVS N from $q \in Q$ with $\omega \in \Omega$ is defined as $\Delta : Q \times \Omega_Z \rightarrow Q$: For $q = (\ldots, q_i, \ldots)$ where $q_i = (s_i, t_{si}, e_i) \in Q_i$, $z \in Z$ and $\omega, \omega' \in \Omega_{Z[0,t]}$, $\Delta(q, \omega) =$

$$\begin{cases} \delta((\ldots,(s_i, t_{si}, e_i + t),\ldots), z) & \text{for } \omega = \epsilon_{[0,t]}(t, z), \delta_1 \vdash \\ \delta((\ldots, \Delta_i(q_i, \omega'_{[0,t]}),\ldots), z) & \text{for } \omega = \omega'_{[0,t]}(t, z), \delta_2 \vdash \\ \text{undefined} & \text{otherwise} \end{cases}$$

where δ_1 and δ_2 means $\delta((\ldots,(s_i, t_{si}, e_i+t),\ldots), z)$ and $\delta((\ldots, \Delta_i(q_i, \omega'_{[0,t]}),\ldots), z)$, respectively.

IV. Verification Framework

This paper focuses our interest on a segment $\omega \in \Omega_Z$ such that $|\omega| = \infty$ and $dom(\omega) = [0, \infty]$ because our interesting system is a *live* system that works forever. For ω s.t. $|\omega| = \infty$, since FD-DEVS has finite states, the resulting $\Delta(q_0, \omega)$ can either stay at a $q' \in Q$ whose $\tau(q') = \infty$ or move around in the set of state in a transition loop. Both cases can be seen as *staying at a strongly connect components* that we will talk about from now on.

A. Behavior of Atomic FD-DEVS

Given a total state $q \in Q$, q is said to be *reachable* to $q' \in Q$ if $\exists\omega \in \Omega_Z$ such that $\Delta(q, \omega) = q'$. The *strong components (or strongly connected components) of* $q \in Q$ is the *maximal set* whose every element is reachable to q, and vice versa. Formally, the strong components of q is defined by a function $\mathsf{SC} : Q \rightarrow 2^Q$ such that

$$\mathsf{SC}(q) = \{q'|\omega, \omega' \in \Omega_Z : \Delta(q, \omega) = q', \Delta(q', \omega') = q\}$$

For example, in CRC of Figure 1 (b), SC((I1,0,0))= {(I1,0,0)} and SC((G,10,[0,10]))= {(G,10,[0,10]), (GR,5, [0,5]), (WW,2,[0,2]), (W,26,[0,26]), (DW,2,[0,2]) }.

To define a set of goals in our model easily, we would use the discrete state $s \in S$ rather than a total set $q \in Q$. To drop time information from a total state $q = (s, t_s, e) \in Q$, we use the *discrete state of* q as $\mathsf{disc\text{-}s}(q) = s$. The disc-s is also overloaded as a function $\mathsf{disc\text{-}s} : 2^Q \rightarrow 2^S$ such that for $Q' \subseteq Q$,

$$\mathsf{disc\text{-}s}(Q') = \{s|(s, t_{si}, e) \in Q'\}$$

For example, in CRC of Figure 1(b), disc-s(SC((G,10,[0,10]))={G, GR, WW, W, DW}

Given a pair of (M, A) where M is an atomic FD-DEVS and $A \subseteq S$ is its acceptance states, the *behavior (or language) of* (M, A) is defined as

$$L(M, A) = \{\omega \in \Omega_Z | \text{disc-s}(\text{SC}(\Delta(q_0, \omega))) \cap A \neq \varnothing\}$$

In other words, the language of (M, A) is the set of all possible segments that reach $s \in A$ infinitely often. Let A={W} for CRC shown in Figure 1(b), then $L(\text{CRC}, A) = \{\omega | \omega$=(0,w:0)(0,g:1)(($t_i$,p)($t_i$+5,g:0)($t_i$+7,w:1)($t_i$+33, w:0)($t_i$+35, g:1))^i where $i = 0$ to ∞. }.

B. Behavior of Verification Frame

If a coupled FD-DEVS N is built for verification of a system, the components D of N may consist of two disjointed components: *components under testing* and tester, called *rejectors*. A rejector is supposed to have its acceptance states as bad situations. If there is no event segment that leads any rejector to reach a bad situation, all behaviors of components under testing can be *legal behavior* unless there no component under testing cannot reach its acceptance state. Thus, we need to show that no rejectors reach bad situations while also showing that all components under testing can reach their acceptance states.

Suppose that N is a verification frame N whose subcomponents D consists of a set of components under testing D_T and a set of rejectors D_R such that $D = D_T \cup D_R$ and $D_T \cap D_R = \varnothing$ while the *set of acceptance states* of N is the n-tuple $A =< \ldots, A_i, \ldots >$ where $n = |D|$ and A_i denotes the set of acceptance states of $M_i \in D$. The *projected discrete states from Q' with i* for a set of total states $Q' \subseteq Q$ is

$$\text{disc-s}(Q', i) = \{s_i \in S_i | (\ldots, (s_i, t_{si}, e_i), \ldots) \in Q'\}$$

Then, the *illegal behavior (or illegal language)* is defined as $L_R(N, A) =$

$$\{\omega \in \Omega_Z | \exists M_i \in D_R : \text{disc-s}(\text{SC}(\Delta(q_0, \omega)), i) \cap A_i \neq \varnothing\}$$

where $q_0 = (\ldots, q_{i0}, \ldots)$ is the *initial total state* of N such that for all M_i $q_{i0} = (s_{0i}, \tau(s_{0i}), 0)$. In other words, $L_R(N, A)$ is a set of segments that lead at least one rejector to reach a bad situation.

In addition, the *behavior (or language) of components under testing* is defined as $L_T(N, A) =$

$$\{\omega \in \Omega_Z | \forall M_i \in D_T : \text{disc-s}(\text{SC}(\Delta(q_0, \omega)), i) \cap A_i \neq \varnothing\}$$

Thus $L_T(N, A)$ is a set of segments that make all components under testing reach an acceptance state infinitely often.

A pair of (N, A) is said to be *accepted* if $L_R(N, A) = \varnothing$ and $L_T(N, A) \neq \varnothing$.

C. Decidability of (N, A) Acceptance

For checking if $L_T(N, A) \neq \varnothing$ and $L_R(N, A) = \varnothing$, to generate all possible segments in Ω_Z might be impossible because the number of segments under testing can be infinite.

Instead of generating all segments directly, we can use the graph structure of all possible transitions of FD-DEVS. This paper utilizes a finite reachable graph for FD-DEVS networks. [5] The finite reachable graph of a a coupled FD-DEVS, $N =< X, Y, D, C_{xx}, C_{yx}, C_{yy} >$ is define as

$$RG(N) =< Z, V, v_0, E >$$

where

- $Z = X \bigcup_{M_i \in D} Y_i \cup \{\epsilon\}$ is the set of triggering events.
- V is a set of zones. A zone $v = ((\ldots, (s_i, t_{si}), \ldots), \mathcal{D})$ consists of a state-scheduler vector $(\ldots, (s_i, t_{si}), \ldots)$ and a clock zone $\mathcal{D}$.
- $v_0 \in V$ is the initial zone such that $v = ((\ldots, (s_{0i}, \tau_i(s_{0i})), \ldots), \mathcal{D}^0)$ where $\mathcal{D}^0$ is the initial clock zone.
- $E \subseteq V \times Z \times V$ is a transition relation that satisfies the following property: For $q = (\ldots, (s_i, t_{si}, e_i), \ldots)$, $q' = (\ldots, (s'_i, t'_{si}, e'_i), \ldots)$ and $z \in Z$,

  $$\delta(q, z) = q' \Leftrightarrow (v, z, v') \in E$$

  such that $v = ((\ldots, (s_i, t_{si}), \ldots), \mathcal{D}), \forall e_i \in \mathcal{D}$ and $v' = ((\ldots, (s'_i, t'_{si}), \ldots), \mathcal{D}'), \forall e'_i \in \mathcal{D}'$. □

A *transition path* is a state transition sequence caused by an event sequence $\bar{z} = z_0 z_1 \ldots z_n \in Z^*$ from the initial vertex v_0 to v and we denote it as $v_0 \xrightarrow{\bar{z}} v_n$ if $(v_0, z_0, v_1) \in E, \ldots, (v_{n-1}, z_n, v_n) \in E$. To define the infinite length of transition sequence, we define strong components of $v \in V$ again. $\text{SC} : V \rightarrow 2^V$ defines a strong components of v such that

$$\text{SC}(v) = \{v' | \exists \bar{z}, \bar{z}' \in Z^*, v \xrightarrow{\bar{z}} v', v' \xrightarrow{\bar{z}'} v\}$$

The *kernel directed and acyclic graph* (shortly kernel DAG) of a graph G [9] is used in checking the emptiness of the illegal behavior. The kernel DAG of $RG(N)$ is

$$K(RG(N)) =< \mathsf{V}, \mathsf{E} >$$

where

- V is the set of $\text{SC}(v)$ for each $v \in V$ of $RG(N)$. In particular, $\mathsf{v}_0 = \text{SC}(v_0)$ is the initial node.
- $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ is the set of arcs such that $(\mathsf{v}_1, \mathsf{v}_2) \in \mathsf{E}$ implies $(v_1, z, v_2) \in E$ of $RG(N)$ such that $\text{SC}(v_1) = \mathsf{v}_1$ and $\text{SC}(v_2) = \mathsf{v}_2$.

[5] For details of a generating algorithm of the finite reachable graph for coupled FD-DEVS, the reader can refer to [7].

In $K(RG(N))$, a node $\mathsf{v} \in \mathsf{V}$ is reachable from v_0 if there is a sequence of arcs that reaches from v_0 to v. And we overload the projected discrete sets functions for $v \in V$ of $RG(N)$ and $\mathsf{v} \in \mathsf{V}$ of $K(RG(N))$, respectively:

$$\textsf{disc-s}(v, i) = s_i \text{ for } v = ((\ldots, (s_i, t_{si}), \ldots), \mathcal{D})$$

and

$$\textsf{disc-s}(\mathsf{v}, i) = \{\textsf{disc-s}(v, i) | v \in \mathsf{v}\}$$

Theorem 1: Suppose that N is a verification frame and A is the vector of acceptance states for N. Then $L_R(N, A) = \varnothing$ and $L_T(N, A) \neq \varnothing$ if $\forall \mathsf{v} \in \mathsf{V}$ reachable from v_0 s.t. $\forall M_i \in D_R, \textsf{disc-s}(\mathsf{v}, i) \cap A_i = \varnothing$ and $\exists \mathsf{v} \in \mathsf{V}$ reachable from v_0 s. t. $\forall M_i \in D_T, \textsf{disc-s}(\mathsf{v}, i) \cap A_i \neq \varnothing$.

Proof: If $\forall \mathsf{v} \in \mathsf{V}$ reachable from v_0 s.t. $\forall M_i \in D_R, \textsf{disc-s}(\mathsf{v}, i) \cap A_i = \varnothing$.
$\Rightarrow \forall M_i \in D_R, \exists v \in \mathsf{v}$ s.t. $\textsf{disc-s}(v, i) \cap A_i = \varnothing$.
$\Rightarrow \forall \omega \in \Omega_Z$ s.t. $\textsf{disc-s}(\Delta(q_0, \omega), i) \cap A_i = \varnothing$.
$\Rightarrow L_R(N, A) = \varnothing$.

If $\exists \mathsf{v} \in \mathsf{V}$ reachable from v_0 s.t. $\forall M_i \in D_T, \textsf{disc-s}(\mathsf{v}, i) \cap A_i \neq \varnothing$.
$\Rightarrow \forall M_i \in D_T, \exists v \in \mathsf{v}$ s.t. $\textsf{disc-s}(v, i) \cap A_i \neq \varnothing$.
$\Rightarrow \exists \omega \in \Omega_Z$ s.t. $\textsf{disc-s}(\Delta(q_0, \omega), i) \cap A_i \neq \varnothing$.
$\Rightarrow L_T(N, A) \neq \varnothing$. ■

Lemma 1: Given the pair of (N, A), checking $L(N, A) = 0$ is decidable.

Proof: By Theorem 1, we can show that $\exists \mathsf{v} \in \mathsf{V}$ is reachable from v_0 s. t. $\forall M_i \in D_T, \textsf{disc-s}(\mathsf{v}, i) \cap A_i \neq \varnothing$ and $\forall \mathsf{v} \in \mathsf{V}$ is reachable from v_0 s.t. $\forall M_i \in D_R, \textsf{disc-s}(\mathsf{v}, i) \cap A_i = \varnothing$ for checking if $L_T(N, A) \neq \varnothing$ and $L_R(N, A) = \varnothing$. It is known that generating $RG(N)$ is decidable in exponential time of $|D|$ that is the number of atomic FD-DEVS [7]. Constructing $K(RG(N))$ is decidable with $O(|V| + |E|)$ complexity where $|V|$ and $|E|$ are the numbers of zones and transitions of $RG(N)$, respectively [9]. ■

Example 2 (Mutual Exclusion between Ⓖ and Ⓦ): Let's consider again the cross road lights Ⓖ and Ⓦ shown in Figure 1(a) and suppose that they are controlled by CRC shown in Figure 1(b) through output ports g and w, respectively. Now we want to check whether Ⓖ and Ⓦ have "on" exclusively or not. To do this, we use rejector MXR that reaches R11 state when CRC violates the requirement, that means, it transmits 1 though g before sending 0 though w, vice versa.

We can build a verification frame, VCM shown in Figure 1(d) where D_T={CRC}, D_R={MXR}; Given the vector of acceptance states $A =< A_{\texttt{CRC}}, A_{\texttt{MXR}} >=< \{\texttt{W}\}, \{\texttt{R11}\} >$, we first check $L_T(\texttt{VCM}, A) \neq \varnothing$ using $K(RG(N))$. Figure 2(a) shows $RG(\texttt{VCM})$ whose a vertex is a reachable state of $RG(\texttt{VCM})$ so there are 7 vertices. We can make $K(RG(\texttt{VCM}))$ by combining $\textsf{SC}(v)$ for each $v \in V$ of $RG(\texttt{VCM})$ so that 5 vertices surrounded by a dashed line are contained by v_2 in $K(RG(\texttt{VCM}))$.

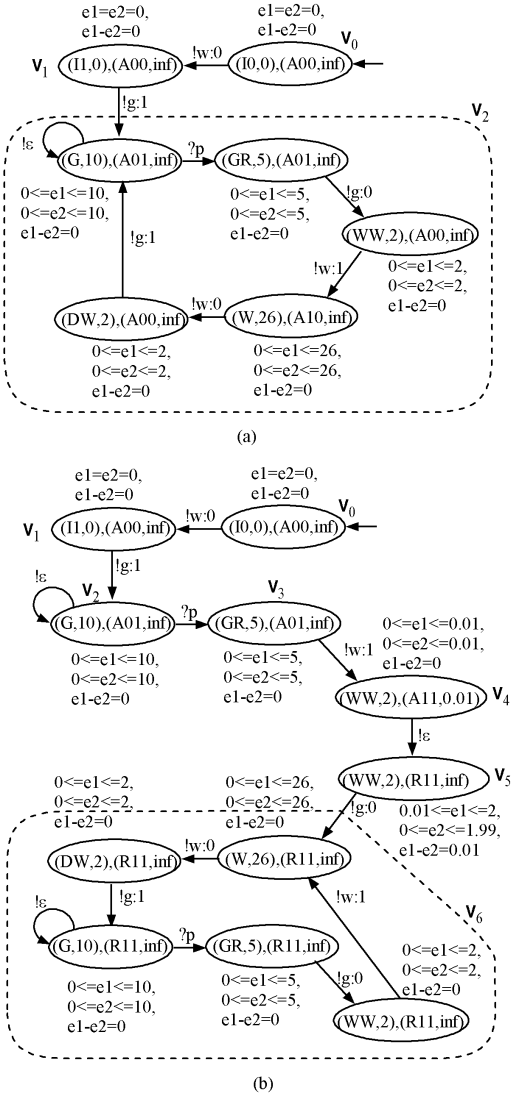

Fig. 2. Reachable Graph of Cross Road Verification: (a) Empty Illegal Behavior (b) Nonempty Illegal Behavior

To verify if $L_R(\texttt{VCM}, A) = \varnothing$, we need to check for all $\mathsf{v} \in \mathsf{V}$ with index MXR such that $\textsf{disc-s}(\mathsf{v}_0,\texttt{MXR}) \cap A_{\texttt{MXR}}=\{\texttt{A00}\} \cap \{\texttt{R11}\} = \varnothing$; $\textsf{disc-s}(\mathsf{v}_1,\texttt{MXR}) \cap A_{\texttt{MXR}}=\{\texttt{A00}\} \cap \{\texttt{R11}\} = \varnothing$; $\textsf{disc-s}(\mathsf{v}_0,\texttt{MXR}) \cap A_{\texttt{MXR}}=\{\texttt{A01, A00, A10}\} \cap \{\texttt{R11}\} = \varnothing$; Thus we can say that $L_R(\texttt{VCM}, A) = \varnothing$.

Similarly, we can check if $L_T(\texttt{VCM}, A) \neq \varnothing$. For R_T={CRC} since $\textsf{disc-s}(\mathsf{v}_2,\texttt{CRC}) \cap A_{\texttt{CRC}}=\{$ G, GR, WW, W, DW $\} \cap \{\texttt{W}\}=\{\texttt{W}\} \neq \varnothing$, so $L_T(\texttt{VCM}, A) \neq \varnothing$. Therefore, we can accept the implementation, CRC, under the tester (rejector), MXR.

However, if we switch outputs of states GR and WW of

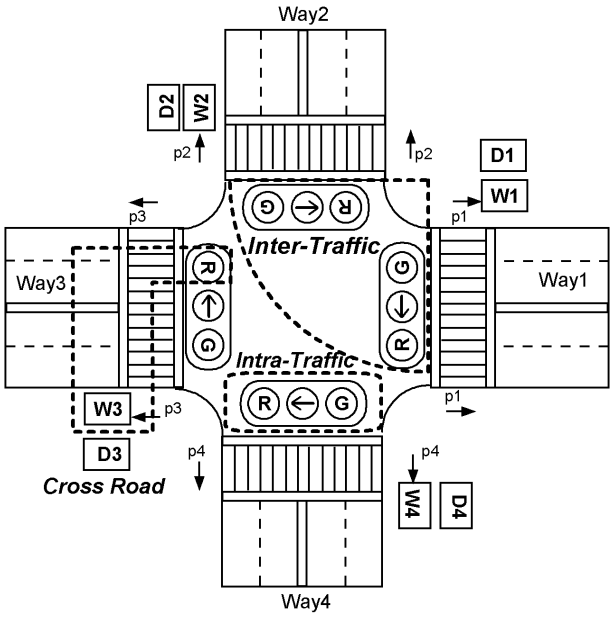

Fig. 3. Configuration of Intersection Lights

CRC such that $\lambda(\texttt{GR}) = \texttt{w:1}$ and $\lambda(\texttt{WW}) = \texttt{g:0}$, we get a different reachable graph from the original, as shown in Figure 2(b). Even though $L_T(\texttt{VCM}, A) \neq \varnothing$ as the same as the original, we can find the bad behavior detected by MXR: $\textsf{disc-s}(\textsf{v}_5, \texttt{MXR}) \cap \{\texttt{R11}\} = \textsf{disc-s}(\textsf{v}_6, \texttt{MXR}) \cap \{\texttt{R11}\} = \{\texttt{R11}\} \neq \varnothing$. Therefore, this implementation of CRC controller is rejected by MXR. □

V. Illustrative Example

This section shows an example of modular verification of an intersection traffic light system that is shown in Figure 3. This intersection has four ways and each way has three lights: green (Ⓖ), left turn (⊖), and red (Ⓡ) for traffic, and two lights: walk (Ⓦ) and don't walk (Ⓓ)) for pedestrians.

A. System Requirements

The requirements for this system control can be enumerated as follows.

1) Intra Traffic Way
   a) (Safety) Only one light among Ⓖ ⊖, and Ⓡ can be on. More than one light on is allowed within 0.01 sec.
   b) (Time Constraints) Once Ⓖ is on, it lasts 35±1 sec., while ⊖ for 15±1 sec.
   c) (Fairness) The circulation order of turning on is Ⓡ, Ⓖ, ⊖ and Ⓡ again unless the pausing signal externally occurs.
2) Cross Road
   a) (Safety) Ⓦ and Ⓓ are mutual-exclusively on with switching tolerant time 0.01 sec.
   b) (Safety) When Ⓦ is on, Ⓡ should be on.
   c) (Fairness and Time Constraints) Without pushing button, Ⓓ is on. If one pushes the button when Ⓓ is on, Ⓦ will be on 2±0.1 sec. after Ⓡ becomes newly on again. This behavior repeats forever.
3) Inter Traffic Ways: Notice that pairs (Way1 and Way3) and (Way2 and Way4) should be controlled in the same way at the same time. For example, Way1 has ⊖ on, so Way3 does. Thus we need to build the control logics for Way1 and Way2 not for all four traffic ways. From now on, we focus the control logic between Way1 and Way2.
   a) (Starting) We can push the starting button after the booting time 5 sec. is passed. When the starting button is pushed, Way1 has Ⓖ.
   b) (Terminating) For emergency or maintenance, when the pause button is pushed, all ways become Ⓡ within less than 5 sec. and the system returns to the booting status.
   c) (Fairness) Ⓖ moves to the other way (Way1 ⇆ Way2). This circulation repeats until the pause button is pushed.
   d) (Time Constraints for Safety) When Ⓖ turns from one to another, there is 2±0.5 sec. delay unless the pause button is pushed.
   e) (Safety) When one way has Ⓖ or ⊖, other side has Ⓡ.

B. Implementation

1) Intra Traffic Controller: Figure 4 illustrates how we implemented a verification frame for the intra-traffic controller. Figure 4 (a) shows an implementation of Spec1-a that is a rejector detecting violation of the mutual exclusion between g, a, and r where MXR1, MXR2, MXR3 are the atomic FD-DEVS introduced in Example 1. For checking time constrains of Spec1-b, we made an atomic FD-DEVS TimeRangeD_R which detects "too early error" from E to RJ or "too late error" from L to RJ where time tolerant range is $[d-t, d+t]$. Using TimeRangeD_R, we could build the Spec1-b checker as shown in Figure 4(c). Figure 4(d) illustrates an atomic FD-DEVS model for the Spec1-c checker that detects the violation of circulating order among Ⓡ, Ⓖ and ⊖. Figure 4(e) shows the verification frame for checking Spec1-a, Spec1-b and Spec1-c are not violated and the controller, GAR has the legal and live behavior. GAR has its initial state IR that generates r:1 to turn Ⓡ on first, and then turn Ⓖ and ⊖ off by output g:0 and a:0 at IG and IA, respectively. When receiving state signal s:1, GAR gets into D that turns Ⓡ off by r:0 and moves to R2G turning Ⓖ on by g:1. After staying 35 sec. at G, GAR turns Ⓖ off by g:0 and changes into G2A that turns ⊖

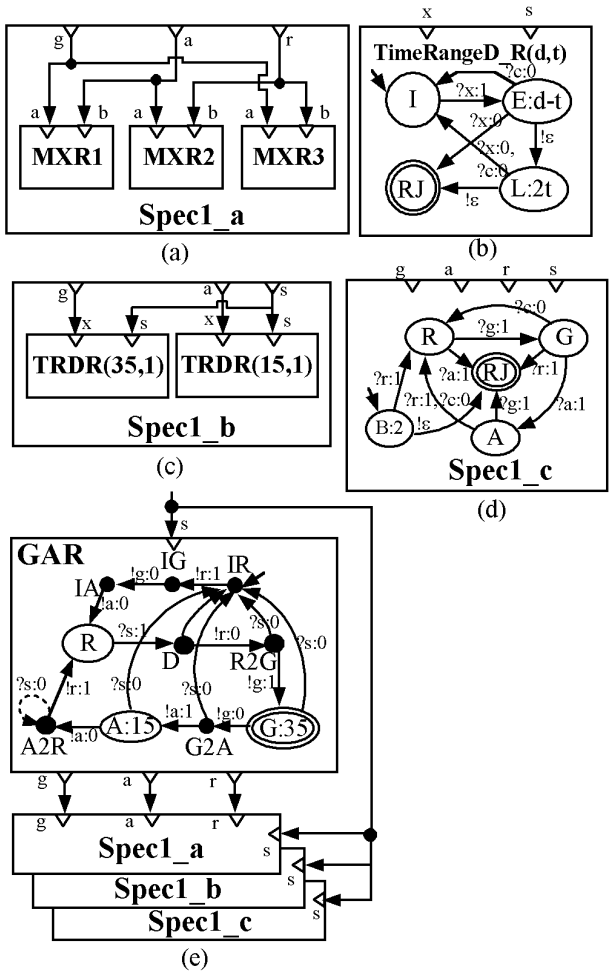

Fig. 4. Verification Frame of Intra-Traffic Control (a) Checker for Spec1-a (b) TimeRangeD_R (c) Checker for Spec1-b (d) Checker for Spec1-c (e) Verification Frame for Spec1

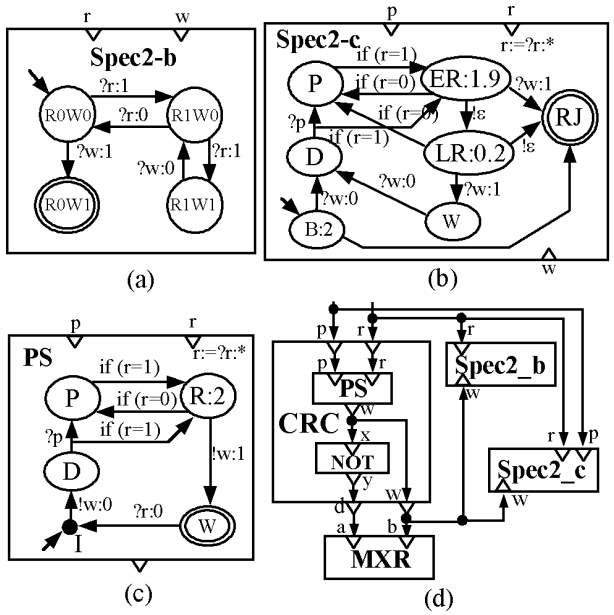

Fig. 5. Verification Frame of Cross Road Control (a) Checker for Spec2-b (b) Checker for Spec2-c (c) Push Switch (d) Verification Frame for Spec2

on by a:1. After staying 15 sec. at A, it turns ⊖ off by a:0 and goes back to R with turning Ⓡ on again. When pausing event s:0 occurs at either D, R2G, G, G2A or A, the current state moves to IR for re-initializing all lights Ⓖ,⊖ and Ⓡ.

2) Cross Road Controller: To construct the verification frame for cross road specifications we model components as illustrated in Figure 5. The checker for safety requirement Spec2-b was implemented as shown in Figure 5(a) while the time constrained behavior of Spec2-c was built as Figure 5(b). The component under testing was a coupled FD-DEVS whose one model is a push switch PS shown in Figure 5(c) and the other is NOT gate (whose state diagram is omitted here). Figure 5(d) shows the verification frame checking the cross road requirements.

3) Inter Traffic Controller: To control inter traffic ways, we used an atomic FD-DEVS model, called module circulator (MC) as shown in Figure 6(a). This model is booting for 2 seconds at B and it becomes ready to handle the starting event s at state I. Once it gets the starting signal s at I, it moves to toM1 whose τ(toM1)=2 for a safety then moves to M1 with output g1:1 that is triggering the circulation of Ⓖ,⊖ and Ⓡ at Way1. When getting r1:1, the circulation moves to Way2 through toM2 and M2. Then it repeats forever unless the pausing event p happens. If p occurs, depending on the current states it moves directly to B or it moves via P1 or P2 to B.

As shown in Figure 7(b), we could construct a inter-traffic controller (ITC) using one MC and two GARs named GAR1 and GAR2 which were introduced first in Figure 4(e). We built the checker for Spec3-abcd as one atomic FD-DEVS as shown in Figure 6(c). That is, it checks, the starting and terminating behaviors, circulation fairness of authority between Way1 and Way2 as well as time constraints for staying at a state. The checker for Spec3-e was implemented using two OR gates and two mutually inclusion rejector, MIR1 and MIR2 as shown at the bottom of Figure 6(d). Whole coupling information of the verification frame for the inter-traffic control is shown in Figure 6(d).

Finally, as we can see Figure 7, the intersection light controller (ILC) could be built using one ITC and four cross road controllers that are grouped as (CRC1, CRC3) and (CRC2, CRC4).

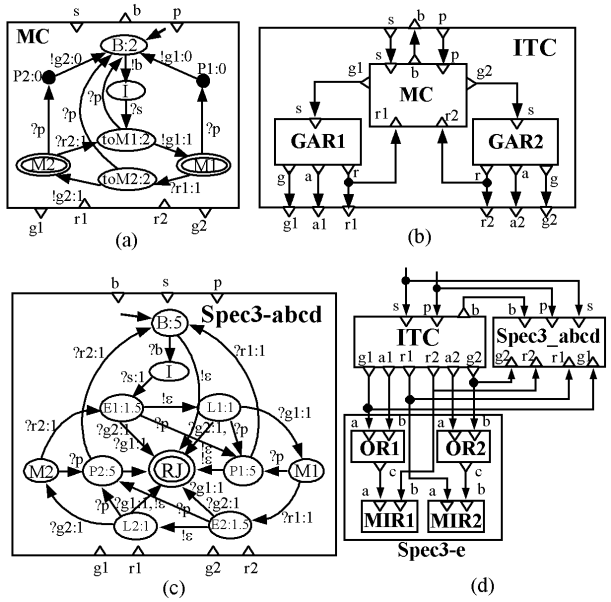

Fig. 6. Verification Frame of Inter-Traffic Control (a) Module Circulator (b) Inter-Traffic Controller (c) Checker for Spec3-abcd (d) Verification Frame for Spec3

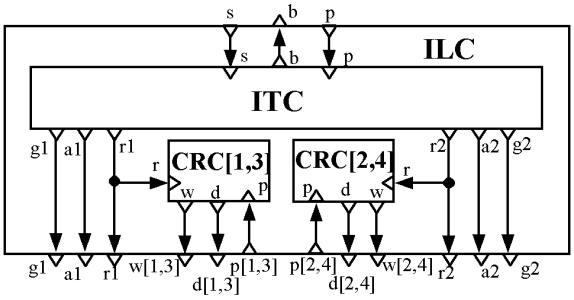

Fig. 7. Implementation of Intersection Lights Control

C. Verification Performance

Hardware platform in which we implemented has Intel Pentium-4™ (3 GHz) with 512 MBytes RAM. We used Microsoft Visual.Net 2005™ for compiling the C-# language.

We first checked the emptiness of illegal behavior using the set of checkers for Spec1, Spec2, and Spec3 whose configurations have been illustrated in Figure 4, Figure 5, and Figure 6, respectively. All of the checkers could not find any bad behavior. For checking the non-emptiness of legal behavior of the integrated components ILC shown in Figure 7, we varied the number of push buttons used ranging from none to 3 (we stopped the case of full usage of all push buttons in cross roads because of the memory lack). All of configurations have the non-empty legal behavior. Table I summarizes the result of our performance experiment.

TABLE I
PERFORMANCE OF CHECKING ILLEGAL AND LEGAL BEHAVIORS

Behavior		$\|D\|$	$\sum \|S\|$	$\|V_1\|$	Time1	$\|V_2\|$	Time2
L_R	a	7	42	38	0.4	2	0.4
	b	5	30	23	0.14	6	0.06
	c	8	65	128	0.14	19	0.1
L_T	0	13	80	311	1.47	272	1.39
	1	13	80	690	2.88	427	2.86
	2	13	80	2,663	11.23	718	11.98
	3	13	80	19,086	1:47.44	1,329	1:58.45

L_R : $L_R(N, A)$; L_T : $L_T(N, A)$; a: Spec1; b: Spec2; c: Spec3; 0: no push button; 1: using push button1; 2: using push buttons 1 & 2; using push buttons 1,2 & 3; $|D|$: the number of involved atomic FD-DEVS; $\sum |S|$: the number of all states in D; $|V_1|$: the number of vertices of $RG(N)$; Time1: the elapsed time for generating $RG(N)$; $|V_2|$: the number of vertices of $K(RG(N))$; Time2: the elapsed time for generating $K(RG(N))$;

VI. Contributions

This paper proposes a verification framework based on a sub-class of DEVS, called FD-DEVS so that the finite reachable graph of FD-DEVS networks is used for checking the emptiness of the illegal behavior as well as the non-emptiness of the legal behavior. One of the advantages of this verification framework over other formalism such as timed automata [2] is its *modularity*, which has been originated from the coupling scheme of DEVS [14]. Each component in our verification framework can work as an independent module as interacting with other components through the coupling relation. However, since the synchronization of state transition between two timed automata is done by the *same name* of events [2] [1] [11], a component might not be reused for different purposes. For example, the identical mutual exclusion rejectors, `MXR1`, `MXR2`, and `MXR3` with input ports `a` and `b` shown in Figure 4(a) should change their port names for synchronization with different events `g`, `a` and `r` when using the synchronization used in the automata theory.

The applicability of this proposed verification framework is much broader than other DEVS approaches [12], [4], [10] that seem to be applicable in only *closed systems* which don't interact with external influences. In addition, compared to the experimental framework which is based on the simulation of DEVS models [15], this proposed framework has advantages because it generates all possible reachable states while simulation attempts to trace one possible trajectory in a run. In other words, we don't know when we can stop the simulation experiment for the perfect testing, especially in case of the system under testing being an open system.

VII. Conclusion and Further Research

To check if the implementation of a system satisfies the requirement set, this paper proposed a verification framework based on FD-DEVS networks whose components are partitioned into testers (or checkers), called rejectors, and components under testing. In the proposed verification framework, the illegal behaviors can be detected modularly by any rejector, while non-empty legal behavior of components under testing can be guaranteed by the kernel DAG of the finite reachable graph that covers all acceptance conditions of each component. As an illustrative example, an intersection traffic control system has been verified that it's implementation met the whole set of requirements.

However, as we could see in Section V, even though the number of reachable states is finite, it can be exploded as the number of subcomponents or the number of external influences increases. Thus, extending scalability can be one of the most demanding research in our approach. One possible way for this might employee the ordered binary decision diagram [3] in which the behavior is preserved while the size of memory requirement can be greatly reduced. Another possibility might be the state reduction in which the behavior can be different from the origin if the difference can be tolerated. To do this, we can think other finite state DEVS with non-determinism in terms of time advance as well as state transitions. In addition, the performance range of coupled FD-DEVS will be needed in terms of the optimistic and pessimistic cases as done in the schedule-preserved DEVS [6]

Acknowledgment

This work was supported by the Korea Research Foundation Grant (No: M01-2004-000-20045-0).

References

[1] R. Alur. Timed Automata. *11th International Conference on Computer-Aided Verification, LNCS*, 1633:8–22, 1999.

[2] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8), 1986.

[4] K.J. Hong and T.G. Kim. Timed I/O Test Sequences for Discrete Event Model Verification. In *13th International Conference on AI, Simulation, and Planning in High Autonomy Systems*, volume 3397 of *LNCS*, pages 257–284. Springer, 2005.

[5] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, second edition, 2000.

[6] M.H. Hwang. Tutorial: Verification of Real-time System Based on Schedule-Preserved DEVS. In *Proceedings of 2005 DEVS Symposium*, San Diego, CA, Apr. 2-8 2005. SCS.

[7] M.H. Hwang and B.P. Zeigler. A Reachable Graph of Finite and Deterministic DEVS Networks. In *Proceedings of 2006 DEVS Symposium*. SCS, http://www.u.arizona.edu/∼mhhwang, 2006. Accepted.

[8] D.H. Withers R.G. Sargent, J.H. Mize and B.P. Zeigler. Hierarchical Modelling for Discrete Event Simulation (Panel). In *Proceedings of the 25th Winter Simulation Conference*, Los Angeles, CA, 1993. ACM Press.

[9] R. Sedgewick. *Algorithms in C++, Part 5 Graph Algorithm*. Addison Wesley, Boston, third edition, 2002.

[10] H.S. Song and T.G. Kim. Application of Real-Time DEVS to Analysis of Safety-Critical Embedded Control Systems: Railroad Crossing Control Example. *SIMULATION*, 81(2):119–136, Feb. 2005.

[11] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18:25–68, 2001.

[12] B. P. Zeigler and S.D. Chi. Symbolic Discrete Event System Specification. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1428–1443, Nov./Dec. 1992.

[13] Bernard P. Zeigler. *Theory of Modelling and Simulation*. Wiley Interscience, New York, first edition, 1976.

[14] Bernard P. Zeigler. *Multifacetted Modeling and Discrete Event Simulation*. Academic Press, London,Orlando, first edition, 1984.

[15] B.P. Zeigler, H.Praehofer, and T.G. Kim. *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, London, second edition, 2000.