Methodology

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
1–19

DOI: 10.1177/0037549715604720
sim.sagepub.com
SAGE SCS

Action-level real-time DEVS modeling and simulation

Hessam S Sarjoughian* **and Soroosh Gholami**

Abstract

For some classes of systems, it is advantageous to develop real-time models instead of step-wise or logical-time models. Toward this goal, the action-level real-time (ALRT) discrete-event system specification (DEVS) modeling and simulation approach is proposed. Modeling of actions is introduced into the parallel DEVS formalism using time invariants defined for real-time statecharts. Actions are specified in terms of time-windows that are to be executed in real-time. An abstract simulator protocol is devised for executing the ALRT-DEVS models under constrained computational resources. The approach is implemented in the parallel DEVS-Suite simulator. These models can be simulated on unitary computing platforms. A simple example switch model is detailed and tested to show the kinds of real-time modeling and simulation studies that can be supported.

Keywords

action-level modeling, real-time simulation, real-time statecharts, real-time DEVS

1. Introduction

A simulation platform refers to a simulator (a kind of software tool) executing on a given computing platform which typically consists of hardware, an operating system, and runtime libraries. If the operations of a model are defined and executed in terms of logical-time, then a computing platform can affect the physical time it takes to simulate the model (i.e., simulation execution can slow down or speed up) but not what the model does. For example a simulation execution lasting a very short physical time (e.g., a few seconds or a few hours) can reveal how throughput of a supply-chain process model can change over a period of one year. However, it is possible for the computing platform to adversely affect a simulation model's behavior. For some simulation studies including embedded systems (e.g., automobile cruise control and network-on-chips) and cyber-physical systems (e.g., self-driving cars and smart manufacturing), it is useful to develop simulation models that can execute as closely as possible in real-time and thus can interact with physical systems.[1] To achieve this goal, we may develop logical-time models and then execute them using a simulator that maps a model's logical-time to the simulator's real-time clock. This approach can lead to inaccurate or incorrect behaviors since time cannot be exactly represented and manipulated in computing platforms.[2]

It is useful to note that developing models that can be guaranteed to execute correctly and accurately in real-time remains among the challenging modeling and simulation research topics.[3] Hybrid (software and hardware) systems are more demanding to design if they are to be executed both in real-time and in the presence of finite resources.[4] Hardware systems naturally lend themselves to independent, concurrent model abstractions. In contrast, software systems are generally abstracted to sequential models. Design methods have been proposed to develop real-time models of software using bounded execution times. We can use the concept of a time-window to account for the variability of the processing cycles and memory of a computing platform on available real-time for executing a system's operations. A time-window defines an upper bound for an operation to be correctly executed. Worst case execution time analysis and static scheduling methods have been proposed for calculating time-windows.[5]

Returning to simulation modeling, execution of the operations in a model can be classified to occur in logical-time, soft real-time, or hard real-time. For some simulation

Arizona Center for Integrative Modeling and Simulation, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA

*SCS member.

Corresponding author:
Hessam Sarjoughian, Ira A Fulton School of Engineering, Brickyard Suite 501, 699 South Mill Avenue, Tempe, AZ 85281-8809, USA.
Email: hss@asu.edu

applications, such as autonomous, intelligent cruise controller[6] and flit-level Network-on-Chip design,[7] operations must be completed within hard real-time deadlines, otherwise the simulation may not be useful or may even be incorrect. For some other kinds of simulation applications, such as human-in-the-loop training exercises, executing operations in soft real-time (also known as best-effort) with some delay can be acceptable. Each of these modeling approaches must support specifications for time-based operations within individual models and across compositions of models. Furthermore, a model specified using one of these modeling approaches should be executed using a simulator having a logical-time, soft real-time, or hard real-time clock.

The classic and parallel atomic discrete-event system specification (DEVS) formalisms are defined in terms of state transitions with inputs and outputs. A model's external transition function maps a state and an input to another state at an instance of time. Similarly, internal transition function maps a state to another state at an instance of time. Operations can be associated with both of these functions, but they do not have specifications in these atomic models. Furthermore, state transition events are defined to occur at instances of time. However, in real-time simulation, state changes and operations may occur during a time interval instead of an instance of time. To allow an event to occur during some period of time, time-window was introduced to classic DEVS formalism.[8] Time-window is a multi-value function as compared with time advance function which is a single-value function. We note that the concept of time-window is different from probabilistic or other methods which compute an instance of time from a range of values. Therefore, neither classic nor parallel atomic DEVS models can represent operations and enforce their executions to be completed within assigned time-windows.

To represent operations that can be performed within designated time-windows, real-time DEVS (RT-DEVS) modeling approach with a simulator has been developed.[9] The concepts of activity set with constraints is introduced for representing operations. Also an activity mapping is introduced to map state to activities. The time-window concept is used to restrict the actual real-time that can be used to execute activities on a given simulation platform. Activities are not explicitly included in the external and internal transition functions. Individual activities with individual time-windows are not accounted for. Therefore, a set of activities cannot be directly assigned to external transition or internal transition functions. Without the ability to represent individual activities with their designated time constraints, they cannot be systematically prioritized. In addition, due to real-time constraints, it is necessary to recover from actions that cannot be completed within their assigned deadlines. We note that RT-DEVS is extended from classic DEVS which allows handling single events in atomic models and requires selecting one out of multiple atomic models that are ready to be executed.

Therefore, to allow representing actions with strict time constraints, we have developed the action-level real-time (ALRT) DEVS modeling approach where state-based actions are specified and sanctioned to be simulated within hard real-time time-windows. With this approach, timings are assigned to *actions* associated with state changes occurring within internal and external transition functions. Unlike logical-time or soft real-time DEVS models, the ALRT-DEVS atomic model is constrained to have time-windows greater than zero and at least as large as the minimum real-time clock resolution provided by the simulation platform. A real-time simulator capable of defining and executing the ALRT-DEVS model has been developed. A real-time model of a switch network is developed and simulated. The purpose of this example is to highlight what can be expected from real-time models that are targeted to be executed on real-time simulation platforms. The contributions of this paper are the ALRT-DEVS model with its abstract simulator protocol and an exemplar switch model.

The rest of this paper is organized as follows. In Section 2, the building blocks upon which this work is developed are presented. In Section 3, the ALRT-DEVS modeling approach is introduced with the focus on atomic model specification. Section 4 presents a simulation protocol for real-time simulation of ALRT-DEVS models. In Section 5, we describe a simple network switch with sample simulation results to show what can be expected using the ALRT-DEVS modeling and simulation approach. Works that are directly related to the proposed real-time modeling and real-time simulation are discussed in Section 6 and finally, Section 7 summarizes the paper and suggests some ideas on future research.

2. Background

The contribution of this research is inspired and built upon works in the modeling and simulation, and the software engineering communities. Specifically, DEVS,[10] a system-theoretic modeling approach, deals with *how to specify discrete-event models and how these models can be executed* and statecharts,[11] a component-based design approach, deals with *how to build software systems*. Few simulators have been developed using classical and parallel DEVS formalisms to support real-time simulations with varying degrees of timing accuracy.[9,12–14] Many DEVS simulators supporting logical-time modeling and logical-time and soft real-time simulations have been developed dating back to around the 1990s. Among these we use DEVS-Suite simulator[15] for convenience and use it to develop the ALRT-DEVS simulator. A variety of methods and tools have developed for designing software systems

that operate in real-time.[16–18] In this section, works that serve as the basis for ALRT-DEVS modeling and simulation are described.

2.1 Discrete-event system specification

In this paper, formulation of real-time modeling and real-time execution is based on the parallel DEVS framework.[10,19] In our work, real-time simulation modeling is aimed at message-based input and output events and independently handling state transitions subject to hard real-time constraint specification and execution. Events may arrive at arbitrary time instances and actions associated with state transitions may consume arbitrary non-discrete time durations. We note that although the computation cost of a input/output messaging scheme is high, it does not outweigh model modularity.

2.1.1 Parallel DEVS models. The parallel DEVS formalism is based on a real-valued time-based abstraction. It can handle multiple input events, multiple output events, and allows simultaneous occurrences of input and output events subject to not violating its legitimacy principle. Simultaneity in external and internal state transitions is generally handled using a state transition that has a time duration of zero. For a coupled model, any of its components can receive multiple input events at the same time or send multiple output events at the same time. Sending and receiving events are independent and are causal. That is, although inputs and outputs can occur at the same time, output events must be first generated and then received as input events. Neither atomic nor coupled models can send output events to itself. Every hierarchical coupled model must have a tree-structure and no coupled model can contain itself. The time in each atomic model is an abstract, real-valued entity. A coupled model does not have its own time. This is because given a coupled model, it takes zero time for an output event of a model to arrive as an input event to another model. Couplings in all coupled models are timeless and define output to input causality. When delivery of a model's output as input to another model takes time, then an atomic model representing a delay can be added to the coupled model.

These atomic and coupled models have a corresponding abstract atomic simulator and coupled coordinator protocols. These protocols must be implemented, which means they can have different software designs. Each atomic and coupled model is assigned its own simulator and coordinator. A root coordinator, assigned to the highest level coupled model, is responsible for executing together all atomic and coupled models that are contained in the highest level coupled model. One such design and implementation is developed for the DEVS-Suite simulator.[15,20] This simulator intrinsically supports simulation in abstract time since state transition with a period of zero time is allowed. Every simulator has an independent abstract clock. An abstract global clock is used for coordinators. These clocks are real-valued. This global clock is used to coordinate message exchanges among atomic models of a coupled model. It is implemented using a the clock provided by the host computing platform and in particular, Java Virtual Machine (JVM) clock. Thus, the abstract global clock can run faster than, equal to, or slower than the JVM clock. Under the assumption that the pace of the abstract clock is the same as the JVM clock and that there is an absence of zero-time state transitions, there still can be no guarantee for the simulation steps to be completed in concert with a physical clock. In such scenarios, the simulator can at best execute as-fast-as-possible in relation to the physical clock with best-effort synchronization between the JVM and physical clocks. Timing for external and internal transition functions are *holistic*, i.e., time is allocated for all operations that belong to external (or internal) functions. The DEVS atomic simulator protocol is not defined to handle time periods for individual operations, priority among actions, or recovery from terminated actions. We note that time advance function can be defined as a probability function, but it still must have a single value.

2.1.2 RT-DEVS. An extension of the classic DEVS formalism[10] called RT-DEVS has been developed.[9,21] A real-time atomic model is specified as $RTAM = \langle X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ti, \psi, A \rangle$. A set representing actions A and a mapping function ψ are introduced to the formalism. Every action is defined to be atomic. The time interval function ($ti(s)$) is defined as a time-window, $ti(s)$: $s \rightarrow \Re^{+}_{0,\infty} \times \Re^{+}_{0,\infty}$. Actions are defined to be completed within a duration that is bounded by a pair of minimum and maximum time instances (i.e., a time-window). Therefore, the external transition function is defined as δ_{ext}: $Q \times X \rightarrow S$, where $Q = \{(s, e) | s \in S, 0 \leqslant e \leqslant ti(s)|_{max}\}$. A state can be associated with a set of actions, ψ: $S \rightarrow A$. Actions associated with states can have their own timings. An external event for a given state can be associated with action and must be completed within its designated time-window. Input (X), state (S), and output (Y) sets as well as the internal transition (δ_{int}) and the output (λ) functions are the same as those defined for classic DEVS. Receiving and sending events are considered to be instantaneous. Real-time atomic models are coupled in the same way as classic coupled models without the ''select'' function. The tie-breaking is delegated to the simulator protocol. The selection is said to reflect randomness inherent in real-time systems.

A real-time simulator protocol has been proposed for RT-DEVS.[9,21] A real-time clock (provided by the operating system (OS)) employed to execute actions is defined in atomic models. The simulator allows executing one action

due to an external event. This action is completed within its allotted time-window unless it is interrupted by another external event, in which case the current action is aborted and another action associated with the new event is started. If an external event is received outside the time-window, then an error is generated. A real-time coordinator supporting two threads is developed. A real-time operating system is assumed for concurrently monitoring and executing the atomic models. One thread monitors for external events and another for executing activities. When multiple atomic models need to be executed, one of them must be chosen. This allows suspending or terminating active external and internal events. A simulator confined to non-zero, finite discrete time-steps ($\mathbb{N}$) is developed using interrupt handlers and priority scheduling provided by a real-time operating system.

2.2 Real-time statecharts

Statecharts offer concepts and constructs to model behaviors of components in terms of states, events, and actions.[11] Transitions can be defined to change the state of one component due to both events that are invoked by itself or other components. The after (Δt) and when ($t = t_\alpha$) constructs allow representing time. However these constructs both syntactically and semantically are not well-suited for handling hard real-time modeling. Instead, a real-time statecharts approach[22] is developed using statecharts and Timed Automata.[23] Hard real-time is introduced into statecharts using clocks from Timed Automata. Clocks allow specifying time intervals in the forms of time invariants associated with states, timed guards for transitions, and worst case execution times (WCET) for actions. Each clock can be reset when a transition fires. These allow placing timing constraints (deadlines) explicitly and independently on both states and state transitions. A state can be active (i.e., the *entry, do*, and *exit* operations of the states in statecharts can be restricted to execute within time-windows) for a specified time period as long as its time invariant is true. Time-windows can be specified for triggering transitions, executing their actions, and setting deadlines (relative to the triggering of a transition or a clock). States that are time dependent are referred to as *locations* in order to separate them from those which are untimed. Several locations can be mapped to one state in order to specify the priority and type of actions to be executed. Figure 1 shows visual notation for a simple real-time statechart (examples of complex, hierarchical real-time statecharts can be seen in the literature[22]).

For real-time statecharts, the time invariants are restricted to $\wedge_{t_i \in C}(t_i \leqslant \mathbb{N} \cup \{\infty\})$. Time guards are restricted to $\wedge_{t_i \in C}(a_i \leqslant t_i \leqslant b_i)$, where $a_i \in \mathbb{N}, b_i \in \mathbb{N} \cup \{\infty\}$ and C is a set of clocks. Real-time statecharts support hierarchical statecharts as defined for extended hierarchical timed automata.[22] In case more than one transition is triggered at the same time, one is (randomly) fired and the others are delayed. Some temporal inconsistencies (e.g., time invariant in a state is contradicted with a transition's time guard) can be avoided using static analysis. Other inconsistencies may be detected by restricting real-time statecharts. Assuming periodic execution, WCET for *entry, do*, and *exit* operations can be computed to avoid temporal inconsistencies.[16]

The above time invariant and time guard specifications are well-suited for defining real-time DEVS external and internal transition functions that are specified to have actions. External and internal transition functions can be specified in terms of actions, each with its own time-window assignment. Thus, an atomic model can have action-level specifications within and across internal and external transition functions as detailed in the following section.

3. ALRT-DEVS modeling

The examination of the parallel DEVS, RT-DEVS, and real-time statecharts shows that they are not well-suited for real-time simulation where individual actions in a model can be assigned deadlines and their executions are subject to finite computational resources. To achieve this goal, the action-level, real-time DEVS (ALRT-DEVS) simulation modeling approach with hard real-time simulation capability is developed as described below. The real-time statecharts modeling syntax and semantics are used for introducing action-level, real-time specification into RT-DEVS. Next, appropriate concepts and modeling constructs are developed.

3.1 Revisiting time

The notion of time has been described and formalized from different points of view of computing.[2,24,25] In this work, time is accounted for in terms of simulation modeling and in particular, from a unitary computing platform with a single processor supporting multiple threads. Time is considered as a totally ordered set of discrete or real numbers for representing passage of time in models.[10] Time used in execution engines (such as simulators) for these models is also considered as a totally ordered set of discrete or real numbers.

A physical clock measures time as a scalar value. The rate of change for physical-time is *linear* and *monotonic*. The rate of change for physical-time is constant and equal to one. Time in a simulator clock is defined relative to a physical clock. The physical and simulator clocks can be related to one another using a numerical *scale factor* $sf \in \mathbb{N} > 0$. Time as a scalar quantity in a simulator can be elongated or shortened with respect to time in another clock which may be a physical clock. This is useful for allowing the physical-time defined in a model to execute

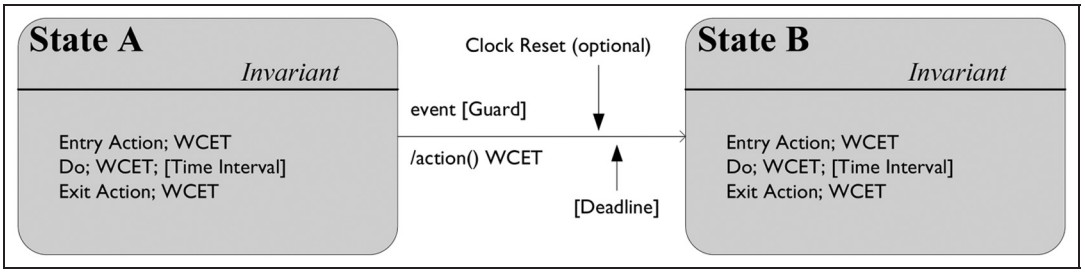

Figure 1. Real-time statecharts.

faster or slower relative to a simulator's clock (e.g., JVM clock or a processor's central processing unit [CPU] clock). A simulation is said to be executing in ideal ''real-time'' if and only if its clock's rate of change is identical to a physical clock. In this paper real-time refers to the simulator's passage of time to be as accurate and precise as the time in a physical system not withstanding that a physical clock and a real-time clock cannot be identical to each other.[2]

Time is also classified as physical-time, simulation time, or wallclock time.[26] Physical-time refers to time in physical systems, simulation time refers to an abstract notion of time for simulation execution, and wallclock time refers to the physical-time while the simulation executes. A simulation can execute in as-fast-as-possible (i.e., logical-time), scaled real-time (faster or slower than wallclock time by a scale factor), or real-time in relation to wallclock time. Time in a simulation is defined as a totally ordered set of values where each value represents an instance of time in some model of a physical or fictitious system. In the context of virtual simulations and training exercises, as much as possible time monotonically increases at the same rate as the wallclock time. The wallclock time is converted using a scalar quantity to define simulation time. To support real-time simulation execution, the rate at which simulated time is increased must be slowed down assuming the simulation executes faster than the wallclock time.[27]

Both theoretical and pragmatic perspectives on the role of time in models, simulators, and computing platforms are important for specifying real-time models and executing them in real-time. In the former, the physical clock is the basis for concretizing time in models and the simulation protocol. In the latter, wallclock time is the basis for executing simulations. In both perspectives, the notion of the global simulator clock (logical-time and real-time) is used for synchronizing time across multiple, integrated simulation models. In the rest of this paper, the real-time clock used in a simulator is assumed to be provided by the clock of some target computing platform with finite accuracy and precision. The real-time clock is assumed to represent the physical-time as accurately and precisely as possible. Physical-time refers to the time it takes for some operation to be completed in some physical system. The basic notions are that real-time is used for measuring simulation executions and physical-time represents passage of time in some operating actual (physical) systems.

1. *Physical-time* refers to the time in the physical world. Ideally, it can be measured with infinite accuracy and precision.
2. *Real-time* is an approximation of physical-time. As an abstract quantity (i.e., virtual time) its rate of change is assumed to be one. Theoretically, real-time defined in a model or for a simulator cannot be equal to physical-time; uncontrollable factors in computing platforms affect its accuracy and precision.
3. *Logical-time* is an abstract computable quantity, ideally having the properties of physical-time.

In order to define timing with respect to real-time models and real-time simulators, we provide the following definitions. A simulation model refers to the implementation of a model in a computer programming language that can be executed using a simulator. The simulator is kind of a real-time software system. The time in a simulation platform should be a scaled value of the time in a physical system. However, during a simulation execution period, the time instances of a simulation model in either logical-time or real-time cannot necessarily be a scaled factor of the time instances of a physical system as described above. A segment of a real-time simulation execution may run faster than (slower than or equal to) its physical system counterpart. We include the physical-time as a theoretic value and use it to show that real-time in a simulation platform can approach it, but not equal it. This is useful when real-time simulation is part of a physical system and the executions of the simulation and non-simulation parts need to be synchronized.

1. *Physical system* refers to an actual system that operates in physical-time.
2. *Real-time modeling and simulation platform* refers to a computational environment in which real-time models can be implemented and also executed in real-time.

3. *Real-time simulator protocol* refers to an algorithm that can execute an abstract real-time model in real-time.
4. *Abstract real-time model* refers to a specification for a physical system where time in the model is a totally ordered set of real values.

Two consecutive time instances $t_{m,a} < t_{m,b}$ for a model m and $t_{s,a} < t_{s,b}$ for a simulator s can be linearly scaled relative to each other and $t_{p,a} < t_{p,b}$ for a physical system p. Every $< t_{m,a},\ t_{m,b} >$ (or $< t_{s,a},\ t_{s,b} >$) represents either *logical-time* or *real-time* instances, but not both. Time durations $\delta t_m = < t_{m,a},\ t_{m,b} >$, $\delta t_s = < t_{s,a},\ t_{s,b} >$, and $\delta t_p = < t'_{p,a}, t'_{p,b} >$ must be *consistent* with each other. Consistency means given $\delta t = t_b - t_a$, $\delta t' = t'_b - t'_a$, $t_a = t'_a$, and a scale factor sf, then $\delta t = sf * \delta t'$. For $sf = 1$, real-time used in a model with its real-time simulator is theoretically equal to the physical-time of its actual system counterpart. Real-time is assumed to increase at the same rate as the physical-time. Differences in real-time and physical-time instances are assumed to be negligible, i.e., computability indistinguishable. We note that the passage of time in any computational modeling and simulation platform can be at most synchronized at a finite number of time intervals with respect to the physical-time. The passage of logical-time in a simulated model can be faster (or slower) compared with its physical-time. For a given modeling and simulation platform, the scale factor between δt_m and δt_s can be $\mathbb{N} > 0$ or a real number with finite-precision approximation. Assuming that time for a model m increases as in physical-time, then $\delta t_s \triangleq sf * \delta t_m$ (or $\delta t_m \triangleq sf * \delta t_s$). Given $\Delta t_m = \Sigma_{i=1}^{h} \delta t_{m,i}$ and $\Delta t_s = \Sigma_{j=1}^{k} \delta t_{s,i}$, then $\Delta t_s = sf * \Delta t_m$ (or $\Delta t_m = sf * \Delta t_s$). It is important to note that theoretical accuracy for time in a model cannot be realized in simulators which invariably represent model time with finite accuracy. This mismatch with some techniques including use of $\mathbb{Q}$ instead of $\mathbb{R}$ are described for DEVS models.[28]

3.2 Atomic model

The above concepts and characterizations of time are used in developing the action-level real-time atomic model. The specifications for the building blocks of the ALRT-DEVS atomic model are exemplified using a simple example shown at the end of this subsection

$$ALRT{-}DEVS = \langle X, Y, S, A, \Gamma, \Omega, \psi, \lambda, ti \rangle$$

It is useful to note that the model specification is at the input/output system (IOS) abstraction hierarchy.[10] The input (X), output (Y) and state (S) sets are the same as those defined for a parallel atomic model.[10,19] The external (δ_{ext}) and internal (δ_{int}) transition functions in parallel DEVS are defined as Γ and Ω sets of functions, respectively. The activity mapping ψ and time-window ti functions defined in the real-time DEVS model specification are reformulated as part of the Γ and Ω. The output function (λ) is also reformulated. The action set (A) is defined in the atomic real-time DEVS specification. In the following sections, we will detail the specifications of the ALRT-IOS atomic model.

3.2.1 Primary and secondary state variables. Next states with hold-in times for parallel DEVS models are generally defined using the *phase* and *sigma* (σ) state variables. These *primary* state variables are necessary for defining how a model responds to external and internal events. Additional state variables are often needed as the dynamics of a model grows in complexity and scale. Thus, the state of an atomic model is specified using a tuple ($S = phase \times \sigma \times v_1 \times \cdots \times v_n$). Variables (i.e., $v_1 \times \cdots \times v_n$) are *secondary* states. These state variables play a fundamental role in specifying real-time DEVS models. State-to-action mappings (ψ: $S \rightarrow A$) can be specified systematically. They are used in formulating state-to-action mappings such that multiple actions can be defined according to specific ordering of actions while satisfying real-time simulation execution constraints. The secondary state concept also aids not only in state-to-action mappings within each of the internal and external transition functions but also for moving between these state transition functions.

3.2.2 State-to-action mapping with time constraints. A finite number of actions can be specified for a location $< phase,\ \Delta t >$. The time-window for any action is defined as $\delta t_{action} : A \rightarrow TW_l \times TW_u$, where $TW_l = \Re^{+}_{[0,\infty)}$, $TW_u = \Re^{+}_{(0,\infty]}$, $0 < tw_u - tw_l \leqslant \infty$, $tw_l \in TW_l$ and $tw_u \in TW_u$. The lower and upper time bounds for an action are tw_l and tw_u, respectively. A finite number of actions can be assigned to a location (see Figure 2). The maximum total amount time for $action_1, \ldots, action_r$ is defined as $\Delta t = \Sigma_{j=1}^{q} \delta t_j, q \leqslant r$. Each δt_i can be its WCET (i.e., tw_u). These actions are sequentially executed to completion subject to the amount of real-time that is available to the simulator. The total real-time expended on executing these actions by the simulator is expected to be consistent with the physical-time an actual system takes to perform the same actions (see Section 3.1). This requires having time specifications for every action. In the event that the available real-time in the simulation platform is less than Δt, one or more actions cannot be executed.

As described in Section 2.2 real-time statecharts provide time constructs (i.e., time invariants, timed-guards, and deadlines) for specifying multiple actions with designated time-windows for external and internal transition functions. Actions are allocated to primary states with

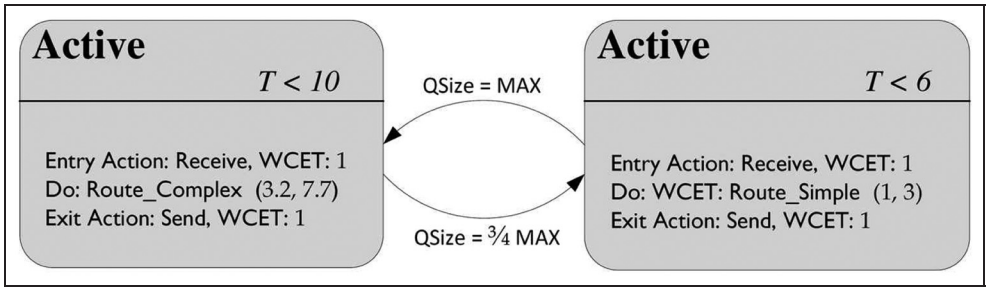

Figure 2. A switch model with two different routing methods.

assigned time-windows. Each transition may invoke one or more actions belonging to a location. A transition from one location to another is triggered by evaluating secondary state variables. One or more secondary state variables act as guard conditions for the actions assigned to a location. An action is completed only if its time-window constraint (WCET) is satisfied, i.e., its execution can be completed in its designated real-time as sanctioned by the simulation platform. The clocks in real-time statecharts can be used to define elapsed times. Therefore, execution of some actions in ALRT-DEVS internal or external transition functions are terminated when sufficient time is not available (i.e., WCET is not satisfied). Simulation execution can benefit from efficient data structures for state transitions and actions.

As an example shown in Figure 2, a list of actions for a switch are defined.[7] The time intervals are associated with actions (e.g., the *Route_Complex* action is specified to have minimum and maximum times 3.2 and 7.7). Distinct sets of actions having their own total durations ($T < 6$ and $T < 10$) can be defined for the *Active* phases using secondary state queue size (*QSize*) as a(n) (un-timed) guard condition. That is, $Qsize = MAX$ and $QSize = \frac{3}{4}MAX$ guards serve to change what sequence of actions is to be executed. Thus, a transition becomes necessary when some actions are to be reordered or dropped. Receiving, processing, and generating events are treated as individual actions. This is necessary since receiving and sending events should not be considered to occur instantaneously. Note that there are two *Active* phases in the external transition function, each having its own time invariant. Transition between $\langle Active, 10\rangle$ and $\langle Active, 6\rangle$ is instantaneous. Finally, note that in ALRT-DEVS actions in external and internal transition functions are associated with states. Although timed-guards and deadlines are not assigned to transitions as in statecharts, this poses no restrictions since an atomic model cannot directly invoke actions of another model. In ALRT-DEVS, as in DEVS, actions in a model may be executed by receiving messages from another model.

3.2.3 External transition function. The external transition function handles the events received via input ports as in parallel DEVS. Processing an event may cause the system to change state and/or invoke a set of actions. In parallel DEVS, an external transition function changes state and executes a set of actions holistically (i.e., a single sigma is assigned regardless of the number of actions). In contrast, in ALRT-DEVS, Γ, ψ, and *ta* are the elements that support defining multiple actions due to external events. A state change can cause a sequence of actions, each of which has its own state and time invariant called location ℓ. The external transition function Γ can be divided such that there are as many locations as there are actions. Location is specified as $q \subseteq s$ since s is a total state. The external transition function must allow receiving input events while any of the actions are executing. The elapse time e acts as a clock in real-time statecharts with the requirement that it is reset upon invocation of any action.

Activity Mapping Function:

$$\psi: L \rightarrow A, L = \{\ell_g\}, \ell_g = (q_g, \delta t_g), \ell_g \rightarrow a \in A,$$
$$q_g \subseteq s_g, s_g \in S, \Sigma_{g=1}^{n} \delta t_g \leqslant \Delta t,$$
$$\delta t_g \in ti(s)$$

Example: $\psi(\langle \text{“Active”}, 6\rangle) = \{Route\_Simple\}$

The function maps the location $\langle$“*Active*”, 6$\rangle$ to action *Route_Simple*.

Time Window Function:

$$ti(s): S \rightarrow tw_l \times tw_u, tw_l = \Re^{+}_{(0,\infty)},$$
$$tw_u = \Re^{+}_{(0,\infty]}, tw_l < tw_u$$

Example: ti (“*Active*”, σ, *Qsize*) = [3.2, 10)

The shortest time required to finish the routing process action is 3.2 (simple routing) and the maximum time is 7.7 (complex routing).

External Transition Function:

$$\Gamma = \{\delta_{ext,g}\}, g \in \mathbb{N}, \delta_{ext,g} : (L, e, x^b) \rightarrow L', 0 < e \leqslant \infty, x^b \in X$$

Example: $\delta_{ext,1}(\langle \text{“idle”}, \infty\rangle, e, flit\_receipt) = \langle \text{“Active”}, 10\rangle$

This example shows the external transition for when a flit is received in the switch model. There is a transition from phase idle to phase routing if the queue is empty.

As in parallel DEVS, ALRT-DEVS supports receiving a bag of input events. The maximum time allocated to execute all actions is Δt. Note that elapse time e is measured in real-time and it must be greater than zero. This prevents simulation steps with zero time duration. The lower bound, tw_l, for time-window of every action must be greater than zero and less than infinity. The upper bound, tw_u, for time-window of every action can be infinity (see Section 3.2.2). The lower bound value must be strictly less than the upper bound value. Unlike soft real-time DEVS where only an external event can interrupt an action, in ALRT-DEVS an action can be terminated once its assigned time-window expires. Time expiration is handled by the real-time atomic simulator (see Section 4). The real-time atomic model does not support the confluent transition function (δ_{conf}) defined for parallel DEVS since receiving and sending events cannot be handled simultaneously (see Section 3.2.2).

3.2.4 Internal transition function. In ALRT-DEVS internal transition function, state can change solely based on the current state. However, unlike parallel DEVS, there can be multiple actions, each with its own time-window. As in the external transition function, states with time invariants (i.e., locations L) are used. These actions, each consuming a positive time duration, can be sequentially executed. The internal transition function Ω can complete all of its individual actions assuming there exists sufficient physical-time available at the time they are to be executed. Therefore, unlike parallel DEVS, there can be no guarantee that all actions in the internal transition function can be completed. Furthermore there can be as many output events as there are actions. These actions are generated at different time instances. This is in contrast to parallel DEVS where all output events are defined with respect to the state of the model (i.e., due to cumulative state changes and actions). The lower and upper bound time instances for every action is handled as in the external transition function.

Internal Transition Function:

$$\Omega = \{\delta_{int,h}\}, h \in \mathbb{N}, \delta_{int,h} : L \rightarrow L', L = \{\ell_h\},$$
$$\ell_h = (q_h, \delta t_h), \ell_h \rightarrow a \in A, s_h \in S,$$
$$q_h \subseteq s_h, \Sigma_{h=1}^{n} \delta t_h \leqslant \Delta t, \delta t_h \in ti(s)$$

Example: $\delta_{int,1}(\langle \text{“Active”}, 10\rangle) = \langle \text{“Active”}, 6\rangle$ if $Qsize = \frac{3}{4}MAX$

In the case where three-quarters of the queue is full, the switch transitions from the complex routing location (which requires more time) to the simple routing location.

3.2.5 Output function. The remaining element of the real-time atomic model is the output function. Outputs are defined in terms of actions. In parallel DEVS, outputs are associated with change of states which implicitly account for any operations that may be called in either external or internal transition functions. Thus, in ALRT-DEVS, output function is a mapping from *actions* to outputs. However, since there is no guarantee for all internal and external transition functions in Γ and Ω to be completed, some outputs may be missed. If a state change remains within its time-window (i.e., a state change is not interrupted due to lack of sufficient physical-time, but only because either $\delta_{ext,g}$ or $\delta_{int,h}$ is completed successfully), then ALRT-DEVS output function degenerates to that of the parallel DEVS output function. Multiple output events may be associated with an action. Output function, unlike external and internal transition functions, is defined to consume no physical-time. This does not strictly limit allocating time for generating output events; instead it requires accounting for time in external and internal transition functions (i.e., increasing tw_u of the location that is responsible for output generation). It is useful to note that in real-time statecharts, outputs are conceptualized as generating actions which can be constrained to complete within designated time periods.

Output Function:

$\lambda(\ell) : A \rightarrow Y^b$

Example:

$$\lambda(Route\_Simple) = (index_{output\,port})$$
where $index_{output\,port} \in \mathbb{N}$

In the simple switch example, the output for routing action is a number which specifies the output port from which the flit is to be sent out.

The ALRT-DEVS modeling approach allows dividing generic external and internal transition functions defined in parallel DEVS into individual actions and producing output for each of them individually. This is depicted in Figure 3. A delay model is defined to handle external events without internal transition function. The operations of the model are: receive an external event; stores the data (received event); fetch the data; and send an output. The external event invokes two actions: $Action_1$ and $Action_2$. For these two actions along with the idle phase, this model requires three locations and at an instance of time it can be in one of them. The right-hand side of Figure 3 shows a sample execution of the model with external event In_1 and the two actions. The external event is encountered at time instance 5 and then the model enters the location L_1. In this location, $Action_1$ is executed and Out_1 is produced at time instance 10. Then the model enters L_2, $Action_2$ is executed and output Out_2 is produced at time instance 17. Both of these actions were executed in their respective time intervals ($Action_1$: [4−6] and $Action_2$: [5−8]).

Here, we provide the ALRT-DEVS specification for the delay model illustrated in Figure 3. As described earlier and shown in equation (1), we consider *phase* and *sigma*

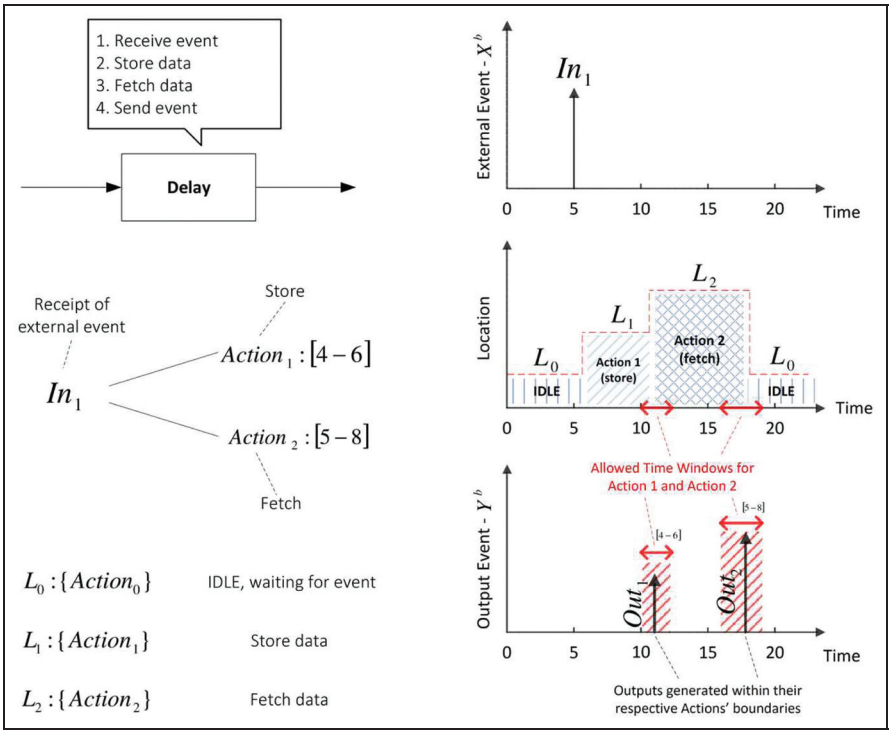

Figure 3. A delay model with multiple actions associated with one external input event.

as primary state variables which together make a *location*. Locations of the delay model are specified in equations (5) to (8). The remaining equations (states, input/output/action sets, external/internal transition functions, output/activity mapping functions, and time intervals) directly correspond to Figure 3.

$$S = \overbrace{\overbrace{\{Active, Idle\}}^{\text{phase}} \times \overbrace{\sigma}^{\text{sigma}}}^{\text{location}} \times \overbrace{\{0,1\}^*}^{\text{data}} \quad (1)$$

$$X = \{(inport, \{0,1\}^*)\} \quad (2)$$

$$Y = \{(outport, \{0,1\}^*)\} \quad (3)$$

$$A = \{Action_1, Action_2\} \quad (4)$$

$$L = \{L_0, L_1, L_2\} \quad (5)$$

$$L_0 = \{(\text{“}Idle\text{”}, \infty), \{\}\} \quad (6)$$

$$L_1 = \{(\text{“}Active\text{”}, 6), \{Action_1\}\} \quad (7)$$

$$L_2 = \{(\text{“}Active\text{”}, 8), \{Action_2\}\} \quad (8)$$

$$\Gamma = \{\delta_{ext,1}, \delta_{ext,2}\} \quad (9)$$

$$\delta_{ext,1}(L_0, data, e, (inport, X)) = (L_1, data = X) \quad (10)$$

$$\delta_{ext,2}((\text{“}Active\text{”}, \sigma), data, e, (inport, X)) = ((\text{“}Active\text{”}, \delta - e), data) \quad (11)$$

$$\Omega = \{\delta_{int,1}\} \quad (12)$$

$$\delta_{int,1}(L_n, data) = \begin{cases} (L_2, data) & [n\ equals\ 1] \\ (L_0, data = null) & [n\ equals\ 2] \end{cases} \quad (13)$$

$$\lambda(L_1, data) = (outport, \text{“}CONFIRM\text{”}) \quad (14)$$

$$\lambda(L_2, data) = (outport, data) \quad (15)$$

$$\psi(L_1, data, index) = \{Action_1\} \quad (16)$$

$$\psi(L_2, data, index) = \{Action_2\} \quad (17)$$

$$ti(L_1, data) = [4, 6] \quad (18)$$

$$ti(L_2, data) = [5, 8] \quad (19)$$

What is shown in this figure and in the specification (equations (1) to (19)) cannot be replicated in P-DEVS or in RT-DEVS. In these modeling approaches, an external event handling and state transition is started with one or more event and then one (RT-DEVS) or more outputs (P-DEVS) produced. By enabling the modeler to specify actions inside external and internal transition functions, time-critical, reactive systems can be more accurately modeled and simulated by imposing strict time constraints to individual actions.

3.3 Coupled model

ALRT-DEVS atomic models can be composed hierarchically to create larger models. Such composed models are at the coupled model abstraction hierarchy.[10] Similar to other timed and un-timed modeling formalisms, a coupled

model does not ascribe time to input/output communication. Instead, there is ordering from outputs and inputs. A component's output must occur prior to being used as an input by another component. In parallel DEVS, there exist three types of communications using input and output ports via couplings. External input coupling allows a coupled model's input to be transmitted as input to its components. External output coupling allows the components of a coupled model to transmit their outputs to the coupled model. Internal coupling allows any component of a coupled model to send its outputs to any other component of a coupled model as inputs. Couplings are instantaneous; they only specify causality (i.e., outputs that are generated in one simulation cycle become inputs at the beginning of the next simulation cycle). Couplings that must have time can be specified as atomic models. The passage of time in a coupled model can only be due to its atomic components. It is straightforward to add a delay from the time output departs a sender to the time it arrives at a receiver. The coupling is replaced with an atomic component with appropriate couplings added.

4. Abstract ALRT-DEVS simulator

Models specified in ALRT-DEVS can be simulated in real-time provided time-windows are strictly enforced using real-time clocks. This requires developing abstract atomic simulator and coupled model coordinator protocols that can ensure correct executions of atomic and coupled models. The parallel DEVS protocols supporting logical-time (and soft real-time) are adapted to execute actions defined in the internal and external transition functions Γ and Ω in real-time. Implementations of these abstract atomic simulators and coupled coordinators are available in the literature.[15]

4.1 Atomic simulator protocol

A real-time atomic simulator protocol is devised such that an action is allowed to execute to completion as long as there is no time violation (i.e., an action can be completed within the time-window that is available in the simulation platform). The protocol manages execution of a sequence of actions.[7] The simulation platform does not guarantee to provide the same real-time duration for an action when it is executed multiple times throughout one or more simulations. Furthermore, the simulator does not ensure the execution of action's time-window takes the exact amount of physical-time as its (actual or hypothetical) system does. As in most atomic or parallel DEVS, every ALRT-DEVS atomic model is assigned its own atomic simulator. The abstract simulator protocol handles all the actions defined in both the external and internal transition functions. Actions to be executed, their ordering, and their time-windows are defined in Γ and Ω. The simulation protocol has T_{min} and T_{max} variables which get their values from real-time clock provided by, for example, Java Virtual Machine. These variables represent the simulator's real-time clock. After every simulation cycle, the values of these variables are updated. An action belongs to a set of actions that are defined either in the external or internal transition function. Every action set can be interrupted when one or more external events are received at an instance of time. Therefore, an action can be preempted or terminated, but not due to real-time constraints. The execution of the actions belonging to the Γ and Ω functions cannot be interleaved.

In order to explain in detail how the atomic simulator protocol is devised, we provide Algorithm 1 and the flowchart diagram shown in Figure 4. The algorithm's *Delta Function* procedure has two arguments called *Message Bag* and absolute time *t*. A simulation cycle is a single path starting and ending with the conditional statement of ''Message bag is empty?'' corresponding to the first node in Figure 4 and the first statement in Algorithm 1 at line 2. Next, we describe the details of the algorithm.

In Algorithm 1, when the conditional statement at line 2 is TRUE then an action-related event has occurred. The conditional statement at line 3 checks for time violation of the current action (*Action*). If time violation has not occurred, the output of the *Action* is generated (at line 4). In the event of a maximum time violation, at line 6, the action is terminated and no output is generated. After this, the protocol checks whether the previous action (*Action*) was the final action in the current *Action Set* (checked at line 7). If yes, an internal transition is necessary; therefore, the internal transition function (δ_{int}) is called at line 8 and the *Action Set* is updated based on a new location. Finally, common to all internal events (whether a time violation, internal transition, or action substitution) is setting the current *Action* (the first action in the *Action Set*), the T_{max} and T_{min} values for this new *Action* (line 10).

If the conditional statement at line 2 returns FALSE (the *Message Bag* is not empty), an external event has occurred. External events are handled by the external transition function (δ_{ext}) at line 12. The execution of δ_{ext} may have two outcomes: 1) change of the location and a new *Action Set* or 2) continuing with the same location, *Action*, and *Action Set*. This condition is checked at line 13. The first case is handled at line 14 which is setting the new *Action* and updating the T_{max} and T_{min} values. Lines 16-18 correspond to the second case. In this statement, the elapsed time of the action is updated and its execution is continued from where it was left off.

Figure 4 shows the algorithm. A description of the protocol with respect to the elements of the ALRT-DEVS model is provided below. Also, in the end, we discuss the time-inconsistencies caused by real-time execution of the simulator and how it is handled in the atomic simulator protocol.

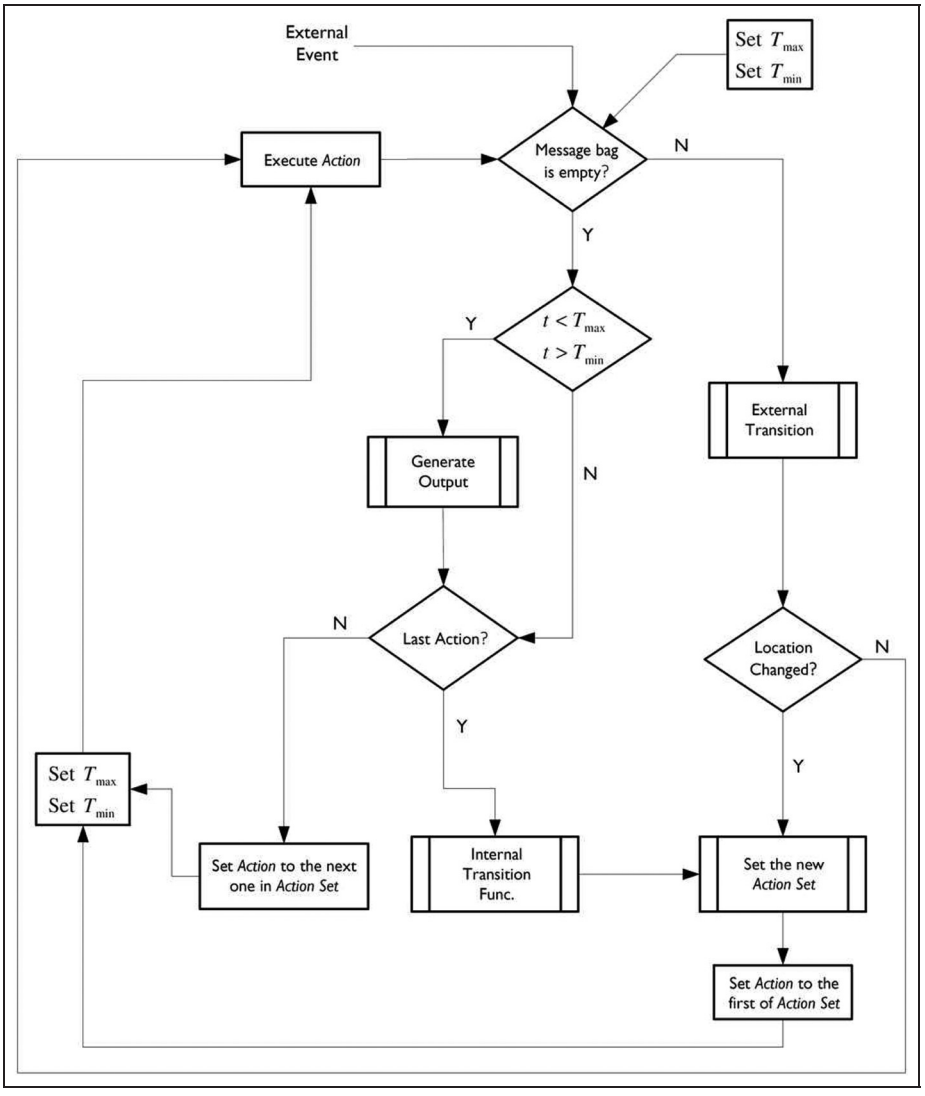

Figure 4. Abstract ALRT-DEVS atomic simulator protocol.

Initialization

- At initialization, the simulator's T_{min} and T_{max} are set. These default values are set to zero which means there are no actions to be executed. If there is at least one action, its lower and upper time-window are used to set T_{min} and T_{max}.

External transition function

- All actions are completed without any interruption as long as their time-window constraints are satisfied. The actions are executed according to the order specified in Γ. Using the ''Last Action?'' conditional, Ω is invoked.
- Some actions are not completed. An action will be executed to completion unless it cannot be completed within its allotted time. An action may miss its upper limit deadline (i.e., $\delta t > T_{max}$) which results in its termination and scheduling of the next action if there is one. For every successful execution of an action, an output can be generated and T_{min} and T_{max} are updated. Execution of an action can also be terminated due to arrival of external events; an active action can be preempted with another action.

External transition function to internal transition function

- After the last action in Γ is completed, the simulator starts to execute actions that belong to Ω.

Internal transition function

- All actions are completed. This requires every action to be executed to completion.
- Some actions are not executed due to missing their deadlines or are terminated due to receiving external events.
- When an internal transition action is scheduled to end at the same time that an external transition action is to begin, then priority is given to the

Algorithm 1. ALRT-DEVS atomic simulator protocol.

```
1:  procedure DeltaFunction (Message Bag,t)
2:    if Message Bag is empty then
3:      if t < T_max ∧ t > T_min then
4:        Call λ_Action
5:      else
6:        Terminate Action
7:      if ActionSet is Empty then
8:        Call δ_int
9:        Set ActionSet
10:     Set Action, T_max, and T_min
11:   else
12:     Call δ_ext
13:     if Location is Changed then
14:       Set ActionSet, Action, T_max, and T_min
15:     else
16:       Elapsed←delta – tL
17:       tL←t
18:       return
19:   tL←t
20:   tN←tL + Dist(T_max, T_min)
```

former action. Generating output events has priority (i.e., external events are lost). This scheme follows the default scheme provided for the confluent transition function defined for parallel DEVS. It should be noted that in ALRT-DEVS, outputs are generated per actions, not in the holistic manner allowed for the external and internal transition functions of parallel atomic DEVS models. It is possible to give priority to accepting external events that are received at the same time an internal transition action is scheduled to end.

Remaining functions

- The time interval, state-to-action mapping, and output functions are timeless. The abstract simulator, therefore, does not allocate consuming time for their executions. Obviously, the simulation platform consumes physical-time even though the passage of time is not represented in the simulation.

The real-time required for executing a complete simulation cycle is for an action either in the external or the internal transition function. The time duration specified for every action must be greater than zero. Thus, the physical-time it takes for executing actions is at least as long as the real-time that the simulator is keeping track of. The real-time atomic model does not include timing for extraneous operations such as reading data from or writing data to an external file or console. Thus, from the physical-time perspective, some amount of time is needed for operations due to all operations in the atomic model except the actions in Γ and Ω and thus, there is discrepancy between the amount of time it takes in the real-time simulation platform to execute a model when measured against the physical-time consumed in the CPU's clock. Furthermore, the timing defined for the atomic model does account for the simulation protocol itself. Physical-time is consumed for operations such as setting T_{min} and T_{max}. Thus, we need to include Δt_{rest} for these kinds of computations that must be completed in running simulation experiments. As noted in Section 3.1, the accuracy of time assigned to actions in an atomic model is imperfect.[28] This can cause undefined concurrent events or cause actual concurrent event to be lost for coupled models (see Section 5).

Execution time for all the operations in an atomic model is subject to the physical-time that can be accommodated in the hardware, hosting some implementation of the simulator protocol. The total time for an atomic simulation cycle $\Delta t_a > 0$ is equal to $\Delta t_{ext} + \Delta t_{int} + \Delta t_{rest}$. Assuming $\Delta t_{ext} + \Delta t_{int} \gg \Delta t_{rest}$, the real-time simulation of ALRT-DEVS atomic model can be accepted as executing in physical-time. The impact of timing and actions as defined in ALRT-DEVS is demonstrated with some experiments (see Section 5). The experiments show the degree to which real-time simulation can be achieved.[29]

4.2 Coupled coordinator protocol

The protocol for ALRT-DEVS coupled model is the same as parallel DEVS. In a coupled model, message routing from one model to another occurs instantaneously. Couplings in coupled models are timeless. They can be modeled as atomic models but inherently the coupled model specification requires input and output exchanges to be ordered, but untimed. We note that the coordinator protocol itself consumes physical-time even though it does consume real-time with respect to the simulation platform. As alluded earlier, execution time of the coordinator protocol is subject to uncontrollable factors as viewed in physical-time. $\Delta t_{I/O} > 0$ is defined to represent the time consumed for all message handling and transmission across external input, internal, and external output couplings. For a flat coupled model with one atomic model, the physical-time for its execution is defined as $\Delta t_{fc} = \Delta t_a + \Delta t_{I/O}$, where Δt_a is the physical-time allocated for execution of atomic models and $\Delta t_{I/O}$ for input and external output messaging. As in the atomic model, there exists extraneous operations that also consume physical-time, but not real-time while being executed in the simulation platform. For a flat coupled model with n atomic models, time for a flat coupled cycle is defined as $\Delta t_{fc} = \sum_{i=1}^{n} \Delta t_{a,i} + \Delta t_{I/O}$. In this case, $\Delta t_{I/O}$ accounts for all three kinds of coupling. The physical-time execution for a hierarchical coupled cycle with n atomic models, p flat coupled models, and q hierarchial coupled models is defined as $\Delta t_{hc} = \sum_{i=1}^{n} \Delta t_{a,i} + \sum_{j=1}^{p} \Delta t_{fc,j} + \sum_{k=1}^{q} \Delta t_{hc,k} + \Delta t_{I/O}$. It is obvious that the physical-time spent executing real-time atomic and coupled models exceeds the real-time consumed in the simulation platform. If $\sum \Delta t_a \gg \sum \Delta t_{I/O}$ for every flat coupled model and for

every hierarchial coupled model, then real-time execution of ALRT-DEVS coupled models can be accepted as executing in physical-time.

5. Example model

The network switch system[7] shown in Figure 5 is developed based on the following scheme. The network-on-chip (NoC) switch consists of input ports for receiving flits, output ports for sending the flits on the link, and a crossbar which connects the input ports to output ports. The switch depicted in Figure 5 has two virtual channels and operates using a 5-stage pipeline (routing, virtual channel (VC) allocation, switch (SW) allocation, SW transfer, link transfer). The input ports possess a number of (equal to the number of VCs) input queues and status arrays for the newly received flits and perform routing and allocation operations on them. Now, the complete model in Figure 5 contains a generator, a sink, two switches and a bus. The couplings between switches and the Bus/Gen/Sink are untimed, as opposed to the bus which has four links, each of which consumes time for transmitting flits. The switch is for a mesh-network with four inputs and outputs, each of which can be configured to have multiple virtual channels. The remaining parts of the switch are crossbar, router, virtual channel allocator, and switch allocator.[15,30] Select actions for these components are time-bounded. The flow control policy is on/off, in which the downstream switch (Switch 2) sends an ''off'' signal to the upstream switch (Switch 1) when its input queue is full so that Switch 1 stops sending it flits. The generator on the other hand, continuously injects new packets into the network without considering the current state of the switches. Without delving into further details, it is useful to note that the model is stochastic in terms of its actions (switches operate in parallel and flits may arrive at the same time and in arbitrary order). Execution is also stochastic due to the unpredictably of threads in the host simulation platform. During simulation some actions may be terminated if they cannot be completed given their designated time-windows.

5.1 Discussion

The configuration parameters of the models are listed in Table 1. The timing parameters are in real-time. Flits traversing from the generator to the switches and sink are assumed to consume no time, but they can be assigned to consume real-time. For this experiment, the injection rate is increased from 5 to 2000 flits per second and a 10-second sample is taken from the network. Based on the configuration provided in Table 1, the modeled system will be saturated quickly at high flit injection rates and consequently lose a high number of flits. The switch model is also developed in Booksim[31] and used to validate the one described here.[7]

For the purpose of comparison, this experiment is performed in both real-time and logical-time settings. The logical-time simulation illustrates the impact of the executing platform for real-time simulation. The models are executed using the DEVS-Suite 2.1.0 simulator[15] on a 32-bit Windows 7 machine with 2.93 GHz, Intel Core 2 Dual processor and 2 GB of DDR2 memory from which only 1.4 GB is used because of the 32-bit JVM. Due to stochasticity in the model, each experiment is repeated five times and the averages of all five experiments are shown in Figures 6 and 7. Multiple experiments are simulated to show the impact of variability in the simulation platform. If the switch model were to be used as a blueprint for implementing a time-critical system, as mentioned in Section 1, worst case execution time analysis and design of experiments among others are needed.

The reason we stopped at five runs for each experiment was the convergence between the results. Repeating the experiments beyond five data points did not add more to our data analysis and showed little variations. Although real-time analysis is mostly done via worst case analysis, for our analysis it was useful to consider averages. These experiments are mainly designed and carried out to show the capabilities of ALRT-DEVS for real-time modeling and simulation. These proof of concept experiments are not purposed for model verification and simulation validation using worse case time of the model or worst case execution time.

Figure 6 shows the drop rate in the system based on the generator's injection rate. Packet loss ratio for the first few injection rates is magnified to illustrate real-time and logical-time simulation executions. Although the data extracted from these simulations represent similar behaviors, a small difference can be observed. This difference is due to the impact of the executing environment on the simulation protocol. Each deadline miss leads to a flit loss; flit losses (although few) cause the simulation results to diverge from those obtained from the logical-time simulation. This impact may become significant if the executing environment is incapable of guaranteeing actions to be completed as required due to high model complexity and granularity or inadequate computational resources. A real-time simulation platform can exhibit significant unpredictability when model requirements cannot be met. This is not the case for logical-time simulation since time is not a limiting factor, time can be stretched or shrunk without any side effect on the simulation results.

Results from a real-time simulator can be categorized to fall into three regions. First, the simulation execution scenarios can be handled by the executing platform; simulation results are not impacted by limited resource availability. Second, the execution platform cannot meet some ideal timing or complexity but the observed system dynamics are within an acceptable error margin. Third, use of the execution platform leads to unreliable results.

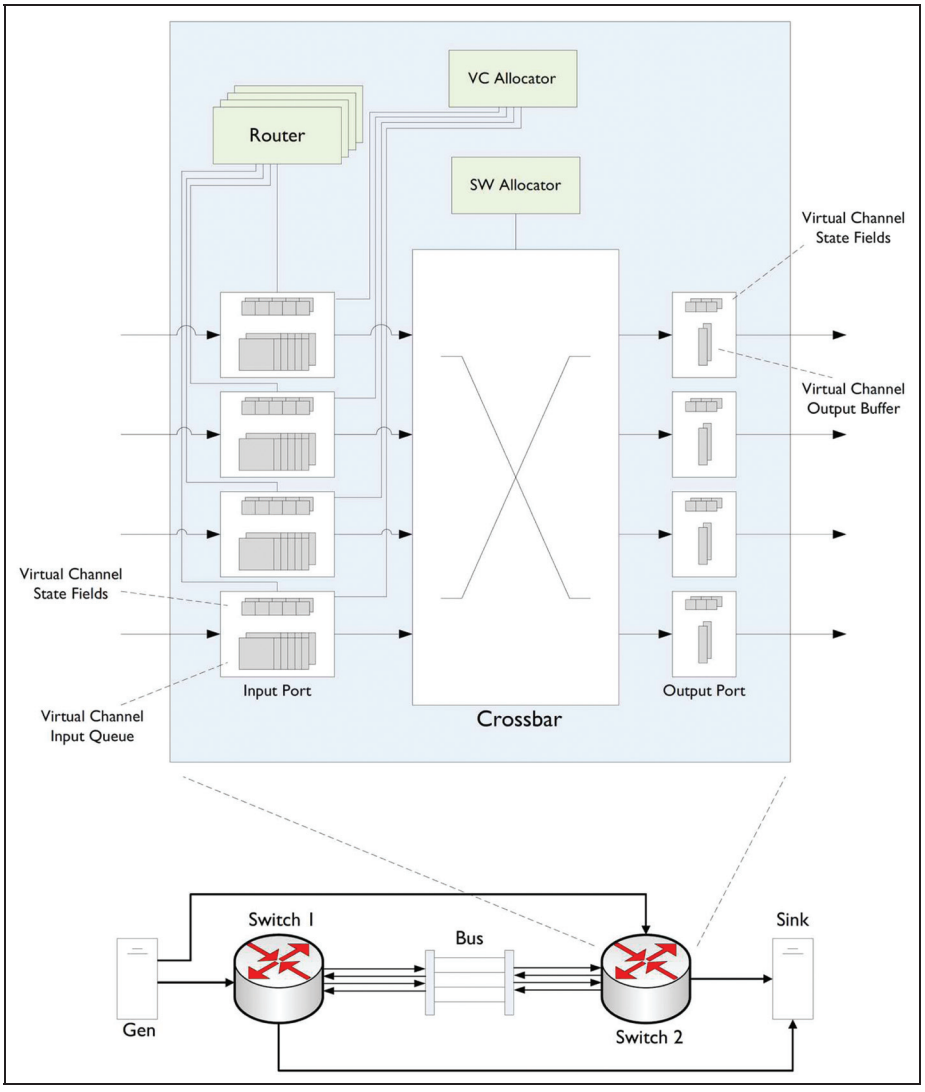

Figure 5. A switch network system.

These categories are also observed in non real-time simulation where observed results in the third category could be due to models that are under specified or are subjected to incorrect experimental settings.[32]

The above three regions are observed in the simulation experiments shown in Figure 7. However, in Figure 6, these regions are not visible since only flit loss ratio is being observed and visualized. In Figure 7(a), average flit latency for the first region is demonstrated (injection rate between 5 to 40 packets per second). In this region the results are ideal since no flit loss has occurred due to real-time execution and constraints in the computing platform. The difference between various injection rates is reflected in high, low, and average flit latency though they all belong to the same region. The results extracted from the third region (injection rates higher than 1000 packets per second) are unreliable. This is evident in Figures 7(b) to (e). In Figure 7(b), the number of deadline misses are recorded for various injection rates. The number of deadline misses experiences a significant increase at high injection rates. One interesting phenomenon in this diagram is the very small difference in deadline losses between 1500 and 2000 (flit per second) injection rates. The reason is that the underlying platform is so overwhelmed with the number of actions that some of them do not even begin to execute and therefore never lose their deadline. However, for the number of actions executed, the system shows the same behavior as shown in Figure 7(c) for the flits injected/lost ratio and in Figure 7(d) for the actual number of injected flits and those of them that are lost (overlapping lines). Figure 7(e) shows the chaotic nature of the simulation in the third region. An approximate number of 15,000 packets are expected to be injected into the system for a period of 10 seconds

Table 1. Network switch system configuration.

Topology and physical specification		Timing specification (10^{-3} s)	
Variable	Value	Variable	Value
		Routing	[9–11]
Virtual channels per link	1	Link traversal	[0.9–1.1]
Number of flits per packet	1	Status traversal	[9–11]
Buffer size	8	SW traversal	[9–11]
Number of input/output ports	4	VC allocation	[9–11]
		SW allocation	[9–11]

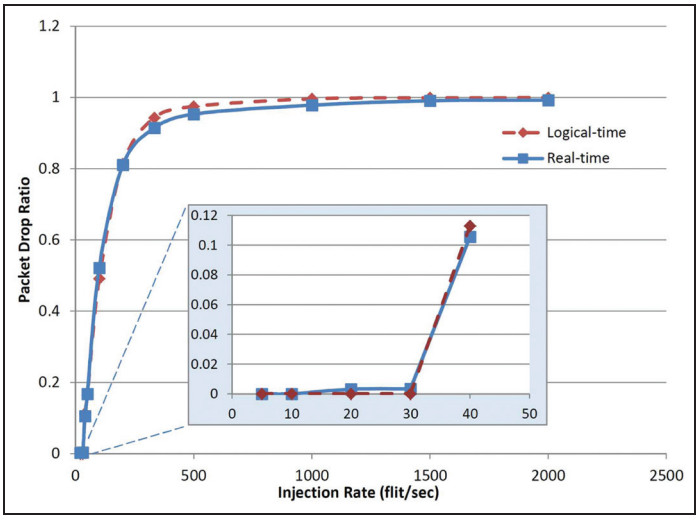

Figure 6. Packet loss ratio based on injection rate.

with an injection rate of 1500 packets per second. However, the number of injected packets demonstrate significant variations (from 5000 to 15,000) across five simulation runs. This variability in the number of injected packets coupled with the number of deadline misses demonstrates the unreliability of the simulation. Finally, Figure 7(f) represents the second region of the real-time simulation in which the results are still reliable but show a small divergence from the logical-time results. Here the difference between the total number of packet losses of logical and real-time simulator is depicted and fashioned in a stock diagram.

As discussed in previous sections, time can be categorized into physical-time, real-time, and logical-time. The third region in simulation occurs when the real-time clock in the simulation cannot stay synchronized with the physical-time. This may happen when executing the model exceeds the limits of hardware and/or software resources. Also, one of the other limiting factors in real-time simulation is handling the concurrency. P-DEVS is based on synchronized parallel simulator protocol. However, true concurrency cannot be achieved in the implementation of real-time simulators such as DEVS-Suite due to slicing of time periods for multiple actions. Therefore, ALRT-DEVS simulator cannot guarantee concurrency as in the parallel DEVS logical-time simulation. This can be observed in the second and third regions where real-time simulation measure diverges from its logical-time counterpart.

6. Related work

Model specifications and execution algorithms have been developed toward real-time modeling and simulation. We do not attempt to compare our work with these; a few from the point of view modeling are discussed in the literature.[7,29] We also refrain from discussing real-time distributed modeling and simulation approaches, an ongoing research area. Instead, we begin with some early works that were aimed at adopting logical-time modeling for real-time simulation through the 1990s. Starting in the late 1990s, there was a surge in developing DEVS-based tools based on classic and parallel DEVS formalisms and their extensions, such as fuzzy logic, probabilistic, and dynamic structure. Furthermore, our focus is not on simulation tools and applications, which are significant research areas in

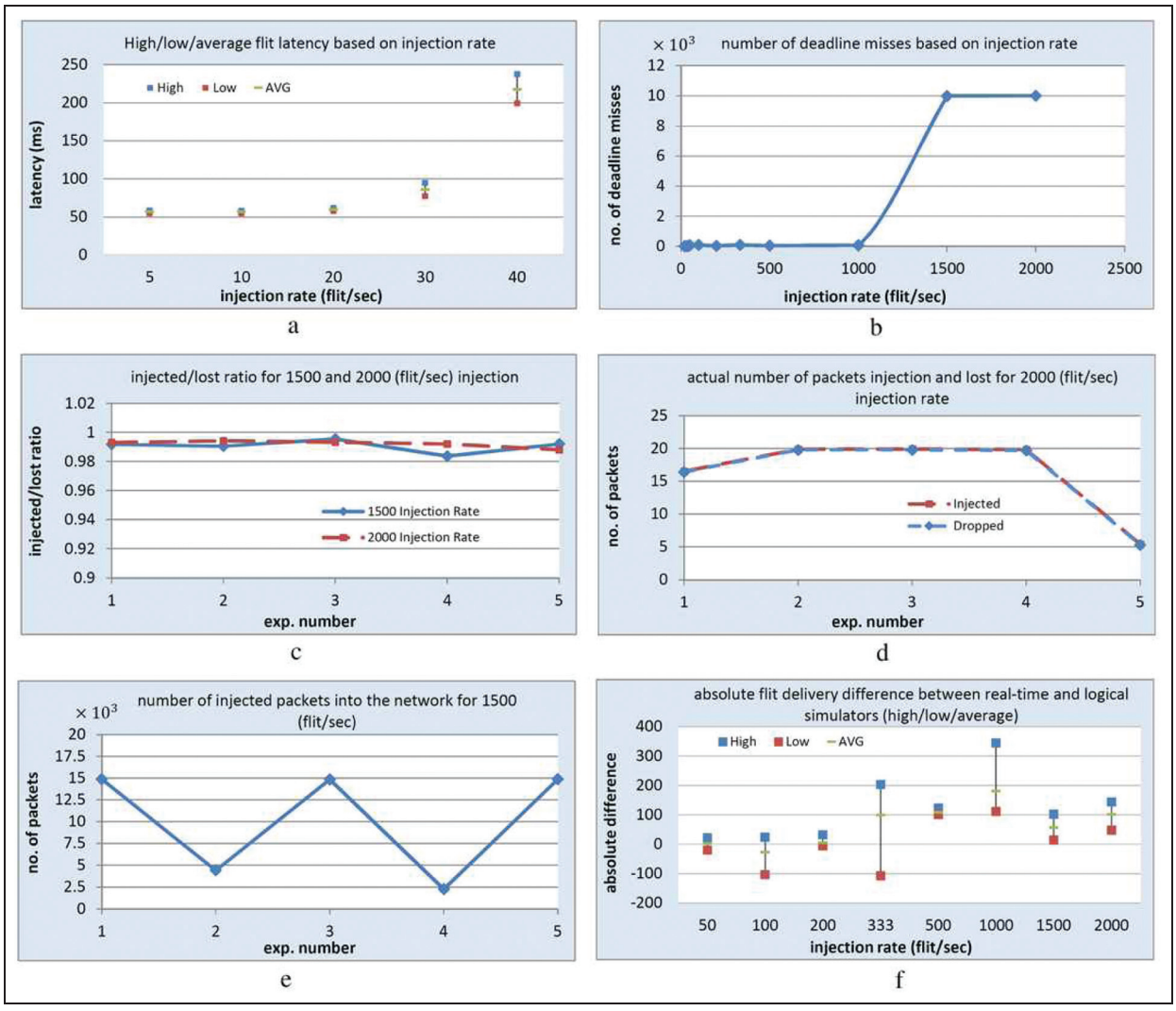

Figure 7. Simulation results for the model shown in Figure 5: (a) a stock diagram manifesting high, low, and average flit latency for the first region with zero packet loss (exponential growth; (b) significant increase in deadline losses for the third region; (c) ratio of the lost flits versus the injected ones for the third region at 99% ratio); (d) actual number of flits injected and lost in the third region (two overlapping lines); (e) variations in injection rate for the third region; and (f) depicting high, low, and average differences in the total number of packet loss between logical and real-time simulations for the second region of the network (small variations).

their own right. We use the DEVS-Suite simulator to support developing ALRT-DEVS models, simulate the example switch network model, and evaluate the simulation results to show the extent in which action-level real-time modeling and real-time simulation are supported.

Classical DEVS modeling implemented in the DEVS-Scheme simulator was developed to execute under the host computer clock.[33] The atomic model does not allow specifying actions. The simulator, which was developed in the Scheme language, uses an interface channel written in C programming language for handling analog signals converted to/from discrete data. Time instances for incoming and outgoing events in atomic and coupled models are coordinated using the host computer clock. In Section 2 we described parallel DEVS and RT-DEVS approaches. We noted that the parallel DEVS modeling formalism does not support representing actions. For example, although models developed for DEVS-on-a-chip simulator[12] built on the DEVSJAVA simulator[34] uses real-time Java, soft real-time execution is supported. RT-DEVS is based on classic DEVS, which precludes assigning sets of contiguous actions separately to external and internal transition functions. In particular, RT-DEVS requires use of the Select function for coupled models and does not support receiving multiple input and sending multiple output events at the same time instance. To handle timing for coupled models, a clock matrix is developed.[21] RT-DEVS supports non-blocking and synchronization loss. The clock matrix is

introduced as an analytic capability to predict, as precisely as possible, the upper and lower bounds for the next state transition times. The differences between RT-DEVS and ALRT-DEVS resulted in developing an abstract simulator protocol for ALRT-DEVS model.

Another approach to real-time classic DEVS modeling is proposed in eCD++.[35] This work expands on the DEVS-on-a-chip by flattening hierarchical models with soft real-time execution. To handle limited resource availability in a given computing platform, interface DEVS is defined.[13] It introduces a function for setting the execution of a state to be either optional or mandatory. Therefore, a state transition (corresponding to a simulation cycle) may not be executed depending on the simulator having enough time or not. The abstract simulator protocol is mapped to a real-time task list for executing atomic external and internal transition functions. A coupled model prioritizes which atomic models must be executed and which ones may be executed depending on the modes assigned to the state transitions. This approach delegates the choice of which state transitions are to be executed to the coupled model. Fewer resources are needed as the ratio of the required to the optional state transitions decreases. Thus, the main objective is reducing the number of state transitions to achieve hard real-time simulation. Real-time execution is achieved by embedding the simulator in network processors with time resolution in nanosecond range executing on RT Linux.[36] Unlike this eCD++, in ALRT-DEVS, actions can be defined with real-time time-windows with its abstract simulator enforcing hard real-time execution. Another DEVS approach to support real-time simulation is PowerDEVS.[37] It is implemented in C++ and uses a real-time clock provided by Linux RTAI. Generation and communication of the output events are synchronized using the aforementioned real-time clock. As in RT-DEVS and eCD++, this approach focuses on using simulator clock with interrupt handling whereas ALRT-DEVS focuses on defining the external and internal transition functions with actions each of which is bounded by a real-time clock.

Real-time modeling and simulation has also been the subject of study in the Ptolemy II project.[38] As in some DEVS-based tools, Ptolemy II supports soft real-time simulation.[29] For hard real-time execution, a Giotto model of computation has been developed in Ptolemy.[39] The Giotto programming model[40] with periodic timing is developed where tasks defined in a Giotto Director coordinate the execution of the actors based on their assigned frequencies or default values. Ptolemy's code generation capability is used to generate C code. The FreeRTOS,[41] which is portable to common target microcontroller platforms, executes the generated code. The program code executes correctly provided there exists a scheduling policy that can meet the requested deadlines given the resources and time constraints of the target platform. As in ALRT-DEVS, it may not be feasible to identify worst-case execution time. In such cases, the execution of tasks are not terminated; instead, the execution continues generating outputs with delays. In ALRT-DEVS, the actions that cannot be completed within the available resources are terminated or ignored. Another difference between these approaches is that the Giotto model requires time-periodic cycles which determine the scheduling policy that can be supported in FreeRTOS. In ALRT-DEVS, timings for actions are event-based and subject to the clock resolution of the computing platform. The allocation of time for the actions is determined by the number of threads (i.e., the number of atomic models) that are not scheduled to execute in any particular order even though the sequence of actions (based on the remaining time) is defined in the external and internal transition functions.

7. Conclusions

We have proposed an approach for developing DEVS models where actions, in addition to state changes, can be specified. Timings in the form of time-windows are assigned to actions. These actions are associated with individual state changes using location and time-invariant concepts developed for real-time statecharts. ALRT-DEVS offers a structured approach for decomposing external and internal transition functions to sets of real-time time-constrained and schedulable actions. To simulate these models, a real-time abstract simulator protocol is proposed based on common virtual multi-threading computing platform and implemented as part of the DEVS-Suite simulator. The simulation protocol can enforce certain hard real-time execution to the degree supported by the computing platform and implementation of the models and the simulator protocol. The action-level real-time modeling is independent of specific hardware and software technologies. Thus, ALRT-DEVS can lend itself to the development of hard real-time computing platforms using real-time operating systems to achieve guaranteed clock accuracy and precision. This is because specification of actions with constrained real-time can be mapped one-to-one to hard real-time simulation cycles. It is necessary to develop analytical techniques to determine (near-)optimal hard real-time timings for actions given worst-case execution times available. ALRT-DEVS approach stands to benefit from code generation targeted for network processors with high precision clocks, specialized hardware, and real-time operating systems. Developing scheduling policies for computing platforms is another important area of research where simulating safety critical systems can be performed as close as possible to physical-time. In terms of use, action-level, real-time modeling has the potential to be used in building simulations that can interact seamlessly with physical systems.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

1. Eidson JC, Lee EA, Matic S, et al. Distributed real-time software for cyber–physical systems. *Proc IEEE* 2012; 100: 45–59.
2. Lamport L. Time, clocks, and the ordering of events in a distributed system. *Commun ACM* 1978; 21(7): 558–565.
3. Lee EA. Computing needs time. *Commun ACM* 2009; 52(5): 70–79.
4. Sastry S, Sztipanovits J, Bajcsy R, et al. Scanning the issue: Special issue on modeling and design of embedded software. *Proc IEEE* 2003; 91: 3–10.
5. Cervin A and Eker J. Control-scheduling codesign of real-time systems: The control server approach. *J Embedded Comp* 2005; 1: 209–224.
6. Cunning SJ, II Shultz S and Rozenblit JW. An embedded system's design verification using object-oriented simulation. *Simulation* 1999; 72: 238–249.
7. Gholami S and Sarjoughian HS. Real-time network-on-chip simulation modeling. In: *SIMUTools*. Desenzano: ICST, 2012, pp. 103–112.
8. Wang Q and Cellier FE. Time windows: An approach to automated abstraction of continuous-time models into discrete-event models. *Int J Gen Syst* 1991; 19: 241–262.
9. Hong J, Song H, Kim TG, et al. A real-time discrete event system specification formalism for seamless real-time software development. *Discrete Event Dyn Syst* 1997; 7: 355–375.
10. Zeigler BP, Kim TG and Praehofer HS. *Theory of modeling and simulation*. 2nd ed. Orlando, FL: Academic Press, 2000.
11. Harel D and Politi M. *Modeling reactive systems with statecharts: The STATEMATE approach*. New York: McGraw-Hill, 1998.
12. Hu X, Zeigler BP and Couretas J. DEVS-on-a-chip: Implementing DEVS in real-time Java on a tiny internet interface for scalable factory automation. In: *IEEE international conference on systems, man, and cybernetics*, volume 5. IEEE, 2001, pp. 3051–3056.
13. Moallemi M and Wainer GA. Designing an interface for real-time and embedded DEVS. In: *Proceedings of the 2010 Spring Simulation Multiconference*, New York: ACM Press, 2010, pp. 137:1–137:8.
14. Ptolemaeus C. *System design, modeling, and simulation: Using Ptolemy II*. ptolemy.org, http://ptolemy.org/systems, 2014.
15. ACIMS, DEVS-Suite Simulator, http://devs-suitesim.sourceforge.net/ (2015).
16. Burmester S, Giese H and Schäfer W. Code generation for hard real-time systems from real-time statecharts. Technical Report TR-RI-03-244, University of Paderborn, Germany, 2003.
17. Henzinger TA and Kirsch CM. The embedded machine: Predictable, portable real-time code. In: *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. New York: ACM Press, 2002, pp. 315–326.
18. MathWorks. Simulink coder, http://www.mathworks.com/products/rtw/ (2012).
19. Chow A and Zeigler BP. Parallel DEVS: A parallel, hierarchical, modular, modeling formalism. In: *Proceedings of the 26th conference on Winter simulation*. Society for Computer Simulation International, 1994, pp. 716–722.
20. Kim S, Sarjoughian HS and Elamvazhuthi V. DEVS-suite: A simulator supporting visual experimentation design and behavior monitoring. In: *Proceedings of the 2009 Spring Simulation Conference*. Society for Computer Simulation International, 2009, pp. 29–36.
21. Song HS and Kim TG. Application of real-time DEVS to analysis of safety-critical embedded control systems: Railroad crossing control example. *Simulation* 2005; 81: 119–136.
22. Giese H and Burmester S. Real-time statechart semantics. Technical Report TR-RI-03-239, University of Paderborn, Germany, 2003.
23. Alur R and Dill D. A theory of timed automata. *Theor Comput Sci* 1994; 126: 183–235.
24. Abadi M and Lamport L. An old-fashioned recipe for real time. *ACM Trans Program Lang Syst* 1994; 16: 1543–1571.
25. Lee I, Wolfe V and Davidson S. Motivating time as a first class entity. Technical Report MS-CIS-87-54, University of Pennsylvania, 1987.
26. Fujimoto R. *Parallel and distributed simulation systems*. New York: John Wiley & Sons, 2000.
27. McLean T and Fujimoto R. Repeatability in real-time distributed simulation executions. In: *Proceedings of the fourteenth workshop on Parallel and distributed simulation*. Los Alamitos, CA: IEEE Computer Society Press, pp. 23–32.
28. Vicino D, Dalle O and Wainer GA. A data type for discretized time representation in devs. In: *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques (SIMUTools'14)*, Brussels, Belgium. Belgium: ICST.
29. Gholami S and Sarjoughian HS. Observations on real-time simulation design and experimentation. In: *Symposium on Theory of Modeling and Simulation*, San Diego, CA, pp. 1–8. New York: ACM Press.
30. Gholami S and Sarjoughian HS. Network-on-Chip (NoC) simulation with ALRT-DEVS: Supplementary material and experiments, 2013. URL http://acims.asu.edu/wp-content/uploads/2014/02/ASUCIDSE-CSE-2013-001.pdf.
31. Dally W and Towles B. *Principles and practices of interconnection networks*. San Mateo, CA: Morgan Kaufmann, 2004.
32. Sarjoughian HS, Hild DR, Hu X, et al. Simulation-based SW/HW architectural design configurations for distributed mission training systems. *Simulation* 2001; 77: 23–38.
33. Zeigler BP and Kim J. Extending the DEVS-scheme knowledge-based simulation environment for real-time event-based control. *IEEE Trans Robotic Autom* 1993; 9(3): 351–356.
34. Zeigler BP, Sarjoughian HS and Au V. Object-oriented DEVS. In: *AeroSense'97*. International Society for Optics and Photonics, 1997, pp. 100–111.
35. Yu YH and Wainer GA. eCD++ : An engine for executing DEVS models in embedded platforms. In: *Proceedings of the 2007 Summer Computer Simulation Conference, SCSC* 2007,

San Diego, CA, 16–19 July 2007, pp. 323–330. San Diego, CA: Simulation Councils.
36. Castro R, Ramello I and Wainer GA. Modeling and simulation based design of real-time controllers embedded on network processors. In: *Proceedings of Spring Simulation Conference (SpringSim12)*, pp. 1–9. Society for Modeling and Simulation International (SCS), ACM.
37. Bergero F and Kofman E. PowerDEVS: A tool for hybrid system modeling and real-time simulation. *Simulation* 2011; 87: 113–132.
38. Lee EA, et al. The Ptolemy project, http://ptolemy.eecs.berkeley.edu/ (2013)
39. Forbes S. *Real-time C Code Generation in Ptolemy II for the Giotto Model of Computation*, http://books.google.com/books?id=2KxMtwAACAAJ (2009).
40. Henzinger TA, Horowitz B and Kirsch CM. Giotto: A time-triggered language for embedded programming. In: *Embedded Software*. New York: Springer, pp. 166–184.
41. Openrtos. The FreeRTOS project, http://www.freertos.org (2013).

Author biographies

Hessam S Sarjoughian received a PhD in electrical and computer engineering from the University of Arizona, Tucson, in 1995. He is an associate professor of computer science and engineering at Arizona State University in Tempe, Arizona. Sarjoughian is co-director of the Arizona Center for Integrative Modeling and Simulation (ACIMS). His research focuses on model composability, visual modeling, service-oriented simulation, and distributed co-design modeling. He led the development of the Online Masters of Engineering in Modeling and Simulation in the Fulton School of Engineering at ASU in 2004. He was among the pioneers who established the Modeling and Simulation Professional Certification Commission in 2001. His research has been supported by NSF, Boeing, DISA, Intel, Lockheed Martin, Northrop Grumman, and the US Air Force.

Soroosh Gholami received his BSc in 2010 and his MSc in 2012, both from the University of Tehran, Iran. He is currently a PhD student at Arizona State University. As a member of ACIMS, he has been involved in various modeling and simulation research projects on hardware chips, supply chain, and environmental phenomenon supported by Intel and NSF.