# Software and Simulation Modeling for Real-time Software-intensive System

Dongping Huang
*Computer Science & Engr. Dept.*
*Arizona State University, Tempe, AZ*
*Dongping.Huang@ASU.EDU*

Hessam Sarjoughian[1]
*Computer Science & Engr. Dept.*
*Arizona State University, Tempe, AZ*
*Sarjoughian@ASU.EDU*

## Abstract

*Successful development of large-scale complex and distributed real-time systems commonly relies on models developed separately for simulation studies and software implementation. Systems theory provides sound modeling principles to characterize structural and behavioral aspects of systems across time and space. The behavior of these models can be observed using simulation protocols that can correctly interpret time-based logical dynamics. Similarly, object-orientation theories and software architecture principles enable modeling static and dynamic behavior of systems. While models described either in system-theoretic or object-orientated languages may be used for both software design and simulation modeling, each has its own strengths and weaknesses. For example, a class of system-theoretic modeling approach called Discrete-event System Specification (DEVS) provides an appropriate basis to develop simulation models exhibiting concurrent and distributed behavior. Similarly, the Unified Modeling Language with real-time (UML-RT) constructs can be used to develop software design models that can be implemented and executed. Since software models are not suitable to be used as simulation models and simulation models may not adequately lend themselves to serve as software design blueprints, it is important to examine these approaches. We show some of the key shortcomings of these simulation and software design modeling approaches by developing some detailed specifications and implementation of a coffee machine with a focus on their treatment of logical and physical time.*

## 1 Introduction

The existing body of theories, methods, and technologies that have been developed over the last two decades support analysis, design, and development of non-real-time software systems. Unfortunately, the same cannot be said about real-time software-intensive systems since their designs remain chiefly ad-hoc. Real-time software-intensive (embedded) system design must account for new kinds of complexity due to timing constraints. Specifications for such systems need to support structural and behavioral modeling of components and their interactions while rigorously accounting for time – i.e., it is not sufficient to model time informally or in abstract terms. For example, while constructs such as "always" and "next-time" [1] can be used to specify time-based behavior of systems, they are not sufficient for precise specification of time constraints using Object-Orientation frameworks such as UML [2]).

Systems theory offers fundamental concepts and theories of sub-systems (components), their compositions, and interactions with one another through well-defined interfaces or ports (e.g., see [3]) which are governed by time. However, system-theoretic specifications do not account for the computational basis required for software design and implementation.

Given the capabilities and limitations of object-orientation and systems theory, it is useful to investigate their relationships and in particular *(i)* the applicability of each for the analysis and design of software-intensive systems and *(ii)* their complementary roles in handling modeling and computational complexity of embedded systems. To achieve our objective, this paper focuses on the capabilities of the UML-Real-time – an object-oriented software specification with real-time object-oriented modeling framework – and DEVS (Discrete-Event System Specification) [4] – a system-theoretic modeling and simulation framework.

## 2 Background

Systems theory [3-5] defines a system in terms of its structure and behavior. A system has a hierarchical structure and can have continuous and discrete behaviors. An important feature of systems theory is explicit support for time – i.e., the semantics of models are directly related to the passage of time and causality of outputs with respect to states and inputs. Components (representing sub-systems) can interact with each others via input and output ports. Models which are composed using these components have sound syntax and semantics and lend themselves to well-defined computation. For example, Discrete-event System Specification (DEVS) [4] supports modeling a set of hierarchical interacting components that are able to respond to external stimuli and exhibit

[1] Corresponding author.

*Accepted for Publication*

autonomous behavior. Behavior and structure of the models are described as atomic and coupled models in environments such as DEVSJAVA [6]. Parallel atomic models can have multiple ports and can process bags of inputs and produce bags of output events. Parallel hierarchical coupled models can be composed of atomic and coupled models with three constraints – (i) atomic models can be at the lowest level in any coupled hierarchical model, (ii) no coupled model can contain itself as a component, and (iii) no direct output to input coupling is allowed for atomic or coupled models.

Modeling and simulation approaches are suitable and widely used for modeling real-time systems. For example, DEVS and its extension Real-time DEVS [7-9] can be used to model time as either abstract or physical entity. These modeling and simulation frameworks have been proposed and developed to support specification and simulation of atomic and coupled models under real-time constraints. We note that recent extensions to DEVS [10] support modeling the class of ordinary differential equations and discrete-time models can be automatically mapped onto discrete-event models.

System theoretic based modeling and simulation framework, however, are not inherently intended for software design and development [11]. Software design blueprints must specify details that are beyond the scope of simulation modeling. Simulation models are extensively used for requirement analysis and evaluation of alternative system designs at relatively high levels of abstractions.

Compared to system-theoretic modeling, software modeling is concerned with software specification and implementation. For example, object-oriented modeling supports characterizing components and their compositions. Object-orientation approaches such as UML offers constructs to model relationships among components that are not possible in system-theoretic approaches. Using UML we can design logical, non-real-time dynamics of software using inheritance, aggregation, and realization relationships that are appropriately defined among classes, interfaces, sub-systems.

For real-time applications, formal timing and concurrency modeling constructs are crucial and various efforts are underway to extend OO principles and methods in order to formally account for time. For example, UML-RT [12] extends UML by incorporating the Real-time Object-Oriented Modeling (ROOM) [13]. In UML-RT Profile [14], the time, schedule and performance related properties that have already been captured are quantitatively codified using standard UML stereotypes, tagged values, and constraints. Nonetheless, there does not exist first-class concepts and constructs for time. The UML-RT – which is gaining popularity as a commercial tool in developing real-time software specifications – offers the system-theoretic concepts of ports and hierarchical composition. Other ongoing efforts are focusing on combining UML with formal languages such as Object-Z (e.g., [15]) or Specification Description Language (SDL) (e.g., [16]). In addition to these, the Model Driven Architecture [17] proposes a framework to bring together Platform Independent Models of business logic to Platform Specific Models of the underlying software architecture which can automatically be mapped into application code. Another research effort is developing a precise and multi-level meta-modeling framework to visually capture the transformations by using graph transformation [18]. Other approaches, primarily driven by the DEVS framework, offer some capabilities to map simulation models to executable code (e.g., [9, 19]).

## 2.1 Modeling time

The specifications of real-time systems or simulation models must account for concurrency, synchronization, and causality among a set of interacting components. A unifying concept for concurrency, synchronization, and causality is time. However, different concepts are used to specify simulation models and real-world (physical) systems. For example, the "happened before" relation in logical time can be used to define the causality relation on the set of events in which an event is only influenced by the past and current events; it cannot be influenced by future events [20]. Events in the same process are causally related, but events in distributed processes could be causally dependent or not. Causally independent events can be thought as concurrent events. The reverse is not true as concurrent events can also be causally dependent. In real-time distributed systems, all the events among the processes may be ordered by the causality relationship. For the causally independent events with identical logical clocks, they can be ordered by taking into account the identifier of processes.

From the discrete event simulation worldview, three notions of time have been defined (see [21] for a detailed treatment). These are physical, simulation, and wallclock times. Physical time refers to time in the physical (real-world) system. Simulation time is introduced as an abstraction of physical time. Wallclock time is used to mark the beginning and the end of a simulation execution. The wallclock time and simulation time have a linear relationship – appropriate correspondence between the physical and the simulation times is ensured. From system-theoretic point of view, the dependencies between components are allowed only through the input/output couplings. Therefore, the model's notion of time, by itself, may not be used directly to deduce causality. In logical processes worldview of parallel discrete events simulation, causally dependent events are identified using event time stamping or ordering.

## 3 Related work

Real-time system development requires the concepts and techniques offered by systems theory and object orientation. To make use of capabilities afforded by these worldviews, researchers have proposed and developed approaches to support both simulation and software development of software-intensive systems. For example, several improvements and extensions of the Unified Process by using the system theoretical methodology for object oriented software analysis [22] have been proposed. I-Logix [23] has developed a framework that integrates systems engineering and software engineering together. The first part of the process focuses on requirement analysis, system analysis and design. Software design and implementation are the focus of the second part of the process. To support the process activities, two software tools have been developed. First, the Statemate MAGNUM is a visual modeling and simulation tool for system engineers. It focuses on system model analysis and design and it allows capturing graphical description of the system prototype using use-case models and scenarios, activity chart, control block diagram and statechart. The environment provides support for maintaining consistency among these models. Second, the Rhapsody implements UML 2.0 and provides a model-driven development environment for software engineering. It supports modeling software design and implementation. The formal languages of activity charts and Statecharts in the Statemate enable the models' execution and verification using mapping rules [24].

In contrast to I-Logix, Ptolemy II [25] provides a unified development environment for both system engineering and software engineering. Ptolemy project [25] specifically is targeted for real-time embedded systems modeling, simulation and design by coupling system designs with control model designs. It supports a hierarchical heterogeneous software environment for model refinement and system implementation. Ptolemy's actor-oriented approach is quite similar to the concepts developed in ROOM. It separates functional models (actors) from component interaction models (frameworks). A framework defines a set of constraints on how actors can interact with another [25]. As a computation model, a framework must maintain a set of states and their transitions. In theory, frameworks may be combined to support heterogeneous models of computations. For an actor transition, the framework state needs to be considered as guard conditions; for a framework transition, actor states also need to be considered. Therefore, it is difficult to maintain consistency between the framework states and the actor states as a model's complexity grows.

Alternatively the DEVS framework can be extended with capabilities for real-time execution (e.g., [9]). While this approach offers important features, its support for software design is limited. In particular, the supported modeling constructs are limited compared with those available from UML (e.g., in software design and implementation it is beneficial to use interfaces, something that is not part of the DEVS framework).

## 4 Model development in UML-RT & DEVS

As we have noted above, object-oriented and system-theoretic approaches allow modeling of real-time embedded systems. In this section, we develop two sets of models using UML-RT and DEVS. Specifications of these models reveal key aspects of each approach. In particular, model specifications serve to highlight strengths and weaknesses of each approach from developing simulation models and software models.

A Coffee Machine is a simple, yet illustrative system for examining the modeling and simulation/execution capabilities of the DEVS and the UML-RT approaches. The Coffee Machine hardware consist of a water pot, a filter container, a coffee filter, a warming plate, a boiler, a brew button, and an indicator light. The Coffee Machine's software controls its operations. A typical scenario for brewing coffee starts with placing a filter in the filter holder, filling it with coffee grounds, and sliding the filter holder into its receptacle. After pouring water into the water strainer, the brew button can be pressed to start brewing coffee. Once the water reaches boiling, it is sprayed over the coffee grounds and coffee starts to drip through the filter and into the coffee pot. The coffee pot is kept warm for extended periods by a warmer plate, which only turns on if there is coffee in the pot. If the coffee pot is removed from the warmer plate while coffee is brewing, the flow of water is stopped in order to avoid spilling coffee on the warmer plate.

A set of sensors monitor the warmer plate, the boiler, the brew and the indicator buttons. Similarly, a set of actuators control the warmer plate heating element, the boiler, and the brew button, and indicator lights. Text/graphic user input and output is used to simulate the behavior of the Coffee Machine. We model the software aspect of the Coffee machine controller as well as the sensors and actuators. The controller consists of four parts: front panel (controls the buttons and display light indicator), sprayer controller (controls the boiler spraying), warmer controller (controls pot warmer), and brewer controller. The brewer controller coordinates the front, sprayer, and warmer controllers.

### 4.2 UML-RT modeling approach

The UML Real-Time, which is an extension of the UML offers additional modeling constructs based on Real-time Object-Oriented Modeling [26] and is targeted for

modeling complex, event-driven, and distributed real-time systems [26, 27]. It supports specifying both logical and physical aspects of real-time systems. UML-RT can be used to model individual structural and behavioral aspects of systems and their relationships. In the following sub-sections, we use Rational Rose RealTime software [12] to model the Coffee Machine.

4.2.1 **Structural modeling.** Capsule, Port, and Connector are important concepts for specifying structure models. Capsule is one of the basic UML-RT modeling elements that represent an active object within a system communicating with other capsules exclusively through one or more interface objects that are called ports. For example, CM_Container (Coffee Machine Container) and CM_Brewer (Coffee Machine Brewer) represent the highest level software model of the Coffee Machine and one of its components, respectively. No public attributes or methods can be defined in capsules. Every port (e.g., ctr_Warmer) is "owned" by a capsule (e.g., CM_Brewer) in the sense that the capsule and its port have the same lifetime – i.e., ports cannot be defined independently of their capsules. The signal messages that are communicated between the ports are defined in Protocol, which is the main concept when specifying the behavior. Capsules and protocols (e.g., CUControl_Warmer) can be shown using standard UML graphical notations (class diagrams). Capsule structural diagram can be defined graphically as shown in Figure 1.

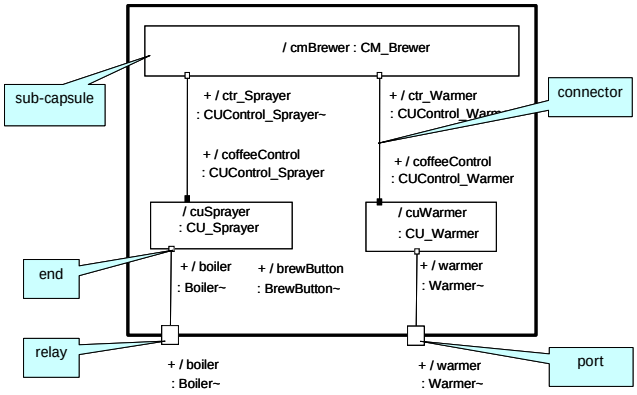


**Figure 1: Structure diagram with capsule composition, ports and protocol specification.**

Sub-capsules can be hierarchically combined into capsules (see Figure 2). Connector is used to represent interaction channel among parent and its immediate sub-capsules as well as among sub-capsules. Two types of ports are defined: *relay* and *end* ports. A relay port transmits signals between a capsule and its sub-capsule. An end port is responsible for processing the signals in the capsule's statechart. Capsule supports both static and dynamic component constructions. The behavior specification inside the capsule implementation is defined as a statechart (see Section 4.2.2). It should be noted that UML modeling capabilities such as inheritance can be used in UML-RT to facilitate structural specification. For example, protocol can take advantage of inheritance to implement message reuse.

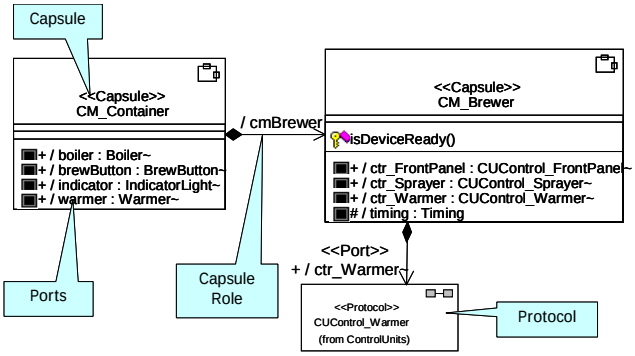


**Figure 2: UML-RT Hierarchical Model with Capsules, Ports and Connectors.**

**4.2.2 Behavioral modeling.** Protocol and Statechart are two essential concepts necessary for specifying behavior in UML-RT. A Protocol is a specification of contractual agreement among the participating capsules, in which the allowable input/output messages are defined based on their roles. Only ports that have compatible protocol roles can communicate with one another. A protocol consists of participants where each has a specific role in the protocol. A protocol role has a unique name and a set of signals that it can receive and send. Protocol has its own statechart to specify the sequence of signal communication. The most common type of protocol is a binary protocol which is defined to have two participants. A binary protocol must have one role (base role) and it may also have a second role (conjugate role). The conjugate role has its incoming message set and the outgoing message set opposite of the base role. In addition, a protocol may be specified to have a restrict set of communication sequences given its state machine. However, the quality of service specification (e.g., message priority) and finer grain specification of all valid message sequences allowed by the protocol (sequences) is not supported.

Each capsule or sub-capsule behavior is specified with a statechart. In UML-RT the trigger events in a capsule's statechart are defined with a port and signal pair. The action is specified by a specific programming code, which is then used for execution. A statechart diagram for the CM_Brewer controller is shown in Figure 3.

As UML-RT provides run-to-completion mechanism for a statechart to process a single event at a time, the concurrency can be due to multiple capsules executing simultaneously. That is, UML-RT supports concurrency to

the same extent as defined by UML. An important shortcoming for modeling concurrency is the inability to guarantee processing of events using priority settings.

As discussed earlier, developing real-time systems requires modeling of time and therefore the necessity of having time as an artifact. UML-RT provides timing service through a standard port called SAP (Service Access Point). The timing service converts time (time provided by the target operating system) into events that can then be handled in the same way as other signal-based events. The resolution (granularity) of time is dependent on the target platform or operating system. Since time is a service provided by the target platform, it cannot be readily manipulated – e.g., making the clock run slower.

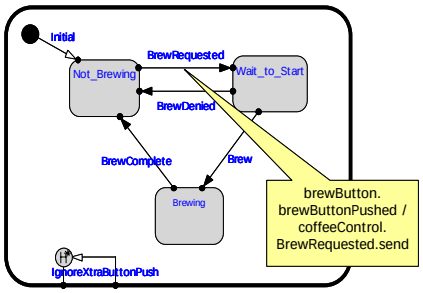


**Figure 3: Statechart diagram for brewer controller.**

To access the timing service of UML-RT, a timing port with a pre-defined Timing protocol can be added to each capsule. Service requests are activated by operation calls to this port (referenced by its name) while replies from the timing service are sent as messages that arrive through the same port. Two types of time-based situations can be modeled in UML-RT: the expiration of a specific time interval and the occurrence of a set instance in time. If a timeout occurs, the capsule instance that requested the timeout receives a message with the pre-defined message signal 'timeout'. Then a transition with a trigger event for the timeout signal must be defined in the behavior in order to receive the timeout message.

A timing service request results in the creation of a dedicated "timer" for the requesting capsule. Therefore each request has its own timer so that concurrent timing requests can be supported. However, since these timers are independent of one another, it is the modeler's responsibility to define suitable timing-constraint among protocols and capsules.

Models can be forward engineered and converted to executable code. The real-time execution is through the timing service of the UML-RT software environment. However, while UML-RT offers important capabilities to model real-time software, it does not provide suitable semantics suitable for simulated time – i.e., without the concepts of physical, simulation, and wallclock times, UML-RT prohibits carrying out simulation studies.

## 4.3 DEVS modeling approach

The Discrete Event System Specification is a mathematical formalism targeted for developing simulation models. The formalism, based on general system theoretic concepts, supports developing models with well-defined behavior [4]. It uses the mathematical set theory and offers a framework for system modeling and simulation. In this framework, every model component is either an atomic model or a coupled model. Atomic model defines a set of inputs with ports and messages, a set of outputs with ports and messages, a set of states, internal transition function mapping from states and time to states, external transition function mapping from states, inputs, and time to states, output functional mapping from states to outputs. In addition, in order to model time, each atomic model has a time advanced function which maps a state to real numbers.

4.3.1 **Structural modeling.** Port is the key concept for modeling the structure of atomic models. Coupling is the key concept important for when modeling the structure of a coupled model. Instead of providing duplex ports as in UML-RT, DEVS distinguishes ports by its communication direction. Input ports only receive messages while output ports only send messages. DEVS formalism doesn't specify any constraints on the message types coming in or going out of a specific port. Therefore, couplings do not specify what kind of messages may be sent or received. Instead, it is assumed the consumer of the message is able to determine whether or not a message has an appropriate type. This implies that producer of the message and its consumers must be consistent with one another. The Brewer Controller atomic model specification is implemented in DEVSJAVA [6]. The CM-Brewer external transition function (delt_ext) is shown in Figure 4. This function specifies that when the CM-Brewer receives an external event Msg.On on input port fromSprayer, it creates two messages, sets the state of the model to Checking and schedules an internal transition with time advance 0. Before the execution of the internal transition function, the simulator sends the output messages job_sprayer and job_warmer on output ports toSprayer and toWarmer based on the CM_Brewer output function (see Figure 5).

DEVS supports communication among multiple ports so one input port can receive messages from multiple output ports. Due to fan-in and fan-out couplings, the same message sent from two or more atomic models need to be identified based on their senders. In UML-RT, this is unnecessary since each protocol is associated with only one pair of ports. The coupling is similar with the connectors specified in UML-RT except that there is no constraint on port communication as long as input

messages go to input ports while output messages go to output ports.

```
public void delt_ext (double e, message x) {
  if (phaseIs("Passive")) {
   for (int i = 0; i < x.getLength(); i++){
     if (messageOnPort(x, "fromSprayer")){
       job = x.getValOnPort("fromSprayer", i);
      if (job.getName().equals(Msg.ON)){
        job_sprayer = new entity(Msg.CHECK_READY);
        job_warmer = new entity(Msg.CHECK_READY);
        state = "Checking";
        holdIn("SendOut", 0);
}}}}
```

**Figure 4: Snippet of the DEVS CU_Brewer model**

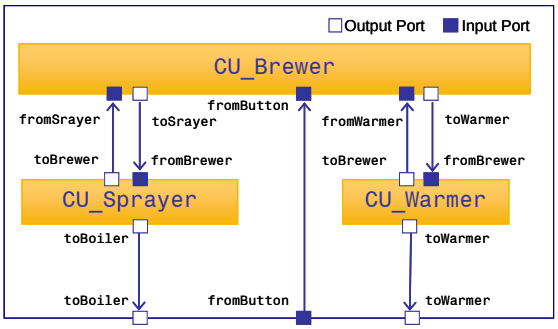


**Figure 5: DEVS CM_Controller coupled model**

4.3.2 **Behavioral modeling.** Internal transition, external transition, confluent, output and time advance functions are all used to specify the behavior of an atomic model. An atomic model has a state set which includes two special state variables *phase* and *sigma* and other state variables. The temporal property of the framework does not require customized treatment of time across all the models. Furthermore, since time is a first-class modeling artifact, the formalism provides well-defined semantics for modeling events that may be processed in zero time intervals (i.e., it supports transient processing of events).

# 5 Simulation and execution of models

To observe behaviors of a model, its needs to have an implementation that lends itself to execution. For models described using a given modeling and simulation approach (e.g., DEVS), there needs to exist a simulator that can correctly interpret (i.e., execute) the model. A simulator subjects a model to input stimuli and computes its resulting output. Correspondingly, an execution engine must exist to execute models described using a software modeling methodology such as UML-RT.

## 5.4 UML-RT model execution

Models specified in UML-RT may be executed using the UML-RT virtual machine or framework. This requires mapping suitable models (capsules and statecharts) into a high level language such as Java or C++. With models expresses in appropriate programming language syntax, they can be executed using a number of services such as communication and timing. The framework consists of Frame Service, Timing Service, Communication Service and other services through the Service Access Point [26]. The UML-RT framework provides the necessary capabilities for executing real-time models specified in the Rational Rose Real-Time. The framework services are coordinated by a central controller system, where the main control of the application is held. The application model is converted to a sub-application class inherited from the framework classes. This allows invoking the framework to execute the functions in the application to pass control to the application objects as required. After it processes all the messages in the queue, the application model will return the control back to the central controller.

UML-RT uses an asynchronous execution model, in which, the time instant at which a component reacts to a received message is left unspecified; it only defines that the reaction happens after receipt of a message. The scheduling algorithm of the ROOM virtual machine does not seem to be defined such that models can be executed using simulation protocols – the multi-task schedule is dependent on the underlying real-time operating system.

From the simulator point of view, UML-RT doesn't provide authoritative simulation capabilities – it simply executes a model's logical specification. Since the framework doesn't offer simulation algorithms, primitive customized timing services must be defined to model desirable, correct temporal behavior which in turns undermines having a well-defined relationship between model specification and the simulation algorithm.

## 5.5 DEVS model simulation

The DEVS formalism separates models from how they may be executed. It has well-defined concepts and protocols for executing (simulating) models. Similar to atomic and coupled models, there exist corresponding atomic and coupled simulators. Each atomic model has its own simulator and each coupled model has its own coordinator. The coordinator and simulator hold the last event time *tL* and the next event time *tN* of the corresponding model, respectively. The coordination among atomic and coupled simulator is managed by a root coordinator which ensures all messages are exchanged based on timing atomic models which must either process incoming messages or undergo internal state changes.

One of the advantages of separation between models and simulators is modularity and therefore the ability to change the underlying simulation engine without requiring changes to the models. Furthermore, simulators can be independently developed and verified which increases re-usability, formal analysis, and model validation. The

simulation engine provides a global simulation time to synchronize all modes. For a given set of model components, initial states and input trajectories, a simulation will generate output trajectories using a logical clock, which is independent of the operating system.

RT-DEVS is an extension of DEVS specifically for real-time system in real-time environment (pertinent details of RT-DEVS can be found in [9]). It allows a DEVS model to be developed in a simulation environment and executed in real time rather than simulation time. Therefore, instead of virtual time, in RT-DEVS, the simulator checks a specified time advance of a RT-DEVS model against a real-time (physical) clock.

## 6 Discussion

Table compares the methodologies of DEVS and UML-RT. The table focuses on modeling aspects as viewed in terms of their suitability for developing simulation models as well as software models. The capabilities as well as shortcomings of these have been described in the previous sections. In particular, the overarching examination of the table suggests the importance of using each framework for the fundamental capabilities it supports instead of introducing extensions to it which may weaken and/or otherwise result in unintended complexities affecting its use. That is, software and simulation are inherently distinct – software models are used to develop physical structural and behavioral specification of a system whereas simulation models are used to simulate alternative desired designs of the system. Therefore, it seems important for neither of these frameworks to subsume the other given the separation of concerns between the simulated and physical implementation of a system withstanding the fact that these have important relationships with one another. Computational aspect (e.g., performance and distributed execution) of these frameworks is also a key factor in resisting the desire to extend one framework to subsume the other.

**Table 1: A comparison of DEVS and UML-RT Frameworks**

| | DEVS | UML-RT |
|---|---|---|
| Modeling framework | It is based on systems theory and provides multiple levels of specification abstractions. | It is based on object-orientation; provides different kinds of model abstractions. |
| Structure specification | Atomic model consists of I/O ports, coupled model consist of I/O ports, couplings, and atomic & coupled models. Strict hierarchical composition. | Capsule consists of ports, container capsule consists of sub-capsules and connectors specify the capsule interaction. Hierarchical composition, Generalization/Specialization, dependency, and association are supported. |
| Coupling constraints and communication | All ports are uni-directional. Input ports can only receive information and output ports can only send information. Uni-directional coupling is supported based on modularity and hierarchy principles. | Uni- or bi-directional of a port depends on its implemented protocol. Connections between any two ports must be compatible based on its protocols. |
| Dynamic structure support | An extension of DEVS support dynamic structure modeling (run-time structural changes). | Capsule can dynamically create sub-capsules during run-time. |
| Behavior specification | Internal/external transitions, output and time advanced functions specify atomic model's behavior; coupling defines behavior for hierarchical coupled models. | Statechart is used to specify behavior of capsules and/or containers. |
| Stimuli events and specification | Events refer to messages arriving at input ports or state changes in atomic models. Messages are specified as part of atomic and coupled models. | Stimuli event is specified by a port and signal pair; events can also be due to capsule's state change. Messages communicated between ports are specified in a protocol. |
| Timing and transition | Transition may take zero to infinity. Provides timing for atomic models as well as a global time; one or more of simulation, physical or wallclock time may be used. | Transition time is negligible, but must be greater than zero. Timing service is derived from an operating system clock and global time is not supported. |
| Simulation and synchronization | Each atomic model has a simulator; each coupled model has a coordinator; a root coordinator is used for global coordination among all models. | Capsule and containers can be converted into an implementation by combining them with an execution framework; event scheduling is used for coordination. |

## 7 Conclusions

Real-time software-intensive system development needs consideration of both software and hardware parts. Successful development of such complex systems relies on models for both high-level system simulation that can establish a basis toward low-level software design and development. System theoretic frameworks such as DEVS provide a formal environment for system modeling and simulation. Software engineering frameworks such as UML-RT enable software analysis and design suitable for real-world operation. System theoretical approach is more capable for system analysis and verification while software captures details that are necessary for implementation of physical systems. Comparison of these two frameworks, suggests that their combination has the potential to lead to an overarching framework that is more than the sum of its parts with respect to real-time software intensive system design and software development. To achieve this, it is key to support transformation of simulation model to their software model counterparts. However, examination of the simulation and software models suggests mappings of simulation behavior specifications to software behavior specification to be non-trivial. Our ongoing and future work, therefore, is focusing on developing schemes to support transformation between simulation modeling and software models. A key aspect of such schemes is handling of timing and concurrency issues in real-time distributed settings.

**Acknowledgement:** The authors thank the reviewers for their comments and suggestions which improved the presentation and quality of the paper. This research is supported by NSF Grant No. DMI-0122227.

## 8 References

[1] Manna, Z. and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992.

[2] UML, http://www.uml.org/, 2004.

[3] Wymore, A.W., *Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*, CRC,1993.

[4] Zeigler, B.P., H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation*, 2nd ed, Academic Press, 2000.

[5] Mesarovic, M.D. and Y. Takahara, *Abstract Systems Theory*, Springer Verlag, 1989.

[6] DEVSJAVA, http://www.acims.arizona.edu, 2003.

[7] Cho, Y.K., X. Hu, and B.P. Zeigler, "The RTDEVS/CORBA Environment for Simulation-based design of Distributed Real-time systems", *Simulation: Transactions of the Society for Modeling and Simulation International*, 79(4), 2003, p. 197-210.

[8] Hong, J.S., et al., "A Real-time Discrete Event System Specification Formalism for Seamless Real-time Software Development", *Discrete Event Dynamic Systems: Theory and Applications*, 7, 1996, p. 355-375.

[9] Kim, T.G., S.M. Cho, and W.B. Lee, "DEVS Framework for Systems Development: Unified Specification for Logical Analysis, Performance Evaluation and Implementation", *Discrete Event Modeling and Simulation Technologies*, H.S. Sarjoughian and F.E. Cellier, Editors, Springer, New York, 2001, p. 131-166.

[10] Kofman, E., *Discrete Event Based Simulation and Control of Hybrid Systems*, Faculty of Exact Sciences, Ph.D. Thesis, National University of Rosario, 2003, Argentina.

[11] Sarjoughian, H.S. and B.P. Zeigler, "DEVS and HLA: Complementary Paradigms for Modeling and Simulation?" *Transactions of the Society for Modeling and Simulation International*, 17(4), 2000, p. 187-197.

[12] UML-RT, http://www-306.ibm.com/software/awdtools/developer/technical/, 2004.

[13] Selic, B. and G. Gullekson, *Real-time Object-oriented Modeling*, John Willey & Sons, 1994.

[14] OMG, *UML Profile for Schedulability, Performance, and Time Specification*, 2003.

[15] Shroff, M. and R.B. France, "Towards a Formalization of UML Class Structures in Z", *Proceedings of the Twenty-First Annual International Computer Software and Applications Conference*, 1997, Washington DC, USA.

[16] Moller-Pedersen, B., "SDL Combined with UML", *Telektronikk*, 4, 2000, p. 36-53.

[17] OMG, Model Driven Architecture, http://www.omg.org/mda/, 2004.

[18] Varro, D. and A. Pataricza, "VPM: A Visual, Precise and Multilevel Metamodeling framework for describing mathematical Domains and UML", *Software and System Modeling*, 2(3), 2003, p. 187-210.

[19] Hu, X. and B.P. Zeigler, "Model Continuity to Support Software Development for Distributed Robotic Systems: a Team Formation Example", *Journal of Intelligent & Robotic Systems, Theory & Application*, 2004, p. 71-88.

[20] Raynal, M. and S. Singhal, "Logical Time: Capturing Causality in Distributed Systems", *IEEE Computer*, 29(2), 1996, p. 49-57.

[21] Fujimoto, R.M., *Parallel and Distributed Simulation Systems, Parallel and Distributed Computing*, John Wiley & Sons, 2000.

[22] Praehofer, H., "Towards a System Methodology for Object-Oriented Software Analysis", in *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies*, H.S. Sarjoughian and F.E. Cellier, Editors, Springer, New York, p. 367-388, 2001.

[23] I-Logix, http://www.ilogix.com, 2004.

[24] Hoffmann, H.-P., "From Function/Data-Oriented Systems Engineering to Object-Oriented Software Engineering: The Statemate-to-UML Bridge", http://www.ilogix.com/whitepaper_PDFs/Stm2Rh_whitepaper.pdf, 2004.

[25] Ptolemy II, http://ptolemy.eecs.berkeley.edu/, 2004.

[26] Selic, B. and J. Rumbaugh, "Using UML for Modeling Complex Real-time Systems", Rational Software, 1998.

[27] Gullekson, G., "Designing for Concurrency and Distribution with Rational Rose RealTime", Rational Software White Paper, http://www.rational.com, 2000.