Observations on Real-time Simulation Design and Experimentation

Author 1 **Author 2**

Keywords: ALRT-DEVS, Experimentation, Real-time Modeling, Real-time Simulation

Abstract

Modeling and simulation provides a convenient way for evaluating operability of designed systems (e.g., Network-on-Chip) in their desired target environments. Where applicable, real-time modeling with real-time simulation extends logical-time M&S capabilities by evaluating the degree to which system requirements can be satisfied in actual operational scenarios. Simulations can be directly observed and evaluated in realistic settings. This brings about a well-known challenge – simulation experimentations are never without computational cost. This leads to real-time simulation (adversely) affecting simulation results, sometimes drastically, if not carefully designed for. In this paper, in view of finite computational resources for executing simulations in real-time, new observations on data collection and evaluations are described for the Action-Level Real-Time DEVS simulator. Simple and complex example real-time models are developed in select simulators, experiments with varying degrees of data collection volume are conducted, and results are discussed.

1. INTRODUCTION

With the growing size and complexity of systems, more practical design and analysis methods are required. Logical-time modeling and simulation offers both design (via modeling) and analysis (simulation of models) for predicting system behavior before implementation. Simulation modeling cannot provide complete assurance for system performance, it is more capable than analytical approaches for systems with large state spaces.

Logical-time simulation modeling does not prove very effective when the system to be simulated is part of a larger system with complex and often partially understood dependencies. The resulting model can be very large and inaccurate to be effectively simulated. To handle this difficulty, the system can be partitioned to two parts. One part is simulated in real-time. The other part is the actual (physical) system. For this reason, real-time simulation is an attractive alternative to logical-time simulation as it can afford its use in actual environments. For example, designing a Network-on-Chip (NoC) system needs to account for constrained resources. Rather than modeling an environment (e.g., sensors and actuators) which together with an NoC form a mobile device, it is advantageous to consider real-time simulation of an NoC alone. The mobile device has two parts: one is NoC and the other is the collection of the sensors and actuators. This allows modeling and simulating NoC in 'isolation' and thus can significantly reduce scale and complexity of simulating NoC with its environment as a whole.

The host computer for the simulator can also be considered its 'environment'. Hardware, OS, and other applications that run alongside the simulator can perturb the simulation in a myriad of unpredictable ways. The hardware and network related processing have no effect on the logical-time simulated behaviors (the model dynamics is independent of the computer executing the simulator due to its use of abstract, idealized clock). In other words, logical-time simulation requires the execution environment (i.e., host computer) to have no side effect on the simulated model behavior. The same cannot be said about real-time simulation.

Real-time simulation poses several limitations. It has to complete its execution cycles within real-time deadlines in order to remain synchronized with physical time (and thus the environment in which it is being used). Simulation's real-time clock, as compared with logical time clock, is not absolute. Physical clock is imperfect and thus real-time clock is not the same as abstract logical clock. Lack of perfect physical clock in real-time simulation fundamentally changes the nature of simulation. A prescribed time period defined for an operation to be executed multiple times during simulation at different time instances can change. Real-time simulation cannot guarantee perfect behavior due to inability to equate logical- and real-time clocks.

As real-time simulation execution is subject to time constraints, operations (except model execution) which consume physical time should be reduced. Operations can include data collection and analysis at run-time. Execution of these operations in logical-time simulations, if handled in a modular fashion as in Experimental Frame concept [9], do not introduce any inaccuracy or wrong model behaviors. This is because time can be stopped or elongated. The only one side effect is on the length of physical time it takes to simulate the model. However, physical time is an irreplaceable resource and all operations should be carefully designed and tested for time feasibility before they are used in real-time simulation. For example, if a control signal to the mobile device actuator is transmitted but reaches its destination late (e.g., due to long execution time), the simulation becomes unreliable as compared with the behavior of its real system counterpart.

In the remainder of this paper, related works on real-time M&S, various methods for data collection, analysis, and experimentation are described. The focus is on real-time data

collection, in relation to its logical-time counterpart, using the Action-Level Real-Time DEVS simulator. This approach is compared with select simulators of the same genre. The results demonstrates how ineffective real-time simulators and their support for data collection can be and in particular the benefits of ALRT[1]-DEVS for real-time modeling and real-time execution offers from experimentation perspective given real-time modeling at the level of actions defined for external and internal transition functions.

The structure of this paper is as follows. We present background on real-time modeling, simulation and experimentation in Section 2. Related works are described in Section 3. Challenges for real-time experimentation with emphasis on ALRT-DEVS and Ptolemy II are discussed in Section 4. Examples including data from NoC simulation and their results are detailed in Section 5. Conclusion and further work is presented in Section 6.

2. BACKGROUND

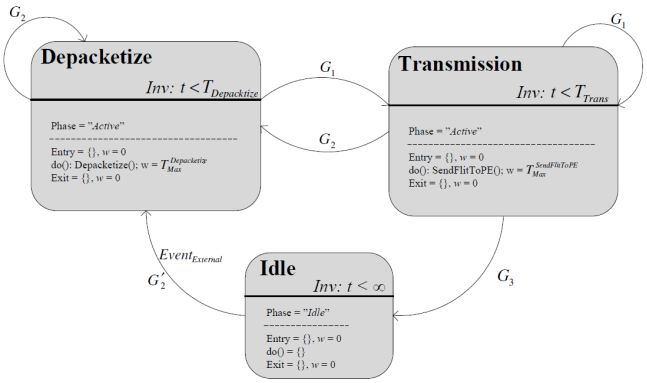
In this section we review some of the basic concepts used in this paper. First of all, the approach for real-time modeling (ALRT-DEVS) and simulation (Real-time DEVS-Suite) is introduced. This will provide an insight for the reader on how real-time modeling and simulation can be developed. In addition, we focus on logical-time experimentation and how it is done in select modeling and simulation tools. Then, we discuss basic shortcomings of logical-time methods for experimentation and propose techniques that can support real-time simulation.

2.1. ALRT-DEVS models

Action-level Real-time model is an atomic Discrete EVent System Specification (DEVS) [8] that can handle concrete real-time modeling of systems. Real-time modeling is aimed at capturing accurate timing information of the target system and reflect it in the model. Furthermore, this model can be later used for real-time simulation. ALRT-DEVS uses *abstract* time interval, actions, and activity mapping function definitions provided in the atomic RT-DEVS = $\langle X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ti, \psi, A \rangle$ real which is an extension of classic atomic DEVS model [8]. An activity mapping function ($\psi : S \rightarrow A$) is introduced for assigning actions (A) to states. Time advance function is defined to have lower and upper bounds ($ti : A \rightarrow \mathbb{R}^{+}_{0,\infty} \times \mathbb{R}^{+}_{0,\infty}$).

In ALRT-DEVS we introduced *concrete* syntax and operational semantics at the level of actions for external and internal transition functions. This is because abstract functions for time windows and actions are insufficient for real-time modeling with direct support for real-time simulation.

[1] It is pronounced alert.



$G_1 = [outgoingDataBuffer \neq \varnothing]$
$G_2 = [incomingFlitQueue \neq \varnothing]$
$G_2' = Event_{External}[G_2]$
$G_3 = [incomingFlitQueue = \varnothing \wedge outgoingDataBuffer = \varnothing]$

Figure 1. Real-time Statechart for depacketizer component

Specifically, the 'state' of a model has to be rich enough to lend itself for specifying actions individually and collectively for both external and internal transition functions. Dynamic decision making for actions given limited time periods is specified using the concepts of locations, transitions, and actions provided in Statecharts [5]. In this approach we assumed a model to have different locations within a state based on remaining time. Also, actions are mapped to each location with a predefined sequence of execution. Finally, guarded transitions between these locations provide a dynamic method of decision making for the model. Guards on transitions are chosen from secondary state variables which can be any variable in the specification other than state and time. This could be the length of the incoming queue in server, the height of water level in a water tank, or the temperature in a nuclear power plant.

Figure 1 depicts a simple statechart model for depacketizer component. Three phases are assumed for depacketizer: *Depacketize* in which flits are converted to streams of data, *Transmission* in which depacketized data is sent to the processing element, and *Idle* for other stages of this component in which the input/output queues are empty. If any of the guards specified in the diagram are satisfied, the system changes its phase to the one conforming to the current evaluation of the secondary state variables.

2.2. Real-time DEVS-Suite

Real-time models can be potentially used for real-time execution. Time in a real-time simulation engine is in synchrony with real-time. A model simulated in a real-time simulator can be connected to its operating environment and communi-

cate via sensors and actuators in order to react to stimulations from outside and provide performance data.

Real-time DEVS-Suite provides real-time simulation by guaranteeing the execution of each action within its specified time window. Each action which violates its firm deadline would be discarded and the model may modify the line of execution using its dynamic decision making system. In addition to executing action within their respective time window, Real-time DEVS-Suite incorporates Java's threading capability to introduce parallel execution to atomic model execution. Before this, models were executed sequentially and this was not useful for real-time execution. Aside from these, basic features of ALRT-DEVS such as locations, transitions, actions, and time windows are also added to the DEVS-Suite [1] to support real-time modeling in addition to real-time simulation.

2.3. Logical-time Experimentation

In logical time simulation, experimentation is usually handled by recording interesting events in a file or a transducer in an experimental frame (EF). The experimental frame is responsible for generating events, injecting them into the simulation model, and recording the simulation outcomes using transducers. DEVS modeling insists on modularity. Therefore, data collection is to be carried out using ports from atomic and coupled models. Hierarchical modeling enforces each level to communicate with its immediate higher/lower level models. This also applies to outside simulation interaction in which communication should occur through an interface and not individually by each atomic model.

In the DEVS preferred data collection and analysis scheme, EF is a coupled model consisting of a transducer and a generator. These atomic models are executed along with other atomic/coupled models of the system by the central simulator engine. To record events, one should connect every port which transmits interesting events to the experimental frame (transducer in specific). The transducer organizes and records the events for analysis. Other simulation engines use various methods for data collection. As for Ptolemy, data analysis is done in specific actors (such as plotters) which is similar to the concept of EF. In order to show the similarity in the function of DEVS-Suite and Ptolemy, a simple ramp example is incorporated and executed in logical-time. The model contains a *CurrentTime* actor which outputs the current time of the whole model, a *Ramp* actor, and a *X-Y Plotter* to sketch the outcome. This model works under a *Continuous* director. In DEVS-Suite the model consists of a generator which triggers the ramp model to generate data. The output of both are lines with steady slope which was expected. The same test is done in real-time in Section 4. to show the difference in real-time simulation. Various methods of data collection and analysis are used in different simulators, however, many of them would be problematic when used for real-time simulation. This is discussed in Section 4. as well.

3. RELATED WORK

Ptolemy II [2] is an integrated Java-based modeling and simulation environment designed at Berkeley. This tool is capable of modeling heterogeneous systems and simulate them under various models of computation [6]. Real-time execution is supported in this tool which is compared with that of DEVS-Suite. eCD++ [7] is a DEVS-based simulator for embedded system simulation. It provides graphical modeling (similar to Ptolemy II) and supports best effort real-time execution. Also, real-time testing in MathWorks by xPC Target [3] is designed to enable HIL (Hardware-in-the-Loop) and RCP (Rapid Control Prototyping) on general purpose computers or on xPC Target Turnkey (real-time target machine for xPC Target). For the purpose of this paper, ALRT-DEVS and Ptolemy II are considered.

4. REAL-TIME EXPERIMENTATION

ALRT-DEVS supports defining parallel atomic models that use default confluent function. Every action is defined to require positive, non-zero duration to be executed in physical time. Output events are to be generated prior to handling input events. The role of real-time data collection, however, is not accounted for in ALRT-DEVS. Coupled parallel models for this kind of parallel atomic model can be supported.

Simulations were carried out without adhering completely to the Experimental Frame (EF) concept where outputs for all atomic models are collected directly from them. We noted the impact of data collection on real-time execution time, but we did not formulate the role of real-time for action-level operations given physical time needed to collect (observe) simulation data (output events). Transient states (requiring zero-time actions) which are needed for confluent function is not defined for ALRT-DEVS. Calling an output function prior to external transition causes the model to undergo transient operations. All simulation operations including generating outputs take time to execute and can cause accuracy degradation if execution of actions cannot be guaranteed to complete fast enough (within each simulation cycle addition time is reserved for data collection). That is, use of transient states has to be accounted for when simulation data collection and analysis is to occur in real-time. Furthermore, if a piece of data is ready but not captured at the time it was created because of time constraints, then the result of the experiment is inaccurate from timing perspective.

We can divide physical time it takes for a complete atomic model simulation cycle to be executed to two parts. One part is for model execution (simulation, Δt_{sim}). The other part is for data collection (observation, Δt_{obs}). The model execution part can be further divided into two parts. One is the time

duration specified for actions in internal and external transition functions ($\Delta t_{ext,int} > 0$). The other part is the time for executing output, time advance, and all other tasks such as state-based data aggregation and disaggregation. The time allocated for this part ($\Delta t_{rest} > 0$) is needed for having a complete atomic model simulation cycle. The total time for an atomic cycle $\Delta t_{ac} > 0$ is equal to $\Delta t_{ext,int} + \Delta t_{rest} + \Delta t_{obs}$. Note that $0 < \Delta t_{obs} < \infty$ accounts for processing time that is needed for collecting data (e.g., writing data to a file or to a display) which is distinct from processing time needed for generating output events.

Aside from atomic model, real-time execution processing time for coupled models also has to be accounted for. This is because logical-time coupled models assume instantaneous communication (i.e., outputs generated from an atomic model can be delivered as input to another atomic model without simulator consuming time). In real-time simulation of coupled models, it takes some non-zero, positive time period ($\Delta t_{I/O} > 0$) for the coupled model to route and deliver output messages as input messages between any two (atomic or coupled) DEVS models. The $\Delta t_{I/O} > 0$ accounts for messages that are communicated across external input, internal, and external output couplings. Therefore, the flat coupled cycle time for a coupled model with one atomic model can be defined as $\Delta t_{fcc} = \Delta t_{ac} + \Delta t_{I/O}$ where time allocated for communication is for communications that may occur across external input and external output couplings. More generally, for a flat coupled model with n atomic models, time for a flat coupled cycle is defined as $\Delta t_{fcc} = \sum_{i=1}^{n} \Delta t_{ac,i} + \Delta t_{I/O}$. For a hierarchical coupled model with n atomic models, p flat coupled models, and q hierarchial coupled models, its execution time for one complete cycle can be defined as $\Delta t_{hcc} = \sum_{i=1}^{n} \Delta t_{ac,i} + \sum_{j=1}^{p} \Delta t_{fcc,j} + \sum_{k=1}^{q} \Delta t_{hcc,k} + \Delta t_{I/O}$.

The above formulation affords defining lower bound on physical time that is needed for real-time data observation in real-time simulation given some finite computing resources. The lower bound can be defined given Δt_{obs} for the atomic and coupled models that are to be observed. The observation time can be controlled as a function of the number ports that are observed. It should be noted that observations for events (data or messages) in coupled models should be avoided because they can only contain data from their respective atomic models. Data that is observed on output ports of any coupled model are intrinsically the same as those generated by atomic models, but delayed in time.

4.1. Challenges

In logical-time simulation, execution of models and the communication among them are all handled sequentially. Therefore, data collection is not an issue. Naturally, data collection requires time. Usually an experimental frame is involved in the process of data collection which gathers all interesting events within the model via ports. Handling all these events in each instance of time can be time consuming. However, since time advance is controlled by the simulation engine, it could be shortened or elongated whenever required. However, this is a major issue in real-time simulation where time cannot be stopped or slowed down. This poses serious limitations on the number of probes an experimental frame can observe, the frequency of occurrence of that event, and the amount of processing that can be done on the gathered data. An experimental frame is itself one model among the other simulation model which consumes memory and computation time. For every single event, the EF model goes through an external transition cycle and analysis. Therefore, the frequency of the EF event processing is the sum of the output generation frequency of all components connected to it! This can be an important source of time loss for the rest of the model. In addition, processing of events can be also problematic if not taken into account. In logical time experimentation, the analysis can be done in the time of the simulation and it does not interfere with the validity of the outcome. However, for real-time simulation, the amount of processing per event can make a substantial difference in the result of the simulation and deadline losses. In cases which the continuation of the simulation is dependent on the analysis of gathered data, the designer must deal with the trade-off of accuracy and validity: if analysis takes so much of the computation time, the results generated by the simulation will be inaccurate because of the frequent deadline losses (contradicts with accuracy) and on the other hand if analysis is done partially or with lower precision, the simulation can go the wrong way since the controller cannot make the right decision based on the feedback from the environment (contradicts with validity).

We faced the differences between experimentations in real and logical-time simulators when working with an NoC model. In Section 5., an experiment on NoC data collection demonstrates this difference. For example, NoC data collection can be done with the usual transducer component. The implications of using this method of data collection was discussed above. One other way is to use the flit itself as the data collector. In this approach, the flit (which is a non-simulatable object) collects timing data while moving in the network from one node to the other. This can include very detailed information such as operations done inside a switch (moving from incoming queue to crossbar or from there to the output buffer) which are not sensed outside the switch. The flit is then archived when it sinked at its destination and used for analysis after the simulation is over. Data can also be collected in each component and reported periodically to a transducer component. This is more efficient that event-by-event reporting and the data is centralized in case they were needed for making central decisions. Of course the method of data collection is tied with the type of the system under simula-

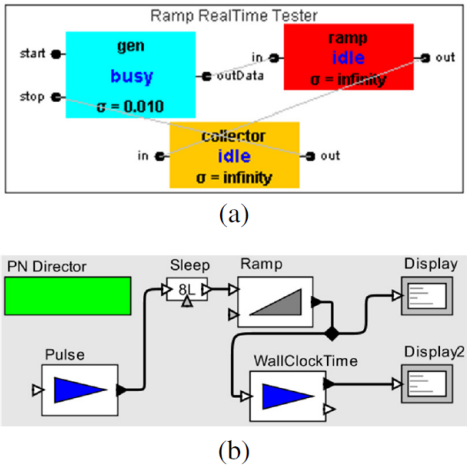


Figure 2. Ramp model: (a) ALRT-DEVS and (b) Ptolemy II

tion. For example, in some of them a moving object (such as flit in NoC) does not exist. System specific characteristics, may eliminate some methods of data collection and analysis. These are discussed in Section 4.2.

In order to see the effect of data observation on the pure model behavior, one does not need to consider complicated models. Here, we used a simple ramp model to show the seriousness of the situation. This quick experiment is done with ALRT-DEVS and Ptolemy II on a Windows 7 64-bit machine with 2 GB memory and Intel dual core, 1.83 GHz processor. The real-time ramp models designed in DEVS-Suite and Ptolemy II are depicted in Figure 2. In order to build the real-time ramp in Ptolemy, we used a *Pulse* to start the process and a *Sleep* component (from Ptolemy's real-time package) to trigger the *Ramp* component every 8 ms. The output is retrieved using a *Display*. Also, the physical time is captured by the *WallClockTime* component and the director we used was *PN* (Synchronous Data Flow). The ALRT-DEVS model is consist of three components: a *ramp* model, a *generator* for triggering the ramp, and a *collector* component which records the outputs and their respective timestamps.

What we expect from the output is a clean ramp line. In Figure 3-a, the output of Ptolemy along with ALRT-DEVS and the real-time deadline of the ramp are depicted to show the difference between them. The real-time deadline is the expected time which we expect the output to be generated from the ramp. In the top left of the plot, the first 10 steps are magnified which clearly show the difference between the outputs retrieved from Ptolemy II and ALRT-DEVS. This difference is caused by a slow initialization phase in Ptolemy II which causes the whole plot to have a displacement when compared to the real-time deadline and ALRT-DEVS outputs. Therefore, the output of Ptolemy is not conforming to the deadline and since this is real-time, we cannot expect similar results from two different executions. Other runs of the same experiment resulted in quite different plots.

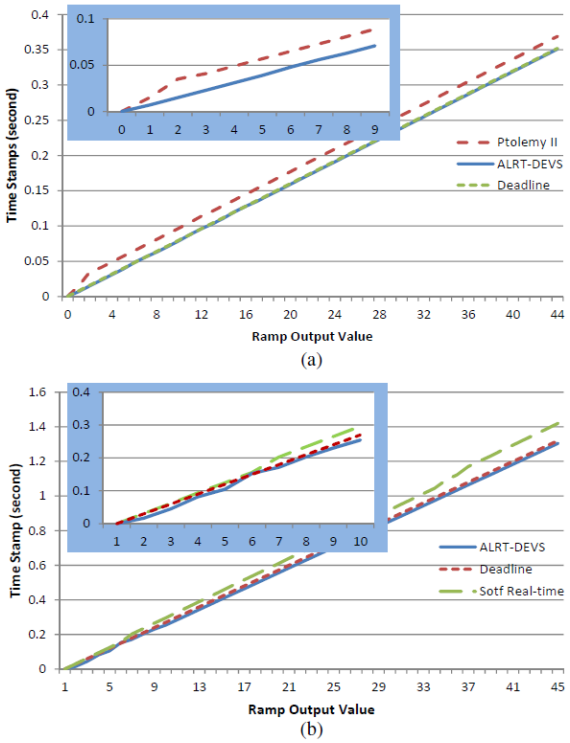


Figure 3. Ramp model behaviors in (a) ALRT-DEVS vs. Ptolemy II and (b) Soft real-time DEVS vs. ALRT-DEVS against real-time deadline

This simple example shows how unreliable the results of a sophisticated system would be if real-time experimentation problems are not addressed. In this paper, we are trying to emphasize on the importance of this issue and suggesting several guidelines for alleviating this problem. The difference in curves resulted from Ptolemy and Real-time DEVS-Suite has its origins in different modeling formulations and simulation protocols. In our ALRT-DEVS modeling approach, as introduced in Section 2., we increased operation granularity of a component to actions and mapped time windows to each of them. This way, time constraints are enforced in each model and for every action. However, including timing information in the model is one thing and using them in the execution platform is another. This is why, Real-time DEVS-Suite adopts a novel real-time execution protocol which is different from the logical-time DEVS-Suite simulation protocol. This protocol insures executing models based on the timing constraints specified in the models. As for Ptolemy, execution in real-time is not guaranteed in the simulation protocol. Also, real-time actors (such as execution time or real-time Plotter) do not impose time constraints, therefore, real-time execution happens at the level of best-effort instead of absolute guarantee.

In order to show the impact of using Real-time modeling and simulation technique (such as the one we developed in ALRT-DEVS and Real-time DEVS-Suite) we compared the output of ALRT-DEVS with the output of soft real-time DEVS-Suite for the ramp model. However, this time the emphasis is on time. Soft real-time DEVS provides best effort (soft) real-time execution. Here we intend to show how much our hard real-time extension affects the timeliness of the output. The hardware configuration of this experiment is the same as above. Figure 3-b contains three lines. The green line is the output of the soft real-time DEVS-Suite and the blue line is the output of the ALRT-DEVS ramp model. The red line shows the deadline of the output as if this was a real system. The result clearly demonstrates the perfect timeliness (the red line is on the blue line) which Real-time DEVS-Suite provides when compared to the soft real-time output of logical-time DEVS-Suite. Also, it is important to notice that the soft real-time output is diverging from deadline over time based on a monotonically increasing function which results in a two second divergence after 30 seconds into the simulation.

4.2. Addressing RT-Exp. Issues

In general, there are three requirements in order to reach reliable real-time experimentation: 1) real-time modeling, 2) real-time simulation protocol, and 3) real-time data collection. Each of these requirements are further illustrated below.

4.2.1. Real-time Modeling

The aim of real-time modeling (which was partially discussed above) is adding time information to the model of the system. These timing information specify upper-bounds and lower-bounds for actions to finish their execution inside them. The resulting model is capable of being executed in real-time since all time constraints are stated in the model. We encourage the reader to refer to [4].

4.2.2. Real-time Simulation Protocol

The simulation protocol, should take advantage of the real-time model and apply those constraints in the execution of the model. As described in the case of Real-time DEVS-Suite, the simulation protocol must guarantee the execution of each action within its time window. The simulation engine must cuts off the execution of one action which is going beyond its specified time window in order to prevent time inconsistency to escalate in the system and affect the results which are to follow.

4.2.3. Real-time Data Collection & Analysis

Real-time M&S provides us with an executable model of the system which operates in real-time and is capable of communicating with the environment. However, experimentation is the primary reason of putting this amount of effort in modeling and simulation. The issue however is the impact of data observation and analysis on the simulation which we call *observation impact phenomenon*. Based on this, observation always impact the phenomenon which is under observation. However, this does not hold for logical-time simulation since it is isolated from the real world (even from the point of view of time advance). On the other hand, real-time simulation is part of the real world and is partially under the influence of this phenomenon. The problem is that every simulator related operation takes its resources from the simulation model. This results in deadline miss and time inconsistency if not addressed properly. Therefore, data collection should be done carefully or it might cause the simulation to diverge from the physical/virtual system it represents. Below are four points that are important in implementing methods of data collection and analysis.

The system which is represented by the model should always be considered when implementing data collection and analysis methods. For logical-time simulation, a general data collection method (such as EF) is always sufficient. However, in real-time simulation of a system one method may be more efficient than the others. In NoC simulation, the number of components (flit-level modeling) may exceed several hundred for a chip with 32 cores. Handling of all events emitted from these components by one transducer is very time consuming. However, flit-based data collection (discussed above) may alleviate this situation. The benefit of this method is that no extra event for the purpose of data collection is needed. To conclude, by designing a domain *specific data collection method*, higher level of efficiency might be reached which is always welcomed for real-time simulation.

Online data analysis means when analysis on data is done simultaneously with the simulation instead of postponing it to the end. Online analysis is useful when the result of the analysis at every instance is used as an input to the simulation (feedback), the user needs intermediate results to be shown, or the analysis determines when the simulation should end. If online analysis is needed for any of these reasons, data collection method is also affected because centralized data is needed. If data is gathered in distributed fashion (like flit-based data collection), a reporting mechanism must periodically submit the collected data to a centralized transducer for online analysis.

Analysis data loss can happen in real-time simulation. We use this term when a performance data which is to be collected for analysis is not collected because there is no time to complete that task (due to firm deadlines). We should look at data collection as part of the real-time simulation which may sometimes miss deadlines and lose data. So, based on the amount of resources dedicated to the simulation, there is a tradeoff between the accuracy of the simulation and the ac-

curacy of data analysis.

The amount of resources given to the simulation engine is effective on the result. In other words, there is a direct relationship between the amount of resource and the accuracy of results. Therefore, one should always consider this factor when analyzing the performance data resulted from the simulation. Logical-time simulation is executed on infinite resources (time is also labeled as a resource). Therefore, accuracy of results and granularity of the model are the only factors to consider when analyzing data from a logical-time simulation. As for real-time simulation, the impact of resources on the final results brings in the amount of resources into consideration as well. In order to illustrate this point, two of the experiments above are done using an Intel 2.93 GHz Dual Core processor with 4 GB of memory. We repeated the experiment done in Figure 3-b on the new configuration. Results showed that the hard real-time still strictly conforms to the deadlines but the soft real-time execution experienced only 1 (instead of two) second divergence after 30 seconds.

5. EXPERIMENTS AND RESULTS

One of the ways of reducing the impact of data collection on the results is periodic reporting instead of event-by-event reporting. We call it: *Periodic Data Collection*. In this approach, the data is gathered on periodic cycles which reduces the number of events to be processed by the transducer. This reduction means less time to be spent on data collection and analysis. In order to illustrate this, three experiments are designed: two of them use the ramp model and the other uses NoC which is relatively more complex. All experiments are carried out on a computer with a Windows 7 32-bit machine, Intel 2.93 GHz Dual Core processor with 4 GB memory.

5.1. Experiment I

In this experiment, the ramp model (used above) is changed in a way to report its output value in a periodic fashion. We have used four different values: 1, 5, 10, and 50 which represent the number of cycles the ramp waits to produce an output. In $Period = 1$, every step is reported to the data collector but in $Period = 50$, the report is made at every 50 cycles. The ramp model in this experiment produces an output every 0.005 seconds. Figure 4 shows the difference between the results extracted from different reporting periods. For $Period = 1$, data is approximately gathered 10.5 *ms* before the deadline. Occasionally, this values changes due to the computation load on the system. Comparing these results with other periods of data collection results in two conclusions: 1) data collection takes less time in longer periods and 2) the number of occasional variations are more when data is collected more frequently which reduces simulation predictability.

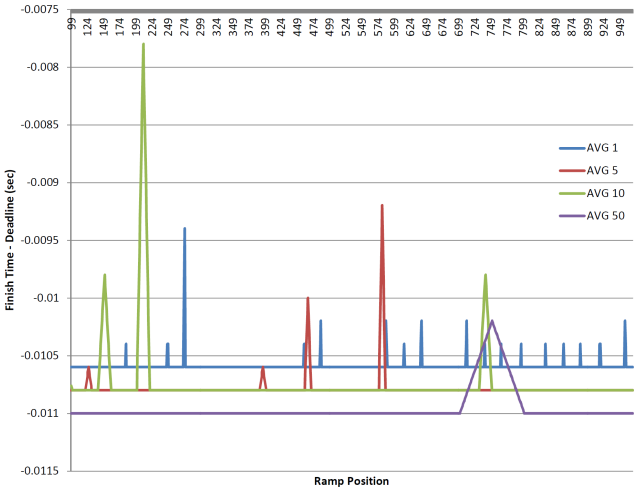


Figure 4. Periodic data collection effect for ALRT-DEVS ramp model

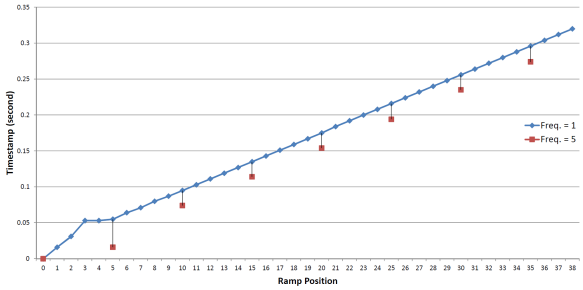


Figure 5. Impact of data collection frequency on Ptolemy II

This simple experiment shows that the effect of data collection is more than generally assumed. The ramp model is relatively simple but still shows drastic changes when the period is changed. This effect can be substantial for more complex systems (as done in experiment III) . Also, there is a limit to the periodic data collection effect. After a certain point, the impact on the results becomes trivial ($\Delta t_{obs} \rightarrow 0$). The rest of the difference with the actual model relates to the model formulation and simulation protocol (represented by Δt_{hcc}, Δt_{rest}, and $\Delta t_{ext,int}$).

5.2. Experiment II

We changed the Ptolemy II model in order to see the periodic data collection effect. For this purpose, a Modal Model was used and a simple five-state filter which passes one piece of data out of every five. When tested the model on 125 (steps per second) data generation rate and the results were as depicted in Figure 5 with a 5-cycle data filtering (individual dots marked with $Freq = 5$). Improvements are evident when this is compared with Ptolemy's ramp output with no data filtering in the blue line.

5.3. Experiment III

This experiments tests periodic data collection in NoC. The NoC in this experiment is a 3×3 network with mesh topology. The parameters watched in this experiment are average flit latency from source to destination and average waiting time at every switch. Here, in order to demonstrate the impact of data collection on final results, we reduced the number of components being observed. In one scenario, the transducer is connected to all 9 processing elements and in another it is connected to (two nodes that are farthest apart). Furthermore, the effectiveness of the injection rate on this phenomenon is analyzed by increasing it from 2.22 (flits/sec) to 4 (flits/sec). The results shown in Table 1 are average points of 5 runs for each configuration. For small injection rates (no network congestion exists and flits are delivered on time), the difference between results are trivial. However, as we move toward higher injection rates (such as 4 flits per second) the difference becomes more substantial.

For this experiment, it is important to notice that the difference in the number of elements under observation is only 7 but the difference observed in the results cannot be neglected. This experiment is a clear demonstration of how –even for this small differences– data observation can significantly affect the output. This is more evident when the system becomes more complex (such as NoC as compared with Ramp) or the simulation is at its *Saturation point*. At the saturation point, the hardware reaches its limits in executing simulation tasks and a sudden increase in deadline losses and time inconsistency is observed. This case is seen when the injection rate is increased in the NoC experiment.

Table 1. 2-node vs. 9-node observation effect on avg. flit latency (Avg. FL) and avg. switch waiting time (Avg. SW) with various injection rates

		Avg. Injection Rate (per sec)			
		2.22	2.86	3.03	4.00
Avg. FL	9	2.95	3.72	3.97	18.98
	2	2.92	3.52	3.86	17.73
Avg. SW	9	0.74	0.93	0.99	4.75
	2	0.73	0.88	0.96	4.43

6. CONCLUSION AND FUTURE WORK

In this paper, we demonstrated how data collection and experimenting can affect results in a real-time simulation. Also, we used DEVS-Suite and Ptolemy II as examples to show the *Observation Impact Phenomenon*. Finally, we suggested several methods for limiting the impact of data collection on simulation execution and demonstrated the difference resulted from these methods using a simple ramp model and a 9-node Network-on-Chip model. It is important to notice that unlike logical-time simulation, in real-time simulation, the impact of data collection and analysis (i.e., experimentation) should be accounted for as illustrated with the ALRT-DEVS NoC simulation. In summary, this paper has illustrated that careful attention is required for designing experiments for real-time simulation models and results must be carefully evaluated in order to be valuable.

REFERENCES

[1] DEVS-Suite simulator, version 2.1.0. http://devs-suitesim.sourceforge.net/, 2009.

[2] J.T. Buck, S. Ha, E.A. Lee, and D.G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Journal of Systems Architecture*, 4:155–182, 1994.

[3] J. Burck, M.J. Zeher, R. Armiger, and J.D. Beaty. Developing the world's most advanced prosthetic arm using model-based design. *The MathWorks News & Notes*, 2009.

[4] S. Gholami and H.S. Sarjoughian. Real-time network-on-chip simulation modeling. In *SIMUTools, Desenzano, Italy*, pages 103–112. ICST, 2012.

[5] H. Giese and S. Burmester. Real-time statechart semantics. *TR-RI-03-239, University of Paderborn*, 2003.

[6] E.A. Lee and S. Neuendorffer. Tutorial: Building ptolemy II models graphically. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2007-129*, 2007.

[7] Y.H. Yu and G. Wainer. eCD++: an engine for executing DEVS models in embedded platforms. In *Proceedings of the 2007 summer computer simulation conference*, pages 323–330. ACM Digital Library, 2007.

[8] B.P. Zeigler, T.G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.

[9] B.P. Zeigler and H.S. Sarjoughian. Introduction to DEVS modeling & simulation with JAVA: Developing component-based simulation models. *Arizona Center for Integrativ Modeling & Simulation*, 2003.