Application of Real-Time DEVS to Analysis of Safety-Critical Embedded Control Systems: Railroad Crossing Control Example

Hae Sang Song
Department of Computer Information & Communication
Seowon University, Cheongju, Korea

Tag Gon Kim
Department of Electrical Engineering & Computer Science
KAIST, Taejon, Korea
tkim@ee.kaist.ac.kr

This article presents an application of the Discrete Event System Specification (DEVS) framework to the design and safety analysis of a real-time embedded control system, a railroad crossing control system. The authors employ an extension of the DEVS formalism, real-time DEVS (RT-DEVS), which has a sound semantics for the specification of real-time systems in a hierarchical modular fashion. The notion of a clock matrix for communicating RT-DEVS models is proposed, which represents a global time between the models. Based on the composition rules and the clock matrix, an algorithm for the generation of a timed reachability tree is developed that can be used for safety analysis at two phases: an untimed and timed analysis phase. A railroad crossing control example demonstrates that the proposed analysis for RT-DEVS models would be effective to verify the safety property of real-time control systems.

Keywords: Real-time DEVS, safety analysis, controllability, real-time embedded discrete event control systems

1. Introduction

Control of discrete event dynamic systems has attracted ever more attention for the past decades, especially for safety-critical real-time control systems. Such systems must satisfy stringent real-time requirements in performing their intended functions. Recent computing technology allows us to embed a microprocessor-based discrete event control software (controller) in real-time control systems. Formal verification of such a discrete event controller against desired system properties is known to be difficult and complex [1]. For the verification, specification of such control systems should be based on a sound formalism for discrete event systems modeling. One such formalism is the real-time Discrete Event System Specification (RT-DEVS) formalism, which provides a seamless framework for the development of real-time control software that includes modeling, design, analysis, simulation, and implementation [2]. This article deals with an application of the RT-DEVS formalism to the design and safety analysis of a safety-critical real-time embedded control system, a railroad crossing control system.

SIMULATION, Vol. 81, Issue 2, February 2005 119-136

DOI: 10.1177/0037549705052229

Much research on the control of discrete event dynamic systems has relied on the supervisory control framework based on automata theory [3, 4]. A plant automaton is regarded as a generator that generates a sequence of events, and a supervisor (controller) acts as a language recognizer. Whenever the supervisor recognizes the plant's behavior at a state, it permissively controls the plant by enabling or disabling the plant's events at the state. Strict synchronization or an interlocking mechanism between interconnected automata is assumed in the framework. To relieve this constraint, automata with *prioritized* synchronization [5] are applied to the supervisory control framework [6]. In the synchronization, a sender model has a set of prioritized events that can be fired by themselves, regardless of whether receiver models are ready to receive the events. A timed extension of the supervisory control framework employs an extended automata model, called a timed discrete event system (TDES) [7], in which a timed transition graph is generated from a TDES, and then the same supervisory framework is applied to the graph. Note that the interaction from the control to the plant is permissive, and modeling is done neither in a modular nor in a hierarchical manner.

Reachability analysis of multiple models has been a basis for verifying real-time features such as safety of an overall control system. Associated with the analysis are two major factors: a synchronization mechanism between

models and a method of timing analysis. Ostroff [8] proposed a timed transition model (TTM) in which each transition of a model is associated with a lower time bound and an upper time bound within which the transition can take place. Interconnection between models is based on the strict synchronization mechanism or interlocking. One drawback of the formalism is the state explosion due to the time dimension, as a single tick generates a new state. TDES [7] adopts a similar notion of time constraints of TTM, which thereby has the same state explosion problem. TA (timed automata) [9] is an extension of automata to the dense time domain in an interval of real values. Most reachability studies for the verification of timed automata are based on strict synchronization [5] between communicating models [10]. An extensive review of real-time software verification based on TA, including the work of VERIMAG, can be found in Sifakis, Tripakis, and Yovine [11].

As explained earlier, analysis of discrete event control systems based on TA and TTM assumes that the communication mechanism between a plant and a controller employs strict synchronization. Strict synchronization, also called interlocking, is a model of message lossless (or synchronization lossless) between communicating entities. However, there is a possibility that communicating entities may lose a message for synchronization, especially in real-time systems. This is because a component of a real-time system can send a message to another component without knowing whether the receiving component is ready. Such a weak synchronization mechanism for realistic communication can be easily modeled in the RT-DEVS formalism, which is defined in section 5. Since weak synchronization considers synchronization loss reachability, analysis based on the synchronization is much more complex than analyses using strict synchronization. Thus, the state space of a composed model with time information using weak synchronization is much bigger than that using strict synchronization, being the cause of the state space explosion problem. To alleviate the state space explosion problem due to the time dimension, we devise precise notions of a vector time and a clock matrix to get a maximal set of behaviors with minimal uncertainty. It should be noted that the RT-DEVS formalism employs dense time intervals, whereas TTM [1, 7] employs integer time intervals. Our safety analysis is based on the clock matrix of an overall control system, which is derived from the weak synchronization of communicating entities in the system.

This article is organized as follows. We overview the DEVS-based system development methodology in section 2. In the following section, we start with the problem statement of a target control system, named a railroad crossing control. In section 4, we review the RT-DEVS formalism with which an RT-DEVS model of a railroad crossing control system is modeled. A weak synchronization mechanism between component RT-DEVS models is defined in section 5. Section 6 applies the DEVS-based controller design framework to the railroad crossing control problem. Section 7 contributes to the timing analysis for RT-DEVS models with dense time intervals. Based on the timing analysis, the safety of the control system under design is analyzed in section 8. We conclude the article in section 9.

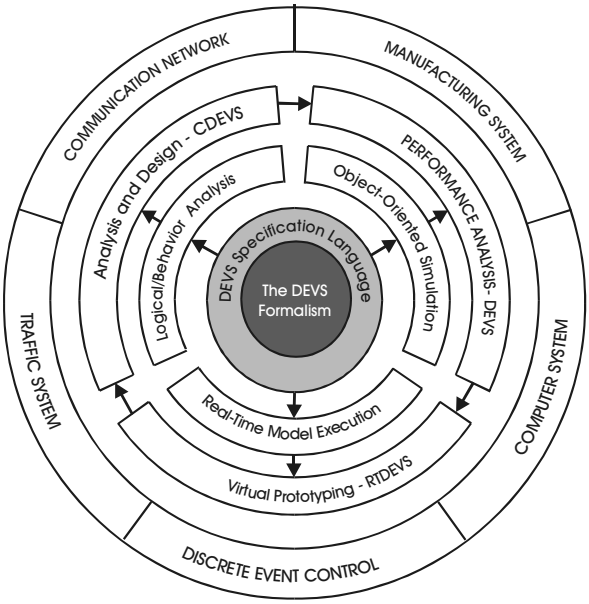


Figure 1. DEVS framework for systems development

2. Overview of the DEVS Framework for Systems Development

This section first describes a brief overview of a general methodology for DEVS-based modeling, simulation, analysis, and implementation of discrete event systems. It then explains the design and analysis of real-time discrete event control systems using the RT-DEVS formalism. Figure 1 shows a unified framework for the design and implementation of discrete event systems, which include communication networks, computer systems, traffic systems, and others. As shown in the figure, the core of the framework is the DEVS formalism. The framework provides a basis for the modeling of discrete event systems on which logical analysis, performance evaluation, and implementation can be performed *based on only one framework*. To do so, a DEVS specification language is developed, which is a language realization of the DEVS formalism, with sojourn times between states unspecified. Now a DEVS model specified by this language is used in the different phases for systems design and implementation by specifying additional information. To support the methodology, a set of tools has been developed, which include DEVSim++ [12] for performance analysis, RT-DEVSim++ [13] for real-time

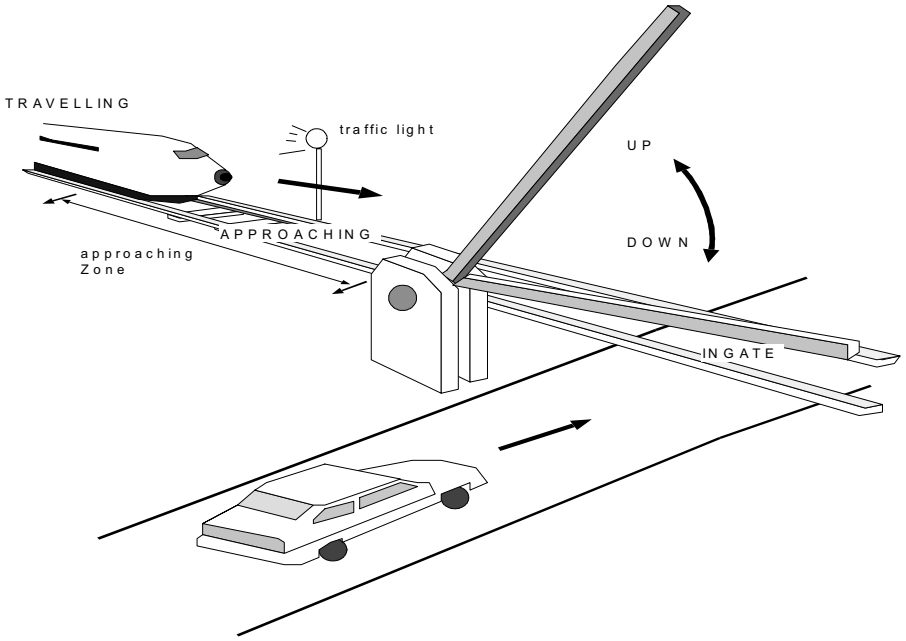


Figure 2. Railroad crossing system

simulation, DEVS Executive [2] for virtual prototyping, DEVSlog [14] for specification verification, HDEVSim++ [15] for hybrid systems analysis, and others. Due to the unified framework with such a rich set of tools, it is worthy to extend the framework to the modeling and analysis of an embedded control system.

With the DEVS framework, we present the design and analysis of a safety-critical embedded control system, a railroad crossing control system. To do so requires modification of our previous work [15] to adapt modeling and analysis of such control systems. For safety analysis of RT-DEVS models, we propose a timed behavior analysis (TBA) algorithm based on a weak synchronization mechanism of communicating atomic RT-DEVS models. The mechanism is an extension of the prioritized synchronization [5, 6] of untimed models to timed ones. As will be seen later, TBA is a general reachability analysis algorithm for both the untimed and timed analysis of RT-DEVS models.

3. Problem Statement and Required Modeling Power

Consider a railroad crossing system, as shown in Figure 2, in which a train and a car have to pass through the same crossing area, annotated by INGATE. If they are at the area at the same time, a disastrous accident might take place. Thus, we say the control objectives of this system are twofold:

(a) Any accident should be prevented by sensing the train's position and manipulating the gate in a right order at the right time (*safeness*).

(b) Furthermore, cars eventually should pass through the gate in a finite time (*liveness*).

A design problem for a real-time discrete event controller can be formulated as follows: given a plant with known dynamics and control objectives, design a discrete event controller that satisfies the control objectives. This article considers a safety analysis problem that checks whether the behavior of the overall control system falls into some *bad* state.

To analyze the system explained above, the RT-DEVS formalism should have additional expressive power that is not in the original DEVS formalism. First, to reflect characteristics of real-time systems, an event occurrence time in the real-time specification may not be an exact value but an interval. Thus, time advance in RT-DEVS should be given by an interval of real numbers. Second, a mechanism for synchronization between two RT-DEVS models should be explicitly defined within the RT-DEVS formalism. In practice, a real-time component that is trying to make an internal transition does not block other components that are not ready for synchronization. Finally, a rich set of mathematical tools are required, such as modeling, simulation, analysis, and implementation in the development framework of safety-critical control systems. The next

section reviews the RT-DEVS formalism with which a plant model of an exemplified control system is developed.

4. Real-Time DEVS Modeling

4.1 Real-Time DEVS (RT-DEVS) Formalism

The RT-DEVS formalism, proposed by Hong et al. [2], is an extension of the DEVS formalism and has the universality for real-time systems modeling, real-time simulation, and real-time execution in one framework. We briefly review the formalism that consists of two classes of RT-DEVS models: atomic models and coupled models, as in the DEVS formalism. An atomic model in RT-DEVS formalism, *RTAM*, is defined as follows:

$$RTAM =< X, S, Y, \delta_{\text{ext}}, \delta_{int}, \lambda, ta, ti, \psi, A >$$

where the first seven-tuples are the same as the DEVS formalism rewritten here:

X, S, Y: the same as the atomic model of DEVS (i.e., sets of input events, sequential states, and output events, respectively);

δ_{ext}: $Q \times X \rightarrow S$, external transition function, where Q is the total state set of $Q = \{(s, e) | s \in S \text{ and } 0 \leq e \leq ta(s)\}$, and $ta(s)$ is the state sojourn time;

δ_{int}: $S \rightarrow S$, internal transition function;

λ: $S \rightarrow Y$, output function;

ta: $S \rightarrow \Re^{+}_{0,\infty}$, time advance function, where $\Re^{+}_{0,\infty}$ is the nonnegative real numbers adjoining ∞.

In addition,

ti: time interval function,

ψ: activity mapping function,

A: set of activities,

with constraints

ψ: $S \rightarrow A$;

ti: $S \rightarrow \Re^{+}_{0,\infty} \times \Re^{+}_{0,\infty}$, where $ti(s)|_{\min} \leq t(a) \leq ti(s)|_{\max}$, $ti(s)|_{\min} \leq ta(s) \leq ti(s)|_{\max} s \in S, a = \psi(s) \in A$, and $t(a)$ is the execution time of an activity $a \in A$;

$A = \{a | t(a) \in \Re^{+}_{0,\infty}, a \notin \{X?, Y!, S =\}\}$, where $X?$ is the action of receiving data from X, $Y!$ is the action of sending data through Y, and $S =$ is the action of modifying a state in S.

Notice that *RTAM* is the same as the original DEVS atomic model, except for three additional elements related to the realization of models: time interval function ti, activity mapping function ψ, and set of activities A. An activity associated with a state is a realization of an abstract state transition to a concrete activity such as message transmission, processing a task, or any valuable computation. A time interval of real numbers (dense time) associated at a state is to model an activity execution time $t(a)$, $a = \psi(s) \in A$, which may not be specified by an exact value. For the purpose of timing analysis, we will use a reduced form of *RTAM* by assuming that $ta(s) = t(a)$, $\psi^{-1}(\psi(s)) = s$ and also ignoring implementation-dependent terms such as activity-mapping function ψ and the associated activity set. Thus, a reduced form of *RTAM* is given with the following seven-tuple:

$$RTAM' =< X, S, Y, \delta_{\text{ext}}, \delta_{int}, \lambda, ti > . \quad (1)$$

It is interesting to note that *RTAM′* is the same as the DEVS formalism, except that the time advance function is replaced by the time interval function. It, however, keeps enough information to analyze timed behavior of real-time systems, which we are concerned with in this article. We will use *RTAM* and *RTAM′* interchangeably hereafter.

Models constructed from components are formalized in the RT-DEVS formalism as coupled models, *RTDN*:

$$RTDN =< X, Y, M, EIC, EOC, IC >, \quad (2)$$

where

X : input events set,

Y : output events set,

M : RT-DEVS components set,

$EIC \subseteq RTDN.IN \times M.IN$: external input coupling relation,

$EOC \subseteq M.OUT \times RTDN.OUT$: external output coupling relation,

$IC \subseteq M.OUT \times M.IN$: internal coupling relation.

Here, *RTDN.IN*, *RTDN.OUT*, *M.IN*, and *M.OUT* represent the input and output ports of the newly constructed coupled model and component models, respectively. Three elements—*EIC*, *EOC*, and *IC*—specify the connections between the set of models M and input and output ports of the coupled model *RTDN*. Note that *RTDN* is the same as the DEVS formalism, except that the tie-breaking selector function in the DEVS formalism is not specified. No specification of the function intentionally represents an execution order of simultaneous events to be random, which reflects characteristics of real-time systems. Accordingly, simulation of a RT-DEVS model would execute such simultaneous events in a random order, which indeed is a generalization of a fixed order. The generalization means that analysis of RT-DEVS models would consider all possible orders of simultaneous events, not just one order specified by a select function.

4.2 Graphical Notation

RT-DEVS models in equations (1) and (2) may be represented with a graphical notation [2, 16] for intuitive modeling and communication. A component model in a coupled model consists of a surrounding box (Fig. 4). Along the boundary of the box, small triangles are placed inward for input ports and outward for output ports. A box can contain other boxes to express closure under coupling defined in the DEVS formalism. A coupling scheme defined in a coupled model specifies links between components. They are represented in a hierarchical fashion by a solid line for three couplings: *IC* from an output port of one component to an input port of another, *EIC* from an input port of the coupled model to an input port of component(s), and *EOC*

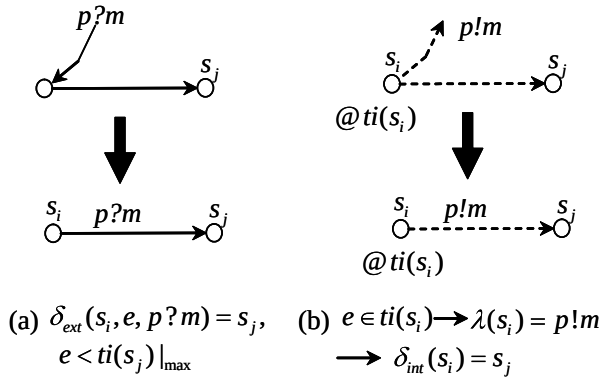


Figure 3. Graphical notations of DEVS transitions: (a) external transition and (b) internal transition

from an output port of a component to an output port of the coupled model itself.

An atomic model is placed inside an innermost box (no box inside the box). Figure 3 represents external and internal transitions in which an input event is represented by "?" and an output event by "!." For example, an input event $in?m$ means that a message m is input at the input port in, and an output event $out!m$ means that a message m is output at the port out. A dotted line represents an internal state transition specified by the internal transition function, δ_{int}. A solid line represents a state transition specified by the external transition function, δ_{ext}. This notation derives from CSP (Communicating Sequential Processes) [17], in which a receiving process waits at a "?" step in its program until the sending process sends a matching "!" message. If ports do not need to be explicit, we replace x by $x = p?m$ and y by $y = p!m$. Finally, a time interval $[t_a, t_b]$ is located near state s, where $ti(s) = [t_a, t_b]$ with the prefixed "@" sign. Usually, the time interval $[\infty,\infty]$ will be omitted in the specification when it is obvious.

4.3 The Railroad Crossing Plant Model

Figure 4 shows a railroad crossing system model N with two component models: a coupled model *TG-PLANT* (or P) and an atomic model *CONTROLLER* (or C). The coupled model P, in turn, consists of two atomic models: train *TRAIN* (or T) and gate *GATE* (or G). The meaning and abbreviation of each state are shown in Table 1. Note that the state name *TRAV* of the train can be represented by the abbreviation $T.T$, or T, for short, if it causes no confusion, where $T.x$ means that the state x belongs to atomic model T. This convention will be used throughout this article. Following the terminology convention used in control systems, we call a subsystem under control a *plant* (coupled model P of the gate and train) and one controlling the plant a *controller* (model C). Then, the coupled RT-DEVS

Table 1. Description of states of the train and gate models

State Name	Abbreviation	Description
TRAV	*T* or *T.T*	Train is traveling the safe region. *T.x* emphasizes that state *x* belongs to model *T* (train).
APPR	*A* or *T.A*	Train is approaching near to the gate.
INGA	*I* or *T.I*	Train is now passing the gate.
UP	*U* or *G.U*	Gate is open. *G.x* means that state *x* belongs to model *G* (gate).
DOWN	*D* or *G.D*	Gate is closed.

model of the plant P in the figure is

$$P.X = \{C\}, \quad P.Y = \{M\}, \quad P.M = \{T, G\},$$

$$P.IC = \{\}, \quad P.EIC = \{(P.C, G.C)\},$$

$$P.EOC = \{(T.M, P.M)\}.$$

And the atomic RT-DEVS of the train T is

$$T.X = \{\}, \quad T.Y = \{M!appr, M!exit, M!enter\},$$

$$T.S = \{TRAV, APPR, INGA\}, \quad \delta_{int}(TRAV) = APPR,$$

$$\delta_{int}(APPR) = INGA, \quad \delta_{int}(INGA) = TRAV,$$

$$\lambda(TRAV) = M!appr, \quad \lambda(APPR) = M!enter,$$
$$\lambda(INGA) = M!exit,$$

$$ti(TRAV) = [90, 95], \quad ti(APPR) = [30, 35],$$
$$ti(INGA) = [20, 25].$$

Note that the train model has three internal (or spontaneous) transitions—$\delta_{int}(TRAV)$, $\delta_{int}(APPR)$, and $\delta_{int}(INGA)$—which cannot be preempted since there is no external transition defined at those states. In the meantime, the atomic RT-DEVS of the gate G has two external transitions associated with two external events: $C?up$ and $C?down$. Note also that there is no direct communication between the train and the gate (no internal coupling of P, $P.IC = \{\}$). Thus, they run independently without the controller's mediation. If we eliminate the time interval in an atomic model on purpose, the resultant model is called an *untimed* model.

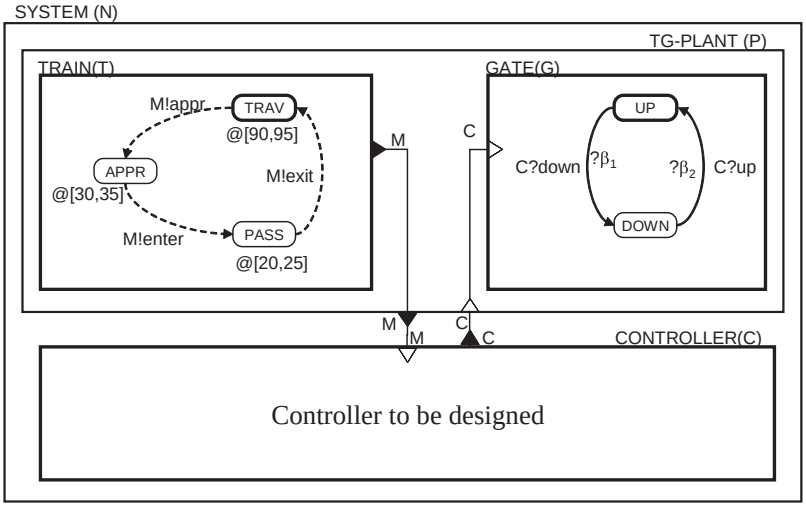


Figure 4. Railroad crossing system model with a controller model unspecified

5. Weak Synchronization: Communicating RT-DEVS Models

The DEVS formalism does not explicitly define a communication mechanism between components coupled together. Here, three assumptions for communication between RT-DEVS models are made: (a) a coupling *immediately* converts an output event of one component model to an input event of the other connected by coupling relations. (b) Every port has a buffer with a length 1, and every arrived message is processed and cleared instantly. (c) A model sending an event does not wait for any acknowledgment from a model receiving the event.

Under the assumptions, RT-DEVS has an explicit definition of a synchronization mechanism between coupled components, called *weak* synchronization, with two characteristics: nonblocking and synchronization loss. Basically, it is similar to prioritized synchronization [11] but differs in that it is based on RT-DEVS models, which have an explicit notion of internal and external transitions in the time domain. It is different from the interlocking mechanism in the strict synchronization adopted by CSP or CCS, in which a transition takes place only if both communicating processes are ready for communication; otherwise, they are blocked. In the weak synchronization of RT-DEVS, a machine performs an internal transition and produces an output event if and only if the internal transition is *eligible*. We first define eligibility of internal and external transitions and then define weak synchronization.

Let $Q \subseteq S \times \Re^{+}_{0,\infty}$ be a set of total states of an atomic model $M =< X, S, Y, \delta_{\text{ext}}, \delta_{int}, \lambda, ti >$. A total state is a pair of sequential states and an associated elapsed time feasible at the state. Note that the total state is an infinite set.

DEFINITION 1. Eligible transition. An internal transition $(q, y, q') \in Q \times Y \times Q$, where $q = (s, e) \subseteq Q$, $y \in Y$ and $q' = (s', e') \in Q$, is called *eligible* if and only if $\delta_{\text{int}}(s) = s'$, $y = \lambda(s)$, $e = ta(s)$ with $ti(s)|_{\min} \leq ta(s) \leq ti(s)|_{\max}$ and $e' = 0$. Similarly, an external transition $(q, x, q') \subseteq Q \times X \times Q$, where $q = (s, e) \subseteq Q$, $x \in X$ and $q' = (s', e') \in Q$, is said to be *eligible* if and only if there is an input event $x \in X$ at an elapsed time e, $0 \leq e \leq ta(s)$ and $\delta_{\text{ext}}(s, e, x) = s'$. By definition, the elapsed time of the resultant total state (s', e') should be always zero (i.e., $e' = 0$).

We denote $q \xrightarrow{!y} q'$ as an eligible internal transition $(q, y, q') \in Q \times Y \times Q$, and notation $q \xrightarrow{!y} |q'$ represents an ineligible transition $(q, y, q') \in Q \times Y \times Q$. The same notations could be used for an external transition in which "?" is used instead of "!."

Now we are ready to define weak synchronization. Consider a coupled model $RTDN = < X, Y, \{M_i, M_j\}, EIC, EOC, IC >$, which has only two atomic models, M_i and M_j—that is, for $k = i, j$, $M_k = < X_k, S_k, Y_k, \delta_{\text{ext},k}, \delta_{int,k}, \lambda_k, ti_k >$. Let $Q_i \subseteq S_i \times \Re^{+}_{0,\infty}$ and $Q_j \subseteq S_j \times \Re^{+}_{0,\infty}$ be a set of total states of M_i and M_j, respectively. A pair of output/input events $(p_i!m_i, p_j?m_j)$ for $p_i!m_i \in Y_i$ and $p_j?m_j \in X_j$ is said to be *matched* if $(p_i!m_i, p_j?m_j) \in M_i.OUT \times M_j.IN \subseteq N.IC$. Assume that at a time instant, they are at their respective total states $q_i \in Q_i$ and $q_j \in Q_j$. Then, a composite total state of the coupled model N is given by $Q \subseteq Q_i \times Q_j$.

DEFINITION 2. Weak Synchronization. Weak synchronization is defined as the following composite transition rules:

1. If $q_i \xrightarrow{p_i!m_i} q_i'$ and $q_j \xrightarrow{p_j?m_j} q_j'$, then $(q_i, q_j) \xrightarrow{p_i!m_i}$

(q_i', q_j), where $(p_i!m_i, p_j?m_j)$ does not match, or

2. If $q_i \xrightarrow{p_i!m_i} q_i'$ and $q_j \xrightarrow{p_j?m_j} q_j'$, then $(q_i, q_j) \xrightarrow{p_i!m_i} (q_i', q_j')$, where $(p_i!m_i, p_j?m_j)$ matches, or
3. Rules above hold if i and j are exchanged.
4. No other transition is defined.

In a successful communication by rule 2, M_i performs an internal transition, and concurrently, M_j undergoes an external transition. However, rule 1 indicates that M_i with an eligible internal transition changes its state alone, causing a synchronization loss for M_j. The rules defined above could be extended to a coupled model with an arbitrary number of atomic models in a straightforward manner. Note that the definition of weak synchronization between untimed models can be easily obtained by just eliminating timing information (ignoring elapsed time), that is, by replacing q_i by s_i from the above definition.

As an example, recall the railroad crossing model in Figure 4. At a total system state ((*TRAV*, 5.5), (*UP*, 5.5)), no internal/external transition is eligible. However, at another total system state ((*TRAV*, 90.5), (*UP*, $-$)), an internal transition of the train is eligible at the state (*TRAV*, 91.2), while no transition of the gate is eligible. The state (*UP*, $-$) of the gate would transit to the state (*DOWN*, 0) as soon as the controller performs its internal transition with an output, which is, in turn, converted to the input event *C?down*. Note that no synchronization can take place between the train and the gate since there is no coupling between the two.

6. Controller Design

6.1 Weak/Strong Path Controllability

This section briefly shows the result of a railroad crossing controller design problem using the DEVS supervisory control framework proposed in Song and Kim [18]. As will be seen in the subsequent section, the design procedure essentially is to deduce the desired state trajectories (or paths) from a composite plant model, which can be converted to a controller model in a straightforward manner with the concept of an inverse DEVS [15, 19]. To conclude ahead, we see that the plant *TG-PLANT* is weakly controllable with respect to the control objectives, as shown in the following. This means that the controller under design has to meet strict timing requirements as well as an event order to guarantee safe system operation. To begin with, we need to define the notion of DEVS *controllability* again. Given a discrete event plant $M =< X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ti >$, a state path $ST = s_0 s_1 \ldots s_{f-1} s_f$, $s_i \in S$ (or *path*) is a sequence of states of S that is in the order that is desirable to be executed by plant M. The terms used in the following definition are slightly modified from the original ones in Zeigler et al. [15] to be more definite.

DEFINITION 3. Weak path controllability. A path is weakly controllable if either an internal or external transition can transit each state to the next state on the path. Formally, a path ST is weakly controllable if, for all s_i, s_{i+1} follows s_i in ST, either

(a) $\delta_{\text{int}}(s_i) = s_{i+1}$, or

(b) there is a pair (e, x), where $0 < e < ta(s_i)$, $\delta_{\text{int}}(s_i) = s' \neq s_{i+1}$, and $x = p?m \in X$, such that $\delta_{\text{ext}}(s_i, e, x) = s_{i+1}$.

The states on a weakly controllable path never deviate from those specified in the path with a proper input event at an appropriate time.

DEFINITION 4. Strong path controllability. A weakly controllable path ST is strongly controllable if, for all s_i, s_{i+1} follows s_i in ST, either

(a) $\delta_{\text{int}}(s_i) = s_{i+1}$, or

(b) there is an input $x = p?m \in X$, such that $\delta_{\text{ext}}(s_i, e, x) = s_{i+1}$ for all $0 < e < ta(s_i) = \infty$.

This means that (1) the plant explores the desired states on the path by a sequence of internal transitions until it meets a state with an infinite time advance, or (2) at the state, an external transition is defined for the transition to the next desired state. Note in definition (b) that $ta(s_i) = \infty$ means that there is no internal transition eligible at the state (i.e., $\exists s' \in S\ \delta_{\text{int}}(s_i) = s'$) such that $ti(s_i) = [\infty, \infty]$. Thus, in that state, the controller has enough time to stimulate an input event, which changes a plant's state to the next desired state by an external transition. Otherwise, a path is called *uncontrollable*.

From the above definitions, we can derive path controllability for an overall system: a system model M is said to be strongly (weakly) controllable with respect to a state path $ST = s_0 s_1 \ldots s_{f-1} s_f, s_i \in S$ if and only if the state path is strongly (weakly) controllable. The framework for the design and implementation of a discrete event controller based on the above definitions can be found in Song [20] and Hong et al. [2].

6.2 Result from the Railroad Crossing Controller Design

Control of an untimed discrete event system model can be done by applying a correct sequence of control input events to the system that keeps the system in a desired execution order. The term *untimed* means that an input event or an output event could occur at an arbitrary time. Thus, only the strong controllability in the above definition can be considered in the untimed domain. In this section, we confine our concern to the design of an untimed control model.

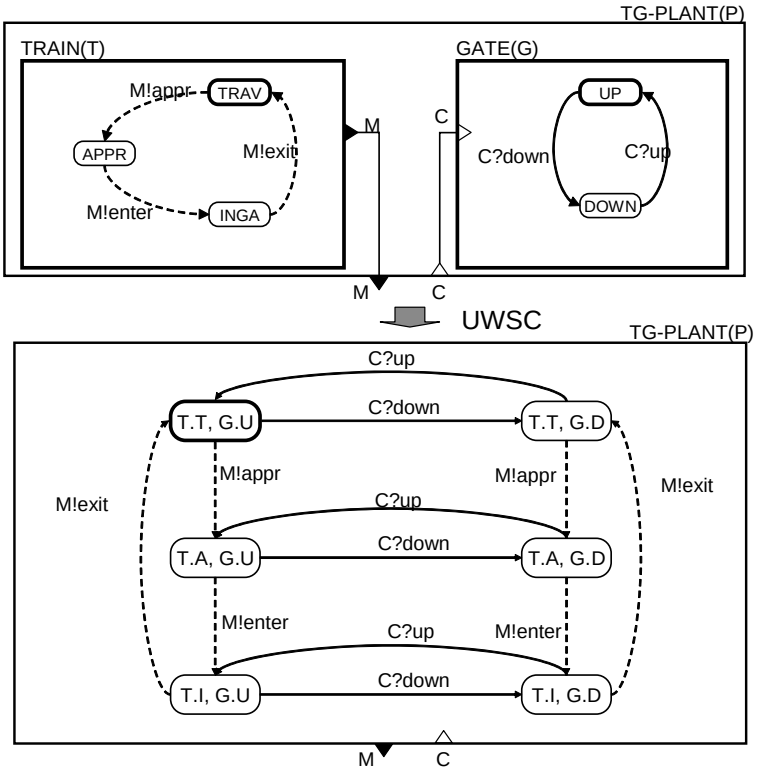


Figure 5. Untimed composite model of coupled model *TG-PLANT*. *T.T = T.TRAV*; *G.U = G* is at state *UP*; etc.

In the following, we employ the controller design method proposed in Song [20]. Given a given plant specification in Figure 4, we first obtain the untimed composite model of the plant *TG-PLANT*, as shown in Figure 5. Then, the next step finds desired state paths. The formal control objectives for safety and liveness, *SPEC*, in temporal logic form must satisfy the following two [14]:

$$SPEC.1 \Box \neg (T.INGATE \wedge G.UP)$$

$$SPEC.2. (T.TRAV \wedge G.DOWN) \rightarrow$$

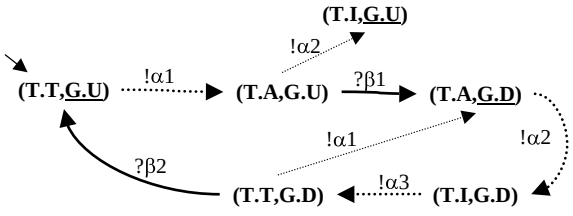
$$\Diamond (G.UP \wedge (\neg (\neg G.UPUT.APPR)))$$

The first logic claims that at any instance, the system should not be at a state where the train and the gate are at the crossing area (safeness). Second, if the gate is down and the train passes the gate, then the gate eventually opens for a car to pass it (liveness).

Following the procedure in Song [20], a set of desired paths that satisfy the given control objectives is obtained (Fig. 6):

$$ST_{desired} := \{[(T.T, G.U)(T.A, G.U)(T.A, G.D)$$

$$(T.I, G.D)(T.T, G.D)]^*\}.$$



!α1:M!appr !α2:M!enter !α3:M!exit ?β1: C?down ?β2: C?up

Figure 6. Controllability analysis of untimed composite model

Unfortunately, as shown in Figure 6, the path is not strongly controllable but weakly controllable. That is, a dangerous state $(T.I, G.U)$ would be reached from a desired state $(T.A, G.U)$ via an internal transition unless the control input β1 is applied before the transition. Thus, we conclude that the plant is not controllable in the untimed domain and that the state $(T.I, G.U)$ is a *bad* state in the sense of safety. Regardless of the result above, strong controllability may be achieved by modifying the plant by adding additional sensors and actuation points to the plant in an appropriate manner.

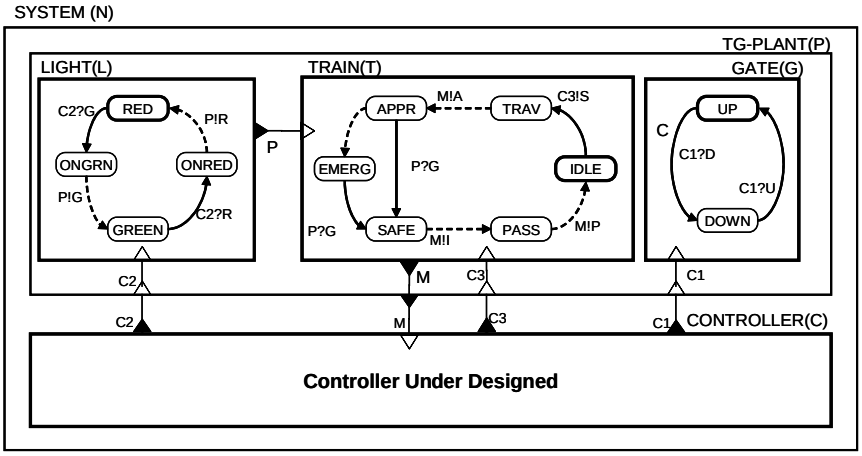


Figure 7. Modified plant system

Figure 7 shows a modified system model in which a traffic light model is added, and the train's behavior is modified so as to accept control input events from the traffic light model. By the similar steps above, a controller model can be obtained, as shown in Figure 8.

Note that the controller obtained in the untimed domain is robust to time specification; that is, the desired state path is guaranteed regardless of time specification. Thus, the DEVS framework for the controller design in the untimed domain is adequate for the control of hard real-time systems in which any synchronization loss can cause a disastrous effect. As a side effect, by the inclusion property of theorem 1 in Song [20] and Song and Kim [18], any timing constraint restricts the system's behavior. Therefore, it is possible for the designed system not to meet the control objectives in the sense of liveness, although it never affects the safeness property. This is because some desired path may not be executed by the time constraints. Thus, design of a weak controller requires the verification of liveness, which can be done by using the timed behavior analysis presented in the next section.

7. Timed Behavior Analysis

The original railroad crossing plant in Figure 4 is not strongly controllable in the untimed domain, meaning that no controller keeps the plant to the desired states without falling into some bad state. However, it is still a weakly controllable plant with respect to the desired path $ST_{desired}$. Thus, we still have an opportunity to control the plant by a timed controller, which generates control events at appropriate elapsed times. While excluding an automated algorithm to generate such times, this article focuses on safety analysis: given a timed controller, whether a control system is safe.

Figure 9 depicts a timed control system that consists of three atomic RT-DEVS models: *TRAIN*, *GATE*, and *CONTROLLER*. For the safety analysis of such a system, we employ an inductive methodology for analyzing the timed behavior of a closed coupled model under weak synchronization between components. We use term *timed* because it provides timing information associated with each state of a composed model. The method proposed here is called the TBA, which gives the maximal set of all timed behavior with minimal uncertainty from multiple models [18]. TBA does not produce just a sample path, as in simulation, but allows us to perform reachability analysis. In other words, it produces all possible sequences of timed events/states that a coupled model can generate. Two problems that make the analysis complicated are an infinite total state space induced by dense time intervals owing to a time interval function and the inherent uncertainty augmented by interval operations. To cope with the problems, we introduce a notion of vector time and a clock matrix in our analysis process. A notion of time representation that effectively deals with dense time intervals is given, followed by the TBA procedure.

7.1 Time System: Vector Time and Clock Matrix

This section introduces a clock mechanism and associated mathematical tools that allow us to predict the upper and lower bounds of the next transition time as precisely as possible. To begin with, we introduce a notion of an artificial clock of each machine (model). A clock of a machine is reset whenever an internal or an external state transition occurs. It then is increased continuously until its next transition, thus being a representation of an elapsed time at a state. In addition, every machine can observe the other machines' transitions only when their clocks are

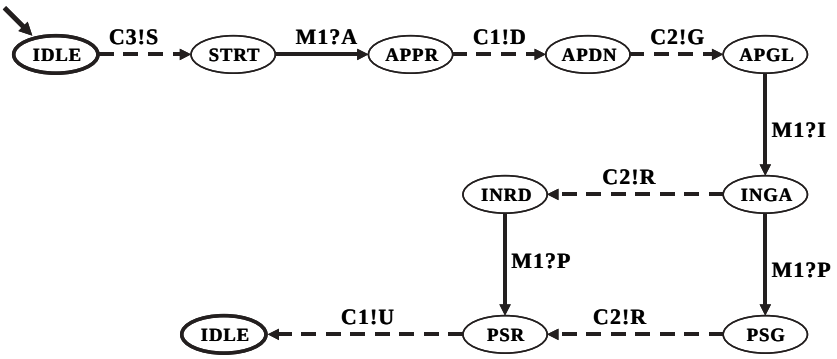


Figure 8. Controller for the modified plant

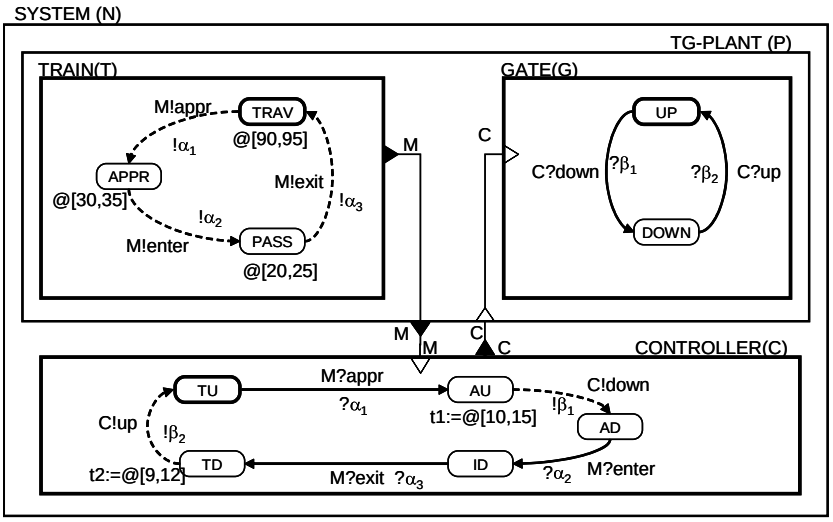


Figure 9. Timed control system for safety analysis

synchronized. Without synchronization, one machine never guesses another machine's total state (state and associated elapsed time), which is necessary for the modular modeling formalism.

To deal with real-valued interval time, we define arithmetic interval operations. A real interval (or just interval) $\bar{t}$ is defined by a compact subset of the field of real numbers, $\Re$, of the form $\bar{t} := [l, u] = \{t | l \leq t \leq u, l, u \in \Re\}$. The set of all closed real intervals is denoted by $I(\Re)$. Two intervals, $a = [a_1, a_2]$ and $b = [b_1, b_2]$, are called equal, $a = b$, if and only if $a_1 = b_1$ and $a_2 = b_2$. Binary operations are defined by (a) addition, $a+b := [a_1+b_1, a_2+b_2]$, and (b) subtraction, $a - b := [a_1 - b_2, a_2 - b_1]$. A single value $t \in \Re$ is regarded as $[t, t] \in I(\Re)$; thus, $t + a$ equals $[t, t] + [a_1, a_2]$. We say $a < b$ if $a_2 < b_1$. The size of an interval, $|a| = a_2 - a_1$, is called the uncertainty level. From now on, time is represented by a real-valued interval, and the terms *time* and *time interval* are to be interchangeable.

The notion of *interval vector time* is an extension of the system of vector time for monotonically increasing the nonnegative integer [19] to real intervals. In the system, each machine has its own artificial local clock that is reset whenever the machine encounters a state transition. Formally, an interval vector time v is a n-dimensional vector of real intervals; that is, for an interval vector time $v[1 \ldots n]$, $1 \leq i \leq n$: $v[i] \in I(\Re)$. A n-dimensional interval vector time domain is denoted by $VT^n \subseteq I(\Re)^n$. An element $v[i] \in I(\Re)$ represents a dense set of possible times of clock i. For example, consider a three-dimensional interval vector time $v_2 = [0.3, 0.5 \;\; 0, 0 \;\; 2, 4] \in VT^3$. The first element [0.3, 0.5] is the possible reading of clock 1, the second [0, 0] of clock 2 reset, and [2, 4] of clock 3. We can

see that machine 2 has just reset, and then the uncertainty level becomes zero. Thus, from machine 2's perspective, machine 1 has spent [0.3, 0.5] after its last transition, and the elapsed time of machine 3 at the current state is within [2, 4]. Recall that a vector time represents possible time ranges of clock readings at a single physical moment of time. Now we define the precedence relationship between vector times.

DEFINITION 5. Precedence of vector times. For n-dimensional vector times $v_1, v_2 \in VT^n$, v_1 is said to precede v_2, denoted by $v_1 \ll v_2$, if and only if

$$\exists m : v_1[m] < v_2[m] \; and \; \forall k \neq m : \neg(v_2[k] < v_1[m]), \quad k, m \in 1 \ldots n. \tag{3}$$

The definition states the following: if there exists at least a single clock that can determine the precedence relation of the two vector times, and there is no contradictory judgment from other clocks, we rely on the judgment of the clock. The key idea is to choose the earliest one among machines for the next internal transition. The precedence $v_1 \ll\gg v_2$ stands for vagueness to determine the time precedence.

To deal with multiple times as a group, it is convenient to stack vector times in a time matrix form. Let us define MT^n as the domain set of $n \times n$ time interval elements. For a time matrix $O[1 \ldots n, \; 1 \ldots n] \in MT^n$, each row $O[i, \cdot] \in VT^n$ is designated to the vector time of machine i. Thus, an element $O[i, j]$ represents the time of machine i in the view of clock j. We write just MT instead of MT^n if no confusion arises; similarly, we use VT instead of VT^n.

DEFINITION 6. Consistency of the time matrix. A time matrix $O \in MT$ is called consistent if and only if for all pairs $i, j = 1 \ldots n$, $i \neq j$, such that $v_i := O[i, \cdot]$, one of the three holds: (a) $v_i \ll v_j$, (b) $v_j \ll v_i$, or (c) $v_i \ll\gg v_j$.

We define a special class of a time matrix, called a (relative) *clock matrix*, that maintains the time differences in the view of each clock in a peer-to-peer fashion. For a clock matrix $O \in MT$, an element $O[i, j]$ implies that time i leads time j by $O[i, j]$ in the view of clock j. Moreover, its own time difference is always zero, $O[i, i] = [0, 0]$, as the definition states.

DEFINITION 7. Validity of the clock matrix. A clock matrix is said to be valid if and only if it holds the following properties:

(a) $O[i, i] = [0, 0]$

(b) $O[i, j] = -O[j, i]$

(c) There is a nonempty subset of clocks, $C_{lead}(O) \subseteq 1 \ldots n$, such that for all $i, k \in C_{lead}(O)$: $O[i, \cdot] = O[k, \cdot]$ and that for some $i \in C_{lead}(O)$ and for all $k \notin C_{lead}(O)$: $O[k, j]|_{\max} \leq O[i, j]|_{\min}$, $j \in 1 \ldots n$.

A clock $i \in C_{lead}(O)$ is called the most leading clock (MLC), and the vector time $v_{lead} := O[i, \cdot]$ is called the most leading time (MLT). Denote a subset $CT \subseteq MT$ to be a set of valid clock matrices, each element of which is valid. For example, a clock matrix

$$O = \begin{bmatrix} 0,0 & 15,25 & 0,0 \\ -25,-15 & 0,0 & -25,-15 \\ 0,0 & 15,25 & 0,0 \end{bmatrix}$$

is valid since it satisfies (a), (b), and $C_{lead}(O) = \{1, 3\}$ and $v_{lead}(O) = [\; 0,0 \quad 15,25 \quad 0,0]$. In particular, with $O[1, 2] = [15, 25]$, it is easy to see that time $O[1, \cdot]$ leads time $O[2, \cdot]$ by [15,25] in the view of clock 2. Conversely, $O[2, 1] = [-25, -15]$ indicates that time 2 is behind time 1 by [-25,-15] (say, –20). It specifies only the boundary, not an exact value. Clocks 1 and 3 are synchronized.

For each row i of a clock matrix, a vector time represents a relative time of the last transition of machine i in the view of each clock. The column j of a clock matrix indicates a set of the relative times of the last transitions of all machines relative to clock j. Since all diagonal elements are zeroes and the clock is reset at each transition, an element of a clock matrix shows the time difference between the two *last* transitions of two machines. For example, $O[1, 2] = [15, 25]$ means that the last transition of machine 1 takes place at time $t \in [15, 25]$ after the last transition of machine 2. Thus, it is an elapsed time for a composite state in a relative form.

7.2 Computing Timed Transitions

This section deals with a timed transition. Consider an externally *closed* coupled system $N = < X, Y, M, EIC, EOC, IC >$, where $M_i \in M$ is an atomic model $M_i = < X_i, A_i, Y_i, \delta_{\text{ext},i}, \delta_{\text{int},i}, \lambda_i, ti_i >$. Let $D = 1 \ldots n$ be a set of indices of models in M, where $n = |D|$. The following steps show the computation of timed transitions, which can be used for safety analysis.

7.2.1 Step 1: Find the Next Transition Time and Transition Machines

The first step is to find the next transition time of eligible candidate models at a composite total state $q = (s, O_L) \in S \times CT$. Our analysis maintains the two kinds of a time matrix: O_L for the last transition time and O_N for the next transition time. The initial last transition time O_L of a system is a zero-clock matrix whose elements are all zero, indicating that all clocks are synchronized. Let us start at a kth composite total state, $q^k := (s^k, O_L^k) \in Q_z \subseteq S \times CT$, where $s^k := (s_1^k, s_2^k, \cdots, s_n^k) \in S = \prod_{i \in D} S_i$, and CT is the set of all $n \times n$ valid clock matrices. Since $O_L^k \in CT$, there must be the most leading time $v_{lead}(O_L^k)$. We call q^k a timed state instead of a *total* state to emphasize the elapsed times in a clock matrix. From the timed state q^k, we can compute the $(k+1)$th transition times of each machine in D, denoted by $O_N^{k+1} \in MT$. A vector time $O_N^{k+1}[i, \cdot]$

represents the next transition time of machine i, which is determined by the current vector time $O_L^k[i, \cdot]$ plus the time to fire $ti_i(s_i^k)$ since then, given by

$$1 \le j \le n : O_N^{k+1}[i, j] := (O_L^k[i, j] + ti_i(s_i^k)) \cap [v_{lead}(O_L^k)[j]|_{\min}, \infty]. \quad (4)$$

The intersection of two vector times is a vector time, each element of which is obtained by an intersection of corresponding elements. The clipping operation of the right term, $[v_{lead}(O_L^k)[j]|_{\min}, \infty]$, is subject to causality in the real world; that is, the next transition time should be greater than or equal to the last transition time in the system. Actually, the time $O_N^{k+1}[i, \cdot]$ designates the time left to the next transition of machine i in the view of each clock since the last transition time $O_L^k[i, \cdot]$ of state s_i^k, if there is no input event to the machine. For example, from the perspective of clock j, an internal state transition of machine i will occur at an instance of time within $O_N^{k+1}[i, j]$.

LEMMA 1. If O_L^k is consistent, then time matrix O_N^{k+1} obtained by (4) is also consistent.

Proof. We can easily reach this result by the definition of the addition operation of intervals.

An internal state transition should occur within $ti(s)$ (i.e., $ti(s)|_{\min} \le ta(s) \le ti(s)|_{\max}$), with no external input event by then. Thus, the time span of the next transition time is given by the minimum of the next transition times in each column, which gives a vector time $v_{N,max}^k[1...n] \in VT$:

$$1 \le j \le n : v_{N,max}^k[j] := \min_{i \in D}(O_N^k[i, j]), \quad (5)$$

where $\min(v_1, v_2, \ldots, v_n) = [\min(l_1, l_2, \ldots, l_n), \min(u_1, u_2, \ldots, u_n)]$, for $v_i := [l_i, u_i] \in I(\Re)$. Time $v_{N,max}^k[j]$ means the least time interval among the next transition times in the view of clock j.

LEMMA 2. The time span in (5) is the maximum interval within which at least one internal transition occurs in the system.

Proof. In the view of clock $j \in D$, the earliest next internal transition time is $vt_{N,max}^k[j]|_{\min}$. Thus, for all $i \in D$, $O_N^{k+1}[i, j]|_{min} \ge vt_{N,max}^k[j]|_{min}$. At the other extreme, there is at least one machine $m \in D$ such that $O_N^{k+1}[m, j]|_{\max} = v_{N,\max}^k[j])|_{\max}$. Machine m has to perform an internal transition until that time, or it violates the system specification. These two factors support the lemma.

Lemma 2 states that at least a machine has to perform an internal transition at a time within the maximum interval $v_{N,max}^k$. It gives us a *valid* next vector time at which at least one machine fires. For all $i \in D$, we write

$$1 \le j \le n : O_{N,max}^k[i, j] := O_N^k[i, j] \cap v_{N,max}^k[j]. \quad (6)$$

With this, we can determine the candidate machines that have a possibility to perform an internal transition at q^k.

$$CAND^k := \{i \in D | O_{N,max}^k[i, j] \ne \emptyset, \forall j \in D\}. \quad (7)$$

The rationale is that a candidate machine for the next internal transition is selected when all clocks agree that at least an internal transition would take place by the time.

EXAMPLE 1. Consider the railroad crossing system shown in Figure 9 and its final result from TBA analysis in Figure 11. To show how TBA works, we assume in the subsequent examples that the system is at a composite state $q_4 = (a_4, O_{L,4})$, where $a_4 = (T, D, TD)$, and its clock matrix $O_{L,4}$ is given by

$$O_{L,4} = \begin{bmatrix} 0, 0 & 35, 50 & 0, 0 \\ -50, -35 & 0, 0 & -50, -35 \\ 0, 0 & 35, 50 & 0, 0 \end{bmatrix}.$$

For instance, the first column represents the last transition times of the train, the gate, and the controller, respectively, in the view of the train model. From q_4 and time intervals at states T, D, and TD in Figure 11, we can obtain $O_{N,4}$ by using equation (4):

$$O_{N,4} = \begin{bmatrix} 90, 95 & 125, 145 & 90, 95 \\ \infty & \infty & \infty \\ 9, 12 & 44, 62 & 9, 12 \end{bmatrix}.$$

Application of (5) to $O_{N,4}$ results in a vector time $v_{N,4,\max} = \begin{bmatrix} 9, 12 & 44, 62 & 9, 12 \end{bmatrix}$. Combining $v_{N,4,\max}$ and $O_{N,4}$ with equation (6), we can get

$$O_{N,4,\max} = \begin{bmatrix} \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset \\ 9, 12 & 44, 62 & 9, 12 \end{bmatrix}.$$

Note that there is only one candidate model for the next transition: the controller model C; that is, we have $CAND_4 = \{C\}$ by equation (7), with $O_{N,4,\max}[C, \cdot] = \begin{bmatrix} 9, 12 & 44, 62 & 9, 12 \end{bmatrix}$.

7.2.2 Step 2: Weak Synchronous Composition

The second step is to find machines, for each output event from a candidate, that synchronously perform external transitions with the associated output events; however, if there is no such machine, the candidate fires alone. A machine in the candidate set $m \in CAND^k$ has at least one internal transition defined at a timed state $q^k := (s^k, O_L^k) \in Q_z$, where $s^k := (s_1^k, s_2^k, \cdots, s_n^k) \in S$. In general, a set of feasible internal transitions of machine $M_i =< X_i, A_i, Y_i, \delta_{\text{ext},i}, \delta_{\text{int},i}, \lambda_i, ti_i >$ on a state $s \in S_i$ is defined as follows:

$$INT_i(s_i) := \{(s_i, y_i', s_i') \in S_i \times Y_i \times S_i | \delta_{int,i}(s_i, y_i') = s_i' \text{ is defined}\}. \quad (8)$$

Application of equation (8) obtains $INT_m(s_m^k)$, a set of internal transitions on a state $s_m^k \in S_m$ of each machine $m \in CAND^k$. Recall that an internal transition $(s_m^k, y_m^{k+1}, s_m^{k+1}) \in$

$INT_m(s_m^k)$ generates an output event $y_m^{k+1} \in Y_m$, which is immediately transmitted to other machines connected by coupling relations. Weak synchronization rules in definition 2 can provide us with a set of machines that synchronously perform external transitions stimulated by the output event $y_m^{k+1} \in Y_m$ at the timed state q^k as

$$SYNC_m^k(y_m^{k+1}) := \{j \in D | \delta_{ext,j}(s_j^k, e_j, x_j^k) = s_j^{k+1} \text{ and } (y_m^{k+1}, x_j^k) \text{ matches}\}. \quad (9)$$

We can ignore elapsed times of the influenced machines as the following lemma states.

LEMMA 3. Every machine $p \in SYNC_m^k(y_m^{k+1})$, $m \in CAND^k$, can synchronously perform an external transition due to the matching input of $y_m^{k+1} \in Y_m$ within the valid next transition time $O_{N,max}^k[m, \cdot]$.

Proof. The next transition time $O_N^{k+1}[p, j]$ is greater than or equal to the upper bound of the span $v_{N,max}^k[j]|_{max}$ by equation (5); for all clocks $j \in D$, it is obvious that an external state transition is always defined within the next transition time $O_{N,max}^k[p, \cdot]$. Thus, when the matching event is received from a sending machine m, the elapsed time of a receiving machine p is less than or equal to its upper time bound at its current state.

By lemma 3, we can get a timed state $q^{k+1} = (s^{k+1}, O_L^{k+1}) \in Q_z$ from $q^k = (s^k, O_L^k) \in Q_z$ using the weak synchronization mechanism as every internal transition of $m \in CAND^k$ and associated external transitions in $p \in SYNC_m^k(y_m^{k+1})$ are eligible up to time $O_{N,val}^k[m, \cdot]$. First, the new composite state $s^{k+1} = (s_1^{k+1}, s_2^{k+1}, \cdots, s_n^{k+1}) \in S$ can be obtained as follows. For all $m \in D$ and $(s_m^k, y_m^{k+1}, s_m^{k+1}) \in INT_m(s_m^k)$,

(a) $s_m^{k+1} = \delta_{int}(s_m^k, y_m^{k+1})$;

(b) $s_p^{k+1} = \delta_{ext}(s_p^k, -, x_p^{k+1})$, for all $p \in SYNC_m^k(y_m^{k+1})$ and (y_m^{k+1}, x_p^{k+1}) matches; and

(c) $s_r^{k+1} = s_r^k$ for all $r \in D - SYNC_m^k(y_m^{k+1}) \cup \{m\}$.

By the definition of weak synchronization, we can see that elapsed times of machines in $SYNC_m^k(y_m^{k+1}) \cup \{m\}$ reset to zeroes because they all undergo state transitions. Thus, the new elapsed time O_L^{k+1} will be adjusted for the new state s^{k+1}, as shown in the following step.

EXAMPLE 2. Considering the result of example 1, the candidate set has only one model, $CAND_4 = \{C\}$. The feasible internal transition set of controller C at state $q_4 = (a_4, O_{L,4})$, $a_4 = (T, D, TD)$ can be easily obtained from the controller model in Figure 9 as $INT_C(TD) = \{(TD, !\beta 2, TU)\}$. By the coupling relation, we see that $SYNC_{C,4}(\beta) = \{G, C\}$ through event $\beta 2$. That is, the controller and the gate will synchronize together through event $\beta 2$, and the resultant composite state will be $q_5 = (a_5, O_{L,5})$, where $a_5 = (T, U, TU)$. That is, $(T, D, TD) \xrightarrow{\#\beta_{(C\#up)}} (T, U, TU)$, where $\#\beta$ denotes the synchronized event. Clock matrix $O_{L,5}$ of the state q_5 can be computed by the following step.

7.2.3 Step 3: Computing the New Clock Matrix

Note that a clock matrix records a relative time, leads, or lags in the view of each machine's eye. Thus, if one of the elements is updated, then other related time elements should be updated accordingly. This section defines the update rules to maintain consistency and correctness in the clock matrix. We can find time O_L^{k+1} from O_L^k and $O_{N,max}^{k+1}$ in equation (4).

PROPOSITION 1. For $m \in CAND^k$, let $SYNC := SYNC^k(y_m^{k+1}) \cup \{m\}$ and $SYNC^c := D - SYNC$ be a set of synchronous machines and its conjugate set, respectively. Then, the $(k+1)$th elapsed time O_L^{k+1}, obtained by the following four rules, is semantically consistent.

$$\forall i \in SYNC, j \in SYNC \quad : \quad O_L^{k+1}[i, j] := [0, 0] \quad (\textit{synchronization}) \quad (10)$$

$$\forall i \in SYNC^c, j \in SYNC^c \quad : \quad O_L^{k+1}[i, j] := O_L^k[i, j] \quad (\textit{unchanged}) \quad (11)$$

$$\forall i \in SYNC, j \in SYNC^c \quad : \quad O_L^{k+1}[i, j] := O_{N,max}^{k+1}[m, j] \quad (\textit{advanced}) \quad (12)$$

$$\forall i \in SYNC^c, j \in SYNC \quad : \quad O_L^{k+1}[i, j] := -O_{N,max}^{k+1}[m, i] \quad (\textit{normalization}) \quad (13)$$

Proof. Equation (10) comes from the fact that after an activity transition occurs, the clocks of machines in $SYNC$ reset to zeroes by either internal transitions (machine m) or external transitions (machine in $SYNC - \{m\}$). Equation (11) holds because there has been no transition in machines in $SYNC^c$, and neither does the reference time of clock $j \in SYNC^c$. Thus, the transition times of the machines in the view of their local clocks are invariant. In the view of clock $j \in SYNC^c$, it is true that clock $i \in SYNC$ increments up to $O_{N,max}^k[m, j]$ in equation (4). Finally, recall that the launching time of a machine $i \in SYNC^c$ is unchanged, whereas clock $j \in SYNC$ is newly updated to [0,0], even though it really advances by $O_{N,max}^k[m, j]$. Therefore, the last transition time of machine $i \in SYNC^c$ should lag relatively behind by the time advance $O_{N,max}^k[m, j]$ in the view of clock j, which gives equation (13).

LEMMA 4. A clock matrix O_L^{k+1} obtained by proposition 1 is *valid*.

Sketch of proof. The validity conditions (a) and (b) in definition 7 can be easily proved by four equations in the

proposition. Furthermore, we have $C_{lead}(O_L) = SYNC$ and $v_{lead}(O_L^{k+1}) = O_L^{k+1}[m, \cdot] = O_L^{k+1}[j, \cdot]$, for all $j \in C_{lead}(O_L)$. It can easily be deduced that $O_L^{k+1}[k, j]|_{\max} \leq O_L^{k+1}[m, j]|_{\min}$ from the four cases in the proposition and the causality constraint in equation (4), which proves (c). It completes the proof.

Now consider elements of a clock matrix, $O_L^{k+1}[i, j]$. It is certain that an element (i, j) of a clock matrix records not only the clock differences between clocks i and j but also the last transition time of machine i in the view of clock j. Now that we have obtained a new starting time, we have finished calculating a transition from $q^k = (s^k, O_L^k)$ to $q^{k+1} = (s^{k+1}, O_L^{k+1})$ with an event y_m^{k+1} and the next transition time $O_{N,max}^{k+1}$ from the timed state q^k. The timed behavior algorithm is a repetition of the above three steps: (1) find the next transition time and candidates, (2) do the state composition, and (3) calculate the elapsed time at the time of the transition.

EXAMPLE 3. Recalling example 2, we have the state $q_5 = (a_5, O_{L,5})$, where $a_5 = (T, U, TU)$, and the clock matrix $O_{L,5}$ of the state q_5 can be computed by proposition 1. From example 1, we obtained vector time $O_{N,4,\max}[C, \cdot] = \begin{bmatrix} 9,12 & 44,62 & 9,12 \end{bmatrix}$, which means the internal transition time of the controller at the current state q_4. By proposition 1, with $SYNC_{C,4}(\beta) = \{G, C\}$ and $O_{N,4,\max}[C, \cdot]$, we can get

$$O_{L,5} = \begin{bmatrix} 0,0 & -12,-9 & -12,-9 \\ 9,12 & 0,0 & 0,0 \\ 9,12 & 0,0 & 0,0 \end{bmatrix}.$$

Note that the diagonal elements are always $[0, 0]$ by the definition of the clock matrix. The rest are obtained as follows. By the synchronization rule (10), we have clock matrix $O_{L,5}[i, j] = [0, 0]$, where $i, j \in SYNC_{C,4}(\beta) = \{T, G\}$. For $i \in SYNC_{C,4}(\beta) = \{G, C\}$, $j \in SYNC_{C,4}^C(\beta) = \{T\}$ (first column), and $O_{L,5}[i, j] = O_{N,4,\max}[C, j] = [9, 12]$ using the advanced rule (12). Finally, for $j \in SYNC_{C,4}(\beta) = \{G, C\}$, $i \in SYNC_{C,4}^C(\beta) = \{T\}$ (first row), and $O_{L,5}[i, j] = -O_{N,4,\max}[C, j] = [-12, -9]$ using the normalization rule (13).

7.3 Timed Behavior Analysis

This section sums up parts of the previous sections and composes a procedure to analyze the maximal timed behavior of a closed component model with minimal uncertainty. The idea is relatively simple: full state exploration based on the spontaneous internal transitions as long as time constraints permit. The timed behavior analysis algorithm gives us a timed reachability graph from a closed RT-DEVS model. Let $N = < X, Y, M, EIC, EOC, IC >$ be a closed system where each machine $M_i \in M$ is an atomic model $M_i = < X_i, A_i, Y_i, \delta_{\text{ext},i}, \delta_{\text{int},i}, \lambda_i, ti_i >$. The timed behavior of a closed coupled model N is a set of timed event-state sequences, $T(N) \subseteq Q_z \times (\bar{\Sigma} \times Q_z)^*$, where $\bar{\Sigma} \in \cup_{i \in D} Y_i \times VT$ is a set of timed events, and $Q_z \subseteq S \times CT$ is a set of timed states.

Algorithm TBA. Timed behavior $T(N)$ of a closed system N is obtained by induction on the length $n = |\omega|$, $\omega \in T(N)$:

1. $n = 0$: Obviously, $\bar{q}^0 := (s^0, O_L^0) \in T(N)$, where $s^0 := (s_1^0, s_2^0, \cdots, s_n^0) \in S$, and a clock matrix $O_L^0 := \mathbf{0}$ is zero matrix whose elements are all $[0, 0]$s.
2. $n = k$: Assume that $\bar{\omega}^k := \bar{q}^0 \bar{\sigma}^1 \bar{q}^1 \cdots \bar{\sigma}^k \bar{q}^k \in T(N)$ is a timed behavior where $\bar{q}^k := (s^k, O_L^k)$.
3. Determine $O_{N,max}^{k+1}$ and $CAND^k$ from $\bar{q}^k := (s^k, O_L^k)$, $s^k := (s_1^k, s_2^k, \cdots, s_n^k)$.
4. $n = k + 1$:$\forall m \in CAND^k$: for each $y_m^{k+1} \in Y_m$, $(s_m^k, y_m^{k+1}, s_m^{k+1}) \in INT_m(s_m^k)$:

$$\bar{\omega} := \bar{\omega}^k(\bar{\sigma}^{k+1}\bar{q}^{k+1}) \in T(N),$$

$$\bar{\sigma}^{k+1} := (\sigma^{k+1}, v^{k+1}) \in \bar{\Sigma},$$

$$\bar{q}^{k+1} := (s^{k+1}, O_L^{k+1}) \in Q_z,$$

where $\sigma^{k+1} = y_m^{k+1}$, $v^{k+1} := O_{N,max}^k[m, \cdot]$, $s^{k+1} := (s_1^{k+1}, s_2^{k+1}, \cdots, s_n^{k+1}) \in S$:

(a) $s_m^{k+1} = \delta_{\text{int}}(s_m^k, y_m^{k+1})$;
(b) $s_p^{k+1} = \delta_{\text{ext}}(s_p^k, -, x_p^{k+1})$, for all $p \in SYNC_m^k(y_m^{k+1})$ and (y_m^{k+1}, x_p^{k+1}) matches;
(c) $s_r^{k+1} = s_r^k$, for all $r \in D - SYNC_m^k(y_m^{k+1}) \cup \{m\}$;

and O_L^{k+1} can be obtained by proposition 1 from $SYNC_m^k(\sigma^{k+1})$, O_L^k, and $O_{N,max}^k$.

This analysis method consists of essentially two parts: untimed behavior analysis and timing analysis. Only the candidate machines restrict the state transitions of machines that have not reached their firing times. Thus, it is true that TBA restricts an untimed behavior by the time constraints. Although this general algorithm gives us the maximal behavior with the minimum uncertainty, it cannot avoid the state explosion problem that is common in the general composite reachability analysis of state machines. Another limitation comes from the assumption that a system is closed in which no input events to the system are defined. However, we can extend the closed system to an open one by adding an artificial environment model, but that is beyond the scope of this article.

8. Safety Analysis

Safety analysis of a safety-critical real-time system model in RT-DEVS is one of application of the TBA in the previous section. Safety analysis is necessary to guarantee that the time constraints on a designed untimed controller, which may deviate from a correct to a bad state path in the case of an incorrect timing specification, are safe.

Due to the untimed behavior analysis algorithm proposed in Song [20], we can easily get a reachability graph of an untimed model of a system. As a timed behavior is usually a subset of untimed behavior [18], once a system is proved to be safe in untimed safety analysis, it is automatically safe in the timed safety analysis. Timed behavior analysis, however, requires more time and space complexity than the untimed one. We first show an untimed safety analysis and then present the application result to the timed safety analysis.

8.1 Safety Definition

We define *safety* as a system that does not go into any *bad* state. For example, state $(T.I, G.U)$ is a bad state because of the possibility of it having a disastrous accident (due to *SPEC*.1). Thus, safety analysis of RT-DEVS models involves a timed behavior analysis based on weak synchronization, which explores all the timed states that the system undergoes. In a reachability analysis, once it reaches a bad (or unsafe) state such as $B = \{(T.I, G.U)\}$, the system turns out to be unsafe, and the exploration stops.

8.2 Untimed Safety Analysis

Consider again a controller shown in Figure 9, where time constraints are specified manually. By applying untimed behavior analysis based on the weak synchronization mechanism to the model, we can get a partial untimed reachability graph of the whole system, as shown in Figure 10. A node represents a composite state of the train, the gate, and the controller, in that order, and an edge designates a transition with a firing event annotated. From the initial state (T, U, TU), where the train is at state T, the gate U, and the controller TU, only the train has a spontaneous (or internal) transition by which an output event α_1 is produced. Then, due to the coupling relation, the event is translated to an input event of the controller, which causes a simultaneous external transition, thus leading to a new composite state (A, U, AU). At this new state, both the train and the controller have internal transitions that produce respective outputs α_2 and β_1. If the controller fires first, the output event β_1 is wired to the gate waiting for the event, which entails a simultaneous transition to the state (A, D, AD); meanwhile, if α_2 fires earlier, the system will be at the state (I, U, AU), a bad state in B. This proves that the system is not safe anymore, and the exploration may stop also. As a consequence, in the untimed safety analysis, the system specified in Figure 9 is proved to be unsafe. Nevertheless, an assignment of appropriate timing constraints could prevent the state (A, U, AU) from moving to the bad state (I, U, AU).

8.3 Timed Safety Analysis

Consider a sample timing specification on the states shown in Figure 9. The train approaches after traveling for about [90,95] time units; then, around [30,35] time units later,

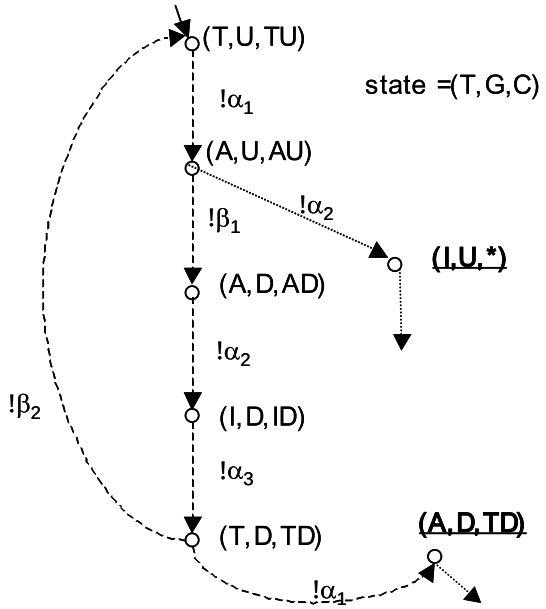


Figure 10. Untimed reachability graph of *N* in Figure 9

the train stays at the crossing area and finally leaves the area in [20,25] time units. The gate always awaits and performs a transition upon receiving control inputs from the controller. When the controller receives an event $M?appr$ from the train, it commands the gate to close after about $t_1 = [10, 15]$ time units; then, it observes the train until the train leaves. After that, it demands the gate to open around $t_2 = [9, 12]$ time units later. Could this timing specification prevent the system from falling into bad states? By the TBA, we can verify whether this is a correct timing specification. If it is not correct, we must choose another value of t_1 and t_2.

Figure 11 depicts the resultant reachability graph, another illustration of the timed behavior obtained by applying TBA to the railroad control system shown in Figure 9. A node in the graph represents a timed state $q_i := (a_i, O_{L,i})$, and a timed transition is represented by a dotted arc with a timed event annotated, a pair of events, and the next transition time $\sigma_i := (y_m, O_{N,max}(m, \cdot))$, where the machine $m \in D$ is one with an initiative internal transition transmitting an output event y_m.

Fortunately, we can see in the figure that the system never falls into a bad state $(I, U, -)$, owing to a correct timing of controller inputs, which was not in the untimed case. For example, the controller sends the output event $C!down$ around [10,15] at the timed state q_1, to which the controller transmits with a synchronous event $M\#appr$ generated from the train at the initial state q_0. It is interesting to see the same thing in the gate's view; for transition $q_1 \rightarrow q_2$, the gate receives input $C?down$ around [100,110] time units after the gate's last transition (no transition since

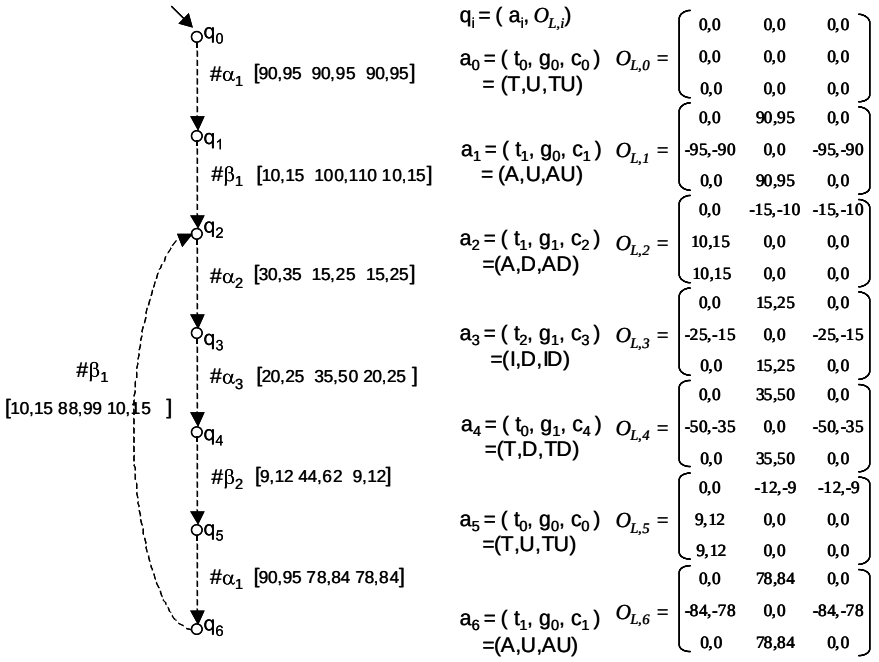


Figure 11. Timed reachability graph by TBA: *T(N)*

the start), while the controller thinks it is around [10,15] time units after the train and controller's last transition.

Regarding the control objectives, recall that in Figure 6, the desired state path is weakly controllable since there are external transitions, $(T.A, G.U) \xrightarrow{?\beta_1} (T.A, G.D)$and $(T.T, G.D) \xrightarrow{?\beta_2} (T.T, G.U)$. The plant's execution would be kept in the desired state path if the input events ?β1 and ?β2 occur before output events !α2 and !α1, respectively. It is a typical example that control depends on both the events' sequence and the times associated with the events for safety and liveness.

For an intuitive understanding of TBA, a *projection* operation is employed. The operation extracts only the behavior of the plant (*TG-PLANT*) from the whole behavior. This can be done in an easy and straightforward way, which is not described in this article (see [18]). Figure 12 shows the resultant projected behavior $T(N) \downarrow \{T, G\}$ to the train and the gate model, T and G.

Consequently, we can see that the timing specification has been chosen so successfully that the controller can control the plant as desired. In particular, at timed states q_1 and q_6, the control input $?\beta_1$proceeds $!\alpha_2$, and $?\beta_2$ happens before $!\alpha_1$ at q_4, which prohibits the system from executing undesired activities. Therefore, we find that an uncontrollable state path in the untimed domain can be controlled in the time domain by choosing appropriate control timing.

8.4 Remarks

The time assignment problem that allocates an appropriate timing for each state of a controller in RT-DEVS is an open question, especially for weakly controllable paths. Although it is out of the scope of this study, it could be solved by methods such as linear programming [21]. With this assumption, we can briefly summarize the design steps in the time domain based on TBA as follows. Let N be a control system with a plant P specified and a controller unspecified.

1. Use TBA to get $T(P)$ and $U(P)$ of the plant, the untimed behavior.
2. Apply the untimed design steps to get a set of the desired state path, $ST \subseteq U(P)$.
3. Get an untimed controller by the inverse transformation for ST.
4. Assign time variables to active activities of the untimed controller and find a solution.
5. Confirm safety and liveness by applying TBA again to the controlled system with time specified.

The design steps in the time domain will be completed when we find a way of solving the time assignment problem; otherwise, one can solve it by an ad hoc method. The

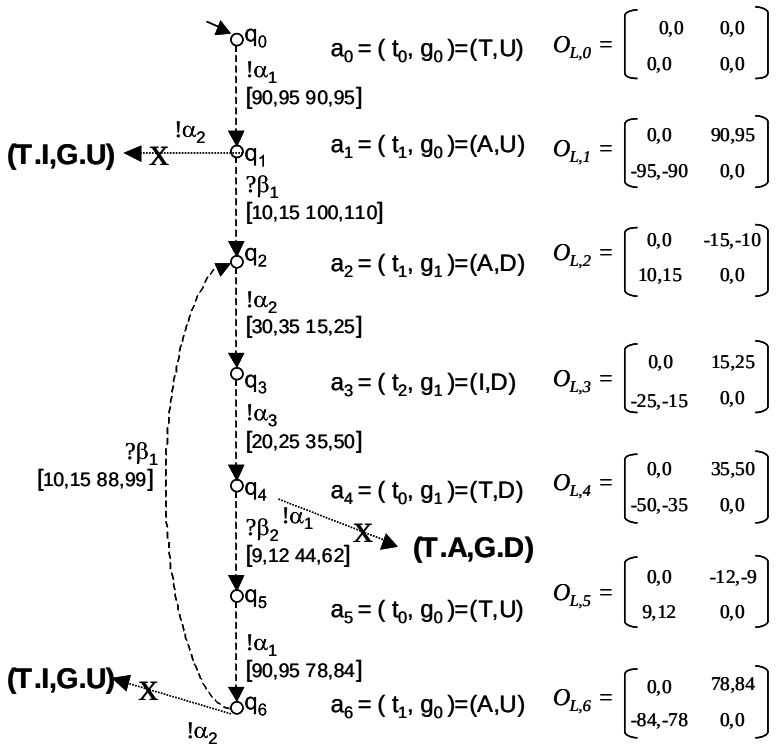


Figure 12. Projection of system *N* to plant: *T(N)*↓{*T,G*}

final step, checking safety and liveness, is also required for the untimed design steps when time is specified. Note that if one needs a robust controller against time, the untimed design steps can be used.

Finally, we briefly remark on the correctness of TBA, which can be dictated by the projection operation to individual machines [20, 21]. If the behavior obtained by the projection operation to a machine is a subset of the whole behavior of the machine, we shall say that TBA is correct.

Figure 13 shows the resultant behaviors of machines—T, G, and C, respectively. First consider the gate behavior $T(N) \downarrow G$, shown on the left-hand side of the figure. Compared to the gate model in Figure 9, the behavior is a subset of the original since the trajectory (state-event sequence) is the same as the original, and the transition times are a subset of the original time $[0,\infty]$. It is interesting to note that the train's behavior in the center figure is the same as the original one, that is, $T(N) \downarrow T \equiv T(T)$. Likewise, the output times of the controller's behavior on the right-hand side of the figure are the same as the original one. Note that the system shows cyclic behavior when it reaches steady states after some transient behavior. This comes from the artificial assumption that, initially, all the machines start at the same time. From these facts, we claim that the proposed TBA produces correct results.

9. Conclusion

This article has dealt with an application of real-time DEVS to the analysis of a real-time discrete event control system in two aspects: a controller design with an untimed model and the safety analysis of the designed controller with a given timing specification. For the analysis, a synchronization mechanism for communicating RT-DEVS models has been defined as a weak synchronization. We reviewed the untimed controller design framework using a weakly controllable path for the railroad crossing control, which was then extended to the timed design framework. To confirm the safety of a timed controller, we proposed precise notions of a vector time and a clock matrix for a reachability analysis algorithm TBA based on weak synchronization. TBA has been effectively applied to the railroad crossing control problem. The notion is to overcome the inherently increasing uncertainty induced by the dense time interval operations in the analysis. We have shown that TBA can generate a de facto finite state space of RT-DEVS models using the vector time and the clock matrix, although the total state space is inherently infinite. However, the space complexity is rather high due to the size of the clock matrices in its representation. Indeed, being based on the DEVS framework, the control analysis approach in this

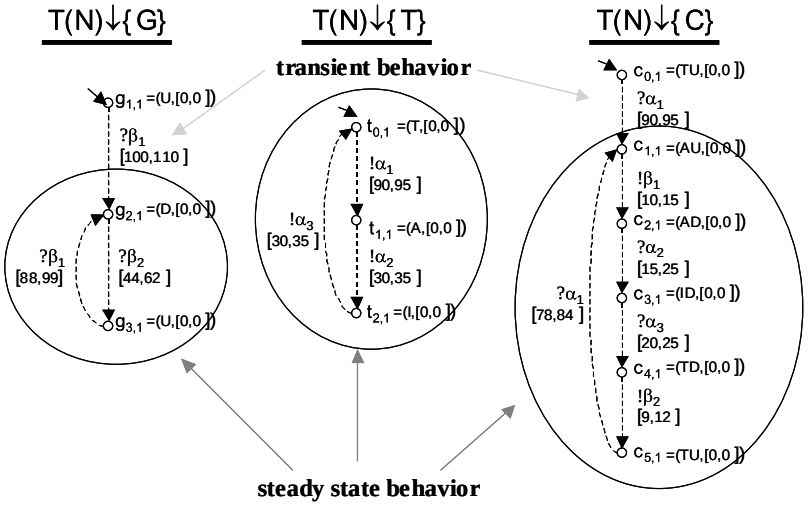


Figure 13. Projection of timed behavior. Left: projected to gate; middle: projected to train; right: projected to controller.

article is different from other approaches in that the same model can be used for all phases of controller design and implementation: modeling, analysis, performance evaluation, and virtual prototyping. Under preparation is further work that deals with details of TBA [18]. Finally, the automatic timing assignment problem for an untimed controller with a weakly controllable path is still open for study.

10. References

[1] Ostroff, J. S. 1992. Formal methods for the specification and design of real-time safety critical systems. *Journal of Systems Software* 18:18-33.

[2] Hong, J. S., H. S. Song, T. G. Kim, and K. H. Park. 1997. A real-time Discrete Event System Specification formalism for seamless real-time software development. *Discrete Event Dynamic Systems* 7 (4): 355-75.

[3] Ramadge, R. J., and W. M. Wonham. 1987. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization* 25 (1): 206-30.

[4] Heymann, M., and F. Lin. 1998. Discrete event control of nondeterministic systems. *IEEE Transactions on Automatic Control* 43 (1): 3-17.

[5] Heymann, M. 1990. Concurrency and discrete event control. *IEEE Control Systems Magazine* 10 (4): 103-12.

[6] Shayman, M. A., and R. Kumar. 1995. Supervisory control of nondeterministic systems with driven events via prioritized synchronization and state path models. *SIAM Journal of Control and Optimization* 33 (2): 469-97.

[7] Brandin, B. A., and W. M. Wonham. 1994. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control* 39 (2): 329-42.

[8] Ostroff, J. S. 1990. Deciding properties of timed transition models. *IEEE Transactions on Parallel and Distributed Systems* 1 (2): 206-30.

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

[10] Kang, I., I. Lee, and Y. S. Kim. 2000. An efficient state space generation for the analysis of real-time systems. *IEEE Transactions on Software Engineering* 26 (5): 453-77.

[11] Sifakis, J., S. Tripakis, and S. Yovine. 2003. Building models of real-time systems from application software. *Proceedings of the IEEE* 91 (1): 100-11.

[12] Kim, T. G. 1994. *DEVSim++ user's manual*. Taejon, Korea: SMSLab, Department of EECS, KAIST. http://smslab.kaist.ac.kr

[13] Cho, S. M., and T. G. Kim. 2001. Real time simulation framework for RT-DEVS models. *Transactions of the Society for Modeling and Simulation International* 18 (4): 178-90.

[14] Hong, G. P., and T. G. Kim. 1996. A framework for verifying discrete event models within a DEVS-based system development methodology. *Transactions of the Society for Computer Simulation International* 13 (1): 19-34.

[15] Zeigler, B. P., H. S. Song, T. G. Kim, and H. Praehofer. 1995. DEVS framework for modelling, simulation, analysis, and design of hybrid systems. In *Hybrid systems*, LNCS 999, edited by Pano Antsaklis et al., 529-55. Berlin: Springer Verlag.

[16] Song, H. S., and T. G. Kim. 1994. The DEVS framework for discrete event systems control. In *Proceedings of AI, Simulation and Planning in High Autonomy Systems*, Gainesville, FL, pp. 228-34.

[17] Hoare, C. A. R. 1985. *Communicating sequential processes*. Englewood Cliffs, NJ: Prentice Hall.

[18] Song, H. S., and T. G. Kim. Timed reachability analysis of real-time DEVS models using clock matrix. In preparation.

[19] Alefeld, G., and J. Herzberger. 1983. *Introduction to interval computations*. New York: Academic Press.

[20] Song, H. S. 2000. A DEVS framework for analysis and design of discrete event systems control. Ph.D. diss., KAIST, Taejon, Korea.

[21] Zeigler, B. P., and S. Chi. 1992. Symbolic Discrete Event System Specification. *IEEE Systems, Man, and Cybernetics* 22 (6): 1428-39.

Hae Sang Song *is an assistant professor in the Department of Computer Information & Communication at Seowon University, Cheongju, Korea.*

Tag Gon Kim *is a professor in the Department of Electrical Engineering & Computer Science at KAIST, Taejon, Korea.*