I-DEVS: Imprecise Real-Time and Embedded DEVS Modeling

Mohammad Moallemi, Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University, Centre of Visualization and Simulation (V-Sim)
1125 Colonel By Dr. Ottawa, ON, Canada.
{moallemi, gwainer}@sce.carleton.ca

Keywords: DEVS, Embedded Systems, Real-Time Systems, Model Based Approach

Abstract
The problem of over-running in hard real-time systems poses critical risks to the hardware under control. The imprecise computation technique offers an effective way of resource utilization in these cases. We introduce Imprecise-DEVS (I-DEVS), a model-driven approach to develop real-time and embedded applications based on the DEVS (Discrete Event Systems Specification) formalism. This approach combines the dynamic advantages of the imprecise computation technique with the rigor of a formal modeling methodology. This framework can be used to develop embedded applications incrementally, integrating imprecise models with hardware components seamlessly. We have defined structural modifications to DEVS in order to allow imprecise model definition.

1. INTRODUCTION

Embedded real-time (RT) software construction has usually posed interesting challenges due to the complexity of the tasks executed. Formal methods have showed promising results, in terms of model design, verification, implementation, testing and maintenance. Also, model-based approaches can be used for formal verification [1] as well as virtual-time simulation, which reduces both end cost and risk, while enhancing system capabilities and improving the quality of the final product. This is a useful approach, moreover considering that testing under actual operating conditions may be impractical and in some cases impossible [2].

One critical aspect of hard RT systems is the production of the outputs before the specified deadline. However, in circumstances with system overloads, it might be impossible to meet the deadlines. As RT and embedded systems are not deterministic, tasks may enter the system at any time and there is no prior knowledge of their occurrence times. The *Imprecise Computation* technique [3] helps to overcome these high computation peaks by discarding optional computations. The main idea is to separate the computation into mandatory and optional parts (the mandatory part affects the correctness of the result and the optional affects its quality). The optional part is executed after the mandatory part; acceptable results are guaranteed, and if resources are available, the execution of the optional part increases the precision of the result. The system can terminate the optional part during transient overloading, producing less precise results but on time.

Imprecise computation has been applied to different fields, including RT and embedded systems [4][5][6], multimedia processing [7][8][9], planning, artificial intelligence [10][11][12] and databases [13][14]. Despite this, imprecise computation is not yet widely used in industrial embedded applications. The reason might be related to strict theoretical assumptions and the lack of cost-effective support method that can be easily implemented in embedded systems.

In this work, we introduce Imprecise-DEVS (I-DEVS), a model-driven framework to develop RT and embedded systems based on DEVS formalism [15], integrated with the imprecise computation technique to improve predictability under transient overloading. The approach supports rapid prototyping, and encourages reuse. DEVS has been extended for RT simulation and also for embedded applications [16][17] [18][19][20][21]. Many existing techniques that have been widely used for the development of RT and embedded systems are also mapped to DEVS [22][23][24][25][26]. I-DEVS enables model designers to assign priorities to the model behavior and balance the execution based on the priorities assigned. The new approach can be easily integrated with previous models. The main goal of this contribution is to develop a dynamic RT DEVS environment capable of managing different high processing conditions and integrate that with the RT DEVS engine.

2. BACKGROUND AND MOTIVATION

Imprecise computation has been used for minimizing the errors caused for transient overloads in RT systems. Many off-line task scheduling algorithms have been proposed based on the imprecise computation technique [22][28][29]. There is no optimal algorithm that minimizes the total error in on-line scheduling systems, when a feasible schedule exists, because of the lack of a-priori knowledge of the occurrence time of the jobs [30]. The *mandatory-first* algorithm assigns processing time to mandatory tasks first, based on statistics to reduce the total error [31][32].

RT-Frontier [4] is a RT operating system that presents an imprecise computation framework. It decomposes computations into mandatory, optional and wind up parts. The wind up part works as a termination function after the end of an optional part, reducing the termination cost and increasing portability. It uses a novel scheduling algorithm called Slack Stealer for Optional Parts (SS-OP), based on the three segment imprecise computation model with small overhead.

Except for this work, no research has aimed on integrating this technique with a formal methodology to be used in RT and embedded system design and construction. The proposed I-DEVS approach, allows the model designer to deploy this technique at the design time, specifying the optional and mandatory behaviors of the target system.

DEVS [15] is a sound formal framework based on generic dynamic systems, including well-defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse. A real system modeled with DEVS is described as a composite of sub-models, each of them being behavioral (atomic) or structural (coupled), that define a hierarchy of models. Coupled models are responsible for maintaining the structure of the hierarchy by keeping the internal connections between atomic and coupled models. A DEVS atomic model is formally defined by:

$$AM = < X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ta >$$

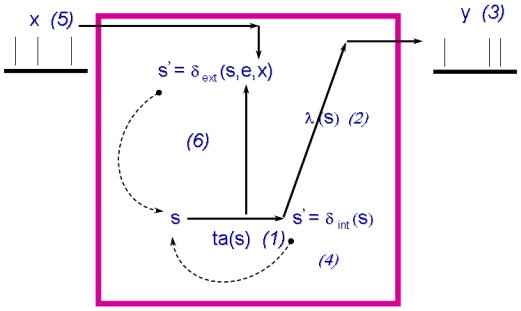

Figure 1. DEVS atomic model state transition [2]

As informally depicted in Figure 1, an atomic model AM is an entity which is affected by external input event X and which in turn generates output event Y. The state set S represents the set of state variables of the model. The internal transition function δ_{int} and the external transition function δ_{ext} compute the next state of the model. When an external event arrives at elapsed time e which is less than or equal to ta(s), a new state s′ is computed by the external transition function δ_{ext}. Then, a new ta(s′) is computed, and the elapsed time e is set to zero. Otherwise, a new state s′ is computed by the internal transition function δ_{int}. In this case, an output specified by the output function λ is produced based on the state *s*. As before, a new ta(s′) is computed, and the elapsed time e is set to zero.

A coupled model connects the basic models together in order to form a new model. This model can itself be employed as a component in a larger coupled model, thereby allowing the hierarchical construction of complex models. The coupled model is defined as:

CM = <X, Y, D, {M_d | d∈ D}, EIC, EOC, IC, *Select*>

X: is the set of input ports and values,

Y: is the set of output ports and values,

D: is the set of the component names,

EIC (*External input couplings*) connects the input events of the coupled model itself to one or more of the input events of its components,

EOC (*External output couplings*) connects the output events of the components to the output events of the coupled model itself,

IC (*Internal coupling*) connects the output events of the components to the input events of other components,

Select is a function used to order the processing of the simultaneous events for sequential events.

In [19] a RT extension to DEVS formalism was proposed based on the parallel DEVS (P-DEVS) [33] formalism. P-DEVS adds the confluent function *δcon* (and input/output bags instead of sets) to the definition of atomic model, in order to handle simultaneous events. The *Select* function is eliminated. This RT method integrates the *driver* object presented in [34] to the DEVS model hierarchy to allow embedded functionality on different hardware and software platforms. The driver object is a user configurable converter of events from external environment to the DEVS model hierarchy and vice versa, which increases portability by separating modeling components from the external framework, allowing both virtual-time and RT simulation for verification and embedded execution on the target. The proposed methodology has been implemented on the ECD++ [18] toolkit, a RT extension of CD++ [35] [2].

3. IMPRECISE DEVS

Despite the theoretical advances in imprecise computation, there are few practical projects aiming at producing effective RT tools based on this technique. The main goal of this work is to integrate imprecise computation technique with the DEVS modeling approach to construct a formal method capable of modeling RT systems, and to build a toolkit based on this approach. The objective is to provide an imprecise framework for applications where the job arrival times are not known a-priori. The approach tries to balance the computation when the system is busy and on the other hand not reducing its performance, while keeping the runtime overhead of the implementation as low as possible.

Imprecise DEVS (I-DEVS) is built on top of RT DEVS presented in [19]. The P-DEVS atomic model definition is modified by adding a deadline and a mandatory or optional condition for each state, as follows:

$$AM = < X, S, Y, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta, d >$$

Where X, Y, δext, δint, δcon, λ and ta are the same as in RT DEVS [19],

S: $\{(s, c) \mid s \in Z^{+}_{0}$ and $c \in \{mandatory \mid optional\}\}$.

d: $S \rightarrow R^{+}_{0,\infty}$, is the relative deadline of each state for output production.

To prove the closure under coupling for I-DEVS, the same process is followed as P-DEVS [33], the only addition is the d(s) of the resultant I-DEVS coupled model, defined as follows:

$d(s) = minimum\{\sigma_i | i \in D\}$, where $s \in S$ and $\sigma_i = d(s) - e_i$.

3.1. DEVS Task System

The main computations in DEVS occur during state transitions and message transfers. We use this information to map the DEVS functions run by abstract algorithms into a RT tasking system. In this way, we obtain a platform where imprecise computation can be applied. Figure 2 shows the processing tasks in a DEVS-based system.

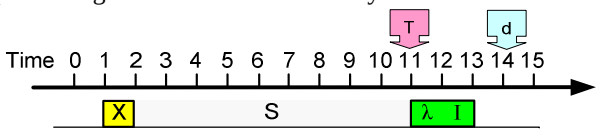

Figure 2. Processing carried for a state transition.

The external transition (*X*) is mapped into a task that initiates the state *S*. The task release time is equal to the arrival of the input to the model (from the external environment in the case of the topmost coupled model, or when the output generated on an output port is received in the atomic model input port). We assume no deadline for the X task. The output (*λ*) and internal transition tasks (*I*) are considered to execute together (task *λI*, as outputs in DEVS are always followed by an internal transition). The release time of task *λI* is equal to the end of the state ***S*** and specified by ta(s) (indicator *T*). Its deadline is specified by the function d(s) (indicator *d*) that we added to the atomic model definition.

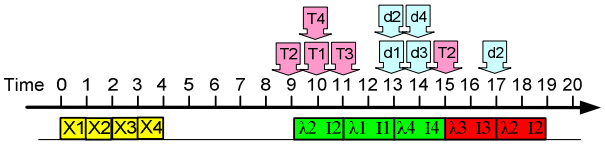

Figure 3. Overload scenario

Figure 3 shows an overload scenario where four inputs are injected, starting external transitions on different atomic models. As we can see, *λ1*, *λ2* and *λ4* meet their deadlines; however, *λ3* and *λ2* (second instance at time 18) do not. The internal transition *I2* produces a new state with ta(s) equal to 4 time units, which exceeds its deadline at time 17.

In order to execute these models, we extended the DEVS atomic model definition by dividing the above-mentioned tasks into mandatory and optional parts, incorporating the imprecise computation concept. We assume that tasks *X* are always mandatory but *λ1* tasks can be optional. The *λ* subtask of an optional *λ1* task can be terminated under transient overloads. For multiple consecutive optional *λ1* tasks, the entire task can be discarded, except for the last one (from which we only discard the *λ* portion), as the last *1* task needs to perform the internal transition. In other words, during overloads, the model skips optional internal transitions and their output functions to save time and resources for the mandatory ones. For instance, an autonomous robot in a bumpy road with obstacles (flooded with obstacle reconnaissance inputs) can discard unnecessary tasks (e.g. reporting). A similar scenario can occur in any RT system where a sequence of optional outputs can be skipped to alleviate the overload situation by keeping the necessary outputs produced on-time. Schedulability analysis can be applied to this model, based on various available methods [38], [39].

3.2. Execution Algorithm

Figure 4 shows the execution algorithm of the Imprecise DEVS. The main algorithm is performed in the Root coordinator (the top coordinator in the DEVS abstract runtime hierarchy), which takes care of the RT discrepancy in the execution of external or internal events.

```
1.  main()
2.  Send I msg to all atomic models
3.  tN = closest internal or eventfile event
4.  forever for each RT-DEVS Atomic model
5.    wait for tN
6.    if a RT external input event
7.      send x msg;
8.      send * msg; //P-DEVS formalism
9.    else if tN time out
10.     send @ msg
11.     send * msg
12.   end if
13.   update tN
14. end forever
15. end main
```

Figure 4. Execution Algorithm of Root Coordinator

We start first by sending an initialization message (*I*) and collecting *done* messages from each of them (which report the next internal event for each component). Whenever there is an external input, the Root coordinator routes it through an external message (*x*) to the destination atomic model (which triggers the δ_{ext} function). Otherwise, it waits for the closest internal event (*λ1*) to send collect (@) and internal messages (*) to the target atomic model. The collect message executes the *λ* function on the atomic model and the internal message executes δ_{int}. The atomic model responds to the @ message by executing the *λ* function and re-

turning the output value through an output (*y*) message. The atomic model also executes δ_{int} in response to a * message, and returns its next internal event time by a *done* message.

```
1.  Receive X msg(s, e, x)
2.    push x in the Q_ext // PDEVS formalism
3.  end external

4.  Receive * msg(s)
5.    if (internal event)
6.      while (s is optional AND ta(s) <= now)
7.        Run δ_int function
8.      end while
9.      if (s is mandatory)
10.           Run δ_int function
11.     end if
12.   else if (external input in Q_ext)
13.     Run δ_ext function
14.   else if (both external and internal events)
15.     Run δ_con function
16.   end if
17.   send done msg
18. end internal

19. Receive @ msg(s)
20.   if (s is optional AND ta(s) < now OR d(s)<now)
21.     send done msg
22.   else
23.     Run λ function
24.     send y msg
25.     send done msg
26.   end if
27. end collect
```

Figure 5. Execution Algorithm of the Atomic Model

Whenever there is more than one internal event to be serviced, the mandatory ones have priority over the optional events. If an optional internal event is to be serviced later than its release time, its output will be discarded. This strategy helps us save time for later mandatory events that have not been released yet. Whenever a sequence of optional events in an atomic model is delayed, the atomic model jumps to the most recent one by executing multiple state changes in one δ_{int} (line 6). This way, we reduce the multiple internal messages transmitted between the Root coordinator and the atomic model and also multiple δ_{int} executions. Figure 5 shows the execution algorithm for atomic model.

Figure 6.a) shows a simple Imprecise DEVS model hierarchy where two atomic models *B* and *C* are coupled into *D*, which is itself coupled with atomic model *A*. Various input/output ports are used to connect the various models in the figure. Figure 6.b) shows the description of model *A* using DEVS Graph [36]. Note that continuous lines indicate external transitions and dashed lines indicate internal transitions. As we can see, the model is initially in state *A1* (with time advance = infinity) until an input *xa* is received on port *InA*. In that case, the external transition produces a state change to *A2*. The model stays in this state for *1t* and its deadline is *4t*. When the time is consumed, it produces the output *y2a* and transitions to *A3* (internal transition). A similar scenario can be seen in states *A3*, *A4* and *A5*, with outputs *y3a*, *y4a* and *y5a* produced respectively. Figure 6.c) and d) show the DEVS Graphs for atomic models B and C, respectively.

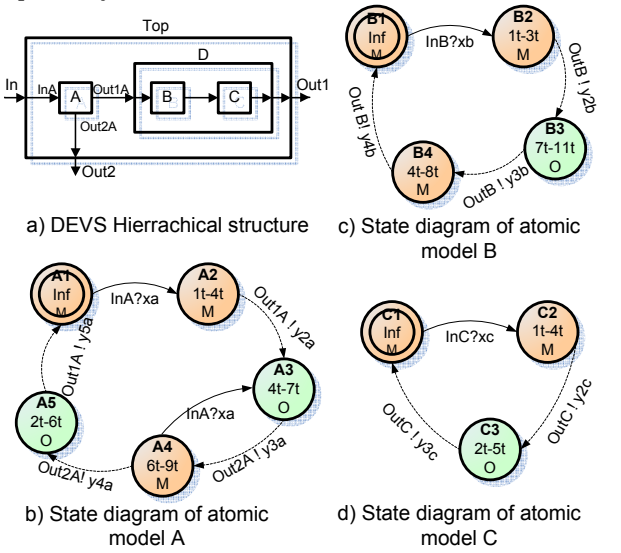

Figure 6. DEVS model definition

We can map the DEVS Graph of atomic model C (shown in Figure 6.d) as follows:

$C=< X, S, Y, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta, d >$, where:

X= {(InC, xc)},
S={(C1,mandatory), (C2, mandatory), (C3, optional)}, and S_0 = C1,
Y={(OutC, y2c), (OutC, y3c)},
δ_{ext} (C1, e, < InC, xc >) = C2,
δ_{int}(C2) = C3, δ_{int}(C3) = C1,
$\delta_{con} = \delta_{ext}$ has priority over δ_{int}
λ(C2) = <OutC, y2c>,
λ(C3) = <OutC, y3c>,
ta(C1) = ∞, ta(C2) = 1t, ta(C3) = 2t,
d(C1) = ∞, d(C2) = 4t, d(C3) = 5t,

The mapping of atomic models A and B are similar and straightforward.

In this example, the state durations are considered very small; however, in reality they are usually longer, compared to the execution time of the *X*, *λ* and *1* tasks. In a system with large number of atomic models, similar overload conditions can happen at different points of time, when multiple *X*, *λ* and *1* tasks from different atomic models are very close

to each other. For instance, Figure 7 shows a possible overload scenario for the DEVS model presented in Figure 6 without considering imprecise computation technique. An input *Xa* enters the system from input port *In* at time zero. Assuming the *X* task takes *1t*, at time 1 (i.e. *1t*) the atomic model *A* moves from the initial state *A1* to *A2*. The ta(s) of state *A2* is *1t*, thus at time 2, we run task *λ212*, producing the output *y2a* (for simplicity reasons we do not show the outputs) and the internal transition from *A2* to *A3*, (as specified in Figure 6.b). The output produced by the atomic model *A* (*y2a*) is translated to an input for the atomic model *B*. Thus, the task *Xb* is executed right after *λ212*, causing the atomic model *B* to change from *B1* to *B2*. The models advance according to the specifications (provided in Figure 6) until t=18. At this point, the tasks *λ414* of A, *λ313* of C and *λ212* (of A, B and C, shown in red) miss their deadlines because of the overload condition in the system.

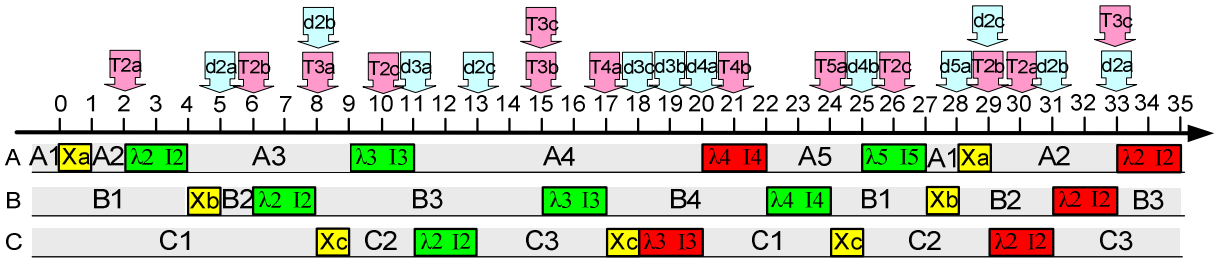

Figure 7. Example transient overload scenario

On Figure 6.b), c) and d), we marked the mandatory and optional states with an M or an O, respectively. By applying the proposed imprecise DEVS technique, *λ3* of A is skipped (because state A3 is optional and *λ3I3* is executed after its release time, *T3a*), causing *λ4I4* to be shifted to time 16 and saved from lateness. The same condition happens for *λ3* of B and C. Hence, by discarding three optional *λ* tasks we save the four mandatory tasks and their associated outputs.

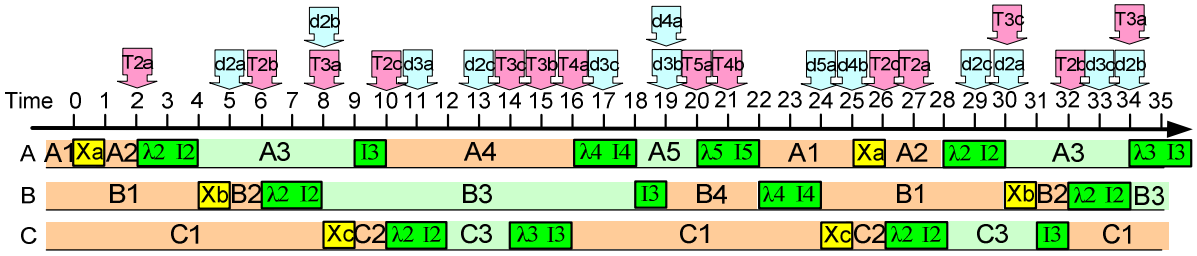

Figure 8. Applying imprecise computation to the sample scenario

4. IMPLEMENTATION AND RESULTS

We implemented the proposed imprecise DEVS formalism on E-CD++ [18], a toolkit that implements RT DEVS formalism proposed in [19] on the Xenomai RT framework [37]. Xenomai provides a RT kernel resting between the hardware and Linux OS, and offers several pervasive hard RT services to user space applications and is seamlessly integrated with GNU/Linux environment. We made *X* tasks user configurable (i.e. *periodic* or *aperiodic*), and their main job is to run user-defined input driver programs as soon as they are spawned. A main RT task implements the DEVS run-time abstract algorithm and takes care of *λI* tasks. This task is also responsible to implement and verify the imprecise DEVS formalism and its execution. The implementation of the imprecise computation on E-CD++ is seamless and backward compatible (i.e. the previous models also can be executed and are considered as precise models).

The proposed implementation of imprecise DEVS on E-CD++ has been tested with variety of modeling scenarios and several criteria has been applied for verification of the final implementation. For instance, we used a synthetic robotic model with 20 atomic models, each of them connected to an external input port, connected to a sonar distance sensor and an output port connected to an electrical motor. To ensure that the same scenario runs every time, the values coming from the sensors were the same in all tests. All the atomic models follow the DEVS Graph diagram in Figure 9. The model is a synthetical representation of a robot controller, which receives inputs from sensors and based on the inputs, instructs the motors. We used 20 atomic models to make it a computation intensive model where overrun situation happens frequently. The DEVS Graph diagram in Figure 9 is composed of three optional states and three mandatory states. Whenever there is an input in states C, D, E, and F the model transitions to state B. We use this model

to perform comprehensive performance tests, and compare the results of the imprecise execution and precise execution. In the case of precise execution, all the states are assumed mandatory.

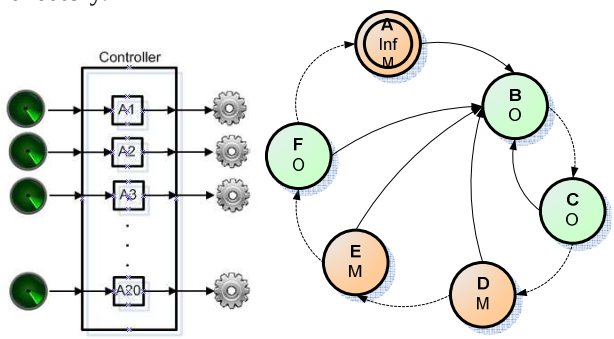

Figure 9. Synthetic robotic model used for verification

The timing for the component models varied for the different tests, performed. The first test discussed in this section compared the number of discarded λ and I tasks versus processor utilization. The diagram in Figure 10 shows the results of this test, for a total execution time of 20 seconds.

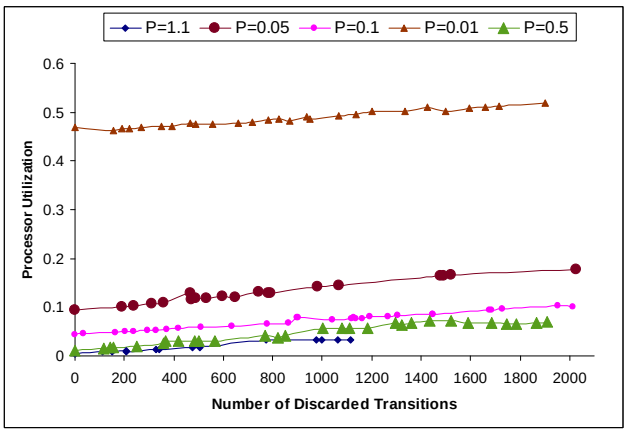

Figure 10. Discarded tasks vs. processor utilization

The test was performed for input period intervals of 1.1, 0.5, 0.1 and 0.001 s. As it is observed from the chart, by increasing the number of discarded tasks (which happens by tightening the state durations and period of the inputs) the processor utilization increases linearly. The result demonstrates the integrity and persistency of the implementation in a medium load scenario. In addition, as the system gets busier the number of discarded tasks also increases. The slope of the diagram for different period configurations stays the same, showing the integrity of the functionality of the algorithm for different levels of load on the processor.

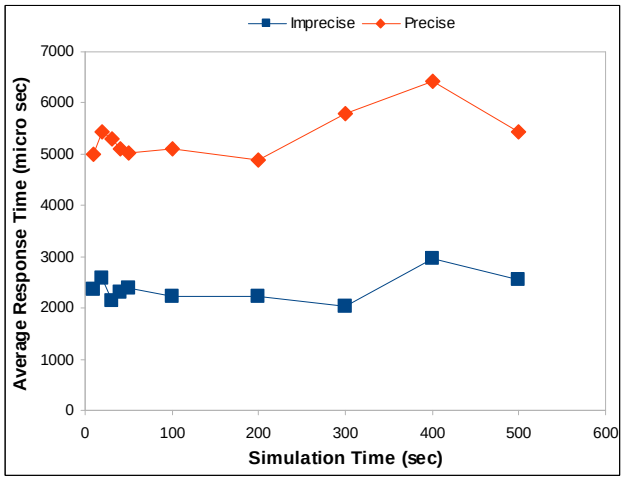

Figure 11. Response time vs. execution time-heavy load

Figure 11 shows the average response time of all the mandatory λI jobs versus the execution time for the same model using imprecise and precise modes. In this case, the input period of all X jobs was fixed (2 ms). The test was performed five times for each instance and the average result has been considered. As the chart shows, the average response time of the mandatory λ jobs drops dramatically in imprecise mode. In this example, there is a heavy load that the system must respond to, which required longer time for mandatory λ jobs to complete in precise mode. Imprecise computation discards the optional tasks, thus the response time of the mandatory tasks shortens.

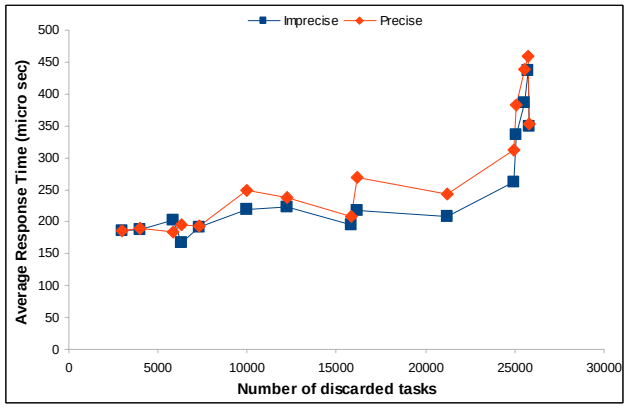

Figure 12. Number of discarded tasks versus average response time in medium load

Figure 12 shows the average response time of the model versus the number of discarded tasks for 20 seconds of execution time. The period of inputs is set to 50 milliseconds, and by varying the state durations, we obtain different number of discarded tasks in imprecise mode. For each in-

stance of the imprecise test, the same configuration was used to run in precise mode and find the average response time of the corresponding number of discarded tasks in precise mode. We can see that the average response time of the corresponding precise execution for each instance is slightly higher than the imprecise one in medium load scenario. The chart shows that by increasing the number of discarded tasks (i.e. tighter state durations) the average response time also increases. However, that this increase is not smooth as the situations change for different state durations.

Figure 13 depicts the processor utilization versus the number of discarded tasks in a heavy load scenario with the input period of 2 milliseconds and 20 seconds execution time. The chart shows steady but higher processor utilization for precise execution. The processor utilization for precise execution in all instances of the test is almost full, therefore as the load increases; the utilization remains almost the same. However, the imprecise processor utilization is instable and decreases as the number of discarded tasks increases. This is due to the instable and varying conditions that occur in a very heavy load scenario in imprecise mode. As the number of the discarded tasks increases, less processor usage is required. This decrease is not smooth neither linear, because of the change in conditions in each run, and admission of more mandatory jobs.

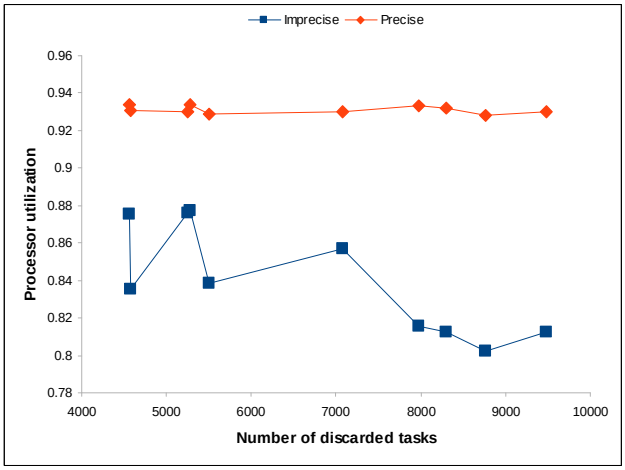

Figure 13. Discarded tasks vs. CPU utilization-heavy load

5. CONCLUSIONS

The development of embedded RT systems with RT constraints has been studied by the software engineering community in the last 20 years. The novel Imprecise DEVS (I-DEVS) formalism we proposed enables model designers to assign priority to the model's behavior and balance the execution burden based on the priorities assigned. The new approach can be easily integrated with previous models providing RT DEVS environment capable of managing different high processing conditions and integrate that with the RT DEVS engine. The approach provides flexibility to the user by prioritizing different behaviors of the system under control, while achieving the maximum throughput from the processor. The implementation and results of different tests have been presented.

The future work for this research includes incorporation of Dynamic DEVS formalism with the proposed I-DEVS formalism to introduce a new imprecise DEVS capable of prioritizing different components of the model besides the behaviors. Schedulability analysis will be applied to the proposed implementation to measure the possibility of the execution of the model on an specific hardware platform.

References

[1] H. Saadawi, G. Wainer. "Verification of Real-Time DEVS Models". In proceedings of DEVS Symposium, San Diego, California, USA, 2009.

[2] G. Wainer. "Discrete-event modeling and simulation; a practitioner's approach". CRC/Taylor & Francis. 2009.

[3] K. Lin, S. Natarajan, J.-S. Liu. "Imprecise Results: Utilizing Partial Computations in Real-Time Systems". In proceedings of the IEEE 8th Real-Time Systems Symposium, San Jose, California, USA, 1987.

[4] H. Kobayashi, N. Yamasaki, "RT-Frontier: A Real-Time Operating System for Practical Imprecise Computation". In proceedings of the 10th IEEE Real-Time and Applications Symposium, Toronto, Canada, 2004.

[5] G. R. Wiedenhoft, A.A. Fröhlich. "Using Imprecise Computation Techniques for Power Management in Real-Time Embedded Systems". 6th IFIP Working conf. on Distributed and Parallel Embedded Systems, Milano, Italy. 2008.

[6] J.W.S. Liu, K-J. Lin, R. Bettati, D. Hull, A. Yu. "Use of imprecise computation to enhance dependability of real-time systems". The International Series in Engineering and Computer Science, 1994, Volume 284, Section 3,157-182.

[7] W. Feng , J. W. S. Liu. "An Extended Imprecise Computation Model for Time-Constrained Speech Processing and Generation" Proceedings of the IEEE Workshop on Real-Time Applications, New York, NY. USA, 1993.

[8] X. Chen, A. M. K. Cheng. "An Imprecise Algorithm for Real-Time Compressed Image and Video Transmission" Proceedings of 6th International Conference on Computer Communications and Networks, Las Vegas, NV., USA 1997.

[9] X. Huang , A. M. K. Cheng. "Applying Imprecise Algorithms to Real-Time Image and Video Transmission" Proceedings of Real-Time Technology and Applications Symposium, Chicago, Illinois, USA, 1995.

[10] K. Fujisawa, S. Hayakawa, T. Aoki, T. Suzuki, S. Okuma. "Real Time Motion Planning for Autonomous Mobile Robot, using Framework of Anytime Algorithm" Proceedings of the IEEE International Conference on Robotics & Automation, Detroit, Michigan, USA, 1999.

[11] G. B. Parker. “Punctuated Anytime Learning for Hexapod Gait Generation” Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System, Beijing, China, 2002.
[12] S. Zilberstein , S. J. Russel. “Anytime Sensing, Planning and Action: A Practical Model for Robot Control” Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France, 1993.
[13] M. Amirijoo, J. Hansson, S. H. Son. “Error-Driven QoS Management in Imprecise Real-Time Databases” Proceedings of the 15th Euromicro Conference on Real-Time Systems, Porto, Portugal, 2003.
[14] J. Hansson, M. Thuresson, S. Son. “Imprecise Task Scheduling and Overload Management using OR-ULD” Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications, Cheju Island , South Korea, 2000.
[15] B. Zeigler, T. Kim, H. Praehofer. “Theory of Modeling and Simulation”. Academic Press 2000, ISBN-10: 0127784551.
[16] Hong J. S, Song H. H, Kim T. G., Park K. H “A Real-Time Discrete Event System Specification Formalism for Seamless Real-Time Software Development”, Springer, Netherlands,1997.
[17] X. Hu, B. P. Zeigler, J. Couretas, "Devs-On-A-Chip: Implementing DEVS In Embedded Java On A Tiny Internet Interface For Scalable Factory Automation." Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference, pp. 3051-3056.
[18] Y. H. Yu, G. Wainer, "eCD++: an engine for executing DEVS models in embedded platforms" Proceedings of the 2007 SCS Summer Computer Simulation Conference, San Diego, CA, USA, pp. 323-330. 2007
[19] M. Moallemi, G. Wainer, "Designing an Interface for Real-Time and Embedded DEVS”. Proceedings of TMS/DEVS Symposium. Orlando, FL. 2010.
[20] Furfaro A., Nigro L. “A development methodology for embedded systems based on RT-DEVS”, Innovations in Systems and Software Engineering, 5(2), pp. 117-127, 2009.
[21] Song H., Kim T. “Application of RT-DEVS to analysis of safety critical embedded control system: railroad crossing example” SIMULATION 81(2), pp. 119-136, 2005.
[22] Schulz, T. Ewing, and J.W. Rozenblit, “Discrete Event System Specification (DEVS) and StateMate StateCharts Equivalence for Embedded Systems Modeling”, Proc. of the IEEE Conference on Engineering of Computer-Based Systems, pp. 308-316, Edinburgh, Scotland, UK, April 2000.
[23] Spencer Borland and Hans Vangheluwe. “Transforming Statecharts to DEVS.” In proceedings of Summer Computer Simulation Conference. Student Workshop, pages S154 - S159. Society for Computer Simulation International (SCS), July 2003. Montréal, Canada.
[24] L. Capocchi, F. Bernardi, D. Federici, P. Bisgambiglia, “Transformation of VHDL descriptions into DEVS models for fault modeling and simulation” in proceedings of the IEEE Systems, Man and Cybernetics Conference, 2003, pp. 1205-1211, Washington, USA, 2003.
[25] C. Jacques, G. Wainer, “Using the CD++ DEVS toolkit to develop Petri Nets” In Proceedings of the 2002 Summer Computer Simulation Conference. San Diego, CA. USA. 2002.
[26] Tao Zheng, Gabriel A. Wainer, "Implementing finite state machines using the CD++ toolkit" In Proceedings of the 2005 SCS Summer Computer Simulation Conference (Student Workshop), Montreal, QC. Canada - 2003
[27] W.-K. Shih, J. W. S. Liu, J.-Y. Chung. “Algorithms for Scheduling Imprecise Computations with Timing Constraints”. SIAM J. Comput., 20(3):537–552, June 1991.
[28] J.W.S. Liu, W.K. Shih. “Algorithms for Scheduling Imprecise Computations with Timing Constraints to Minimize Maximum Error”. IEEE Trans. Comput., 44(3):466–471, 1995.
[29] H.Aydin, P.Mejia-Alvarez, R.Melhem, D.Mossé. “Optimal Reward-Based Scheduling of Periodic Real-Time Tasks”. Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.
[30] W.-K. Shih , J. W. S. Liu. “On-Line Scheduling of Imprecise Computations to Minimize Error” SIAM J. Comput., 5(5):1105–1121, 1996.
[31] S. Baruah, M. Hickey. “Competitive On-Line Scheduling of Imprecise Computations” IEEE Trans. Comput., 47(9):1027–1032, Sept. 1998.
[32] J.-Y. Chung, J.W. S. Liu, K.-J. Lin. “Scheduling Periodic Jobs That Allow Imprecise Results” IEEE Trans. Comput., 39(9):1156–1174, Sept. 1990.
[33] Chow A, Kim D, Zeigler B. “Parallel DEVS: A parallel, hierarchical, modular modeling formalism” Proceedings of Winter Simulation Conference, 1994, Orlando, Florida.
[34] Cho S. M. , Kim T. G. “Real-Time DEVS Simulation: Concurrent, Time-Selective Execution of Combined RT-DEVS Model and Interactive Environment” Proceeding of 1998 Summer Simulation Conference, Reno, Nevada.
[35] Wainer, G. "CD++: a toolkit to define discrete-event models". Software, Practice and Experience. Wiley. Vol. 32, No.3. pp. 1261-1306. November 2002.
[36] B.P. Zeigler, H. Song, T. Kim, H. Praehofer. "DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems". Proceedings of HSAC, LNCS, Vol. 999. Ithaca, NY. 1995.
[37] Xenomai: Real-Time Framework for Linux. Website available at: www.xenomai.org. accessed March, 2010.
[38] C. L. Liu , J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”. JACM, Vol. 20, No. 1, 1973.
[39] S. Manolache. “Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times”. Licentiate Thesis No. 985, Dept. of Computer and Information Science, IDA, Linkoping University, Sweden, December 2002