Scheduling Predictability in I-DEVS by Schedulability Analysis

Braulio Adriano de Mello
Universidade Federal da Fronteira Sul
108 Av. Fernando Machado, Chapeco, SC. Brazil
braulio@uffs.edu.br

Gabriel A. Wainer
Carleton University
1125 Colonel By Dr. Ottawa, ON. Canada.
gwainer@sce.carleton.ca

ABSTRACT

The Imprecise DEVS approach combines the advantages of imprecise computation with a formal modeling methodology in order to avoid the transient overloads on real-time systems. This process requires efficient scheduling methods to find the best schedule of the computations to guarantee the deadlines and to increase the quality of the results by reducing the discarding of the optional computations. This work introduces a solution to integrate schedulability analysis with Imprecise DEVS to improve the predictability and the feasibility for scheduling. The proposed schedulability analysis and scheduling methods are based on Earliest Deadline First, priority-driven and mandatory-first approaches and they are considered to be executed dynamically. The schedulability tests contribute to avoid unnecessary discarding of optional computations improving the quality of the results.

Author Keywords: Imprecise DEVS; Real-Time Systems; Schedulability Analysis.

1. INTRODUCTION

Hard Real-Rime Systems (RTS) have high design complexity, mainly due to complexity in meeting timing constraints. Failing to guarantee that all the computations meet their deadlines could be catastrophic. Most designing methods for RTS are complex to apply mostly on large scale systems and they do not guarantee free-errors systems. Modeling and Simulation (M&S) techniques have shown to help in reducing the effort and the cost for the overall designing process of RTS [15].

However, the M&S techniques often require great effort to model features of specific target systems, for example, the timing constraints on the RTS. The Imprecise DEVS formalism [16] is a recent attempt that extends a model-driven framework to develop RTS based on the DEVS [26] formalism and the Imprecise Computation (IC) technique [13] to deal with timing constraints. In the IC technique, parts of the computations could be discarded to guarantee the deadlines if imprecise results are acceptable [12, 14]. In this case, the computations are divided in mandatory and optional parts. As the mandatory parts affect the correctness of the result then all of them must be completely executed. Optional parts affect the quality of the result and they can be discarded if they will not meet their deadlines.

SpringSim-TMS/DEVS, 2016, April 3-6, Pasadena, CA, USA.

In order to guarantee the deadlines as well as the quality of the results, an efficient scheduling algorithm is required to determine the best order to execute the computations. Most existing research efforts spend efforts to improve the scheduling algorithms, however, there are no works based on addressing the best order for scheduling computations integrated in the M&S environments. We propose a new methodology that extends I-DEVS to integrate schedulability analysis to improve predictability and feasibility for scheduling of computations. The schedulability tests are based on the Worst Response Time (WRT) and on the Worst Case Execution Time (WCET) measures [27, 21, 25].

Our approach makes I-DEVS able to identify when a given set of computations is schedulable or not dynamically. If it is not schedulable, the schedulability test is able to show how to improve the schedule to guarantee the execution of all mandatory computations, and reducing the discarding of the optional ones. It combines the mandatory-first and priority based approaches with WRT and Earliest-Deadline First scheduling in a new method for M&S environments to simulate hard RTSs.

2. BACKGROUND

The workload is one of the most critical issues for real-time systems. When the system needs more computing resources than those available in order to guarantee the timing constraints (transient overload), then it might be impossible to meet the deadlines [5]. Flexible applications based on the IC technique [12] are able to reduce the resource demands by degrading the quality of the results gracefully at runtime while keeping the result quality acceptable [14, 13]. In hard RTS, graceful degradation is better than obtaining late results.

The IC method helps overcoming overload scenarios by dividing the computations into mandatory (M) and optional (O) parts. All the mandatory parts must be completed in time to guarantee the correctness of the system. Optional parts can be discarded if there are no resources available to execute them in a timely fashion [3, 7]. A system is considered *feasible* if all the mandatory computations of the system are schedulable. To study this, different scheduling algorithms have been proposed. These algorithms must be able to guarantee feasibility and, in general, they are priority-driven and they usually execute the mandatory computations first (mandatory-first approach) [14]. In the following section, we discuss some of these algorithms.

2.1. Scheduling Algorithms

The most popular priority-driven algorithms for RTS are the Rate Monotonic (RM), Deadline Monotonic (DM) [1] and Earliest Deadline First (EDF) [5]. RM and DM work dynamically with static priority. EDF works dynamically, and it could works with static or dynamic priority and it is considered optimal on single processor systems (when the CPU is not overloaded). RM uses a periodic scheduler and it assigns high priorities to the computation with the biggest rates (smallest periods). DM assigns high priorities to the computations with the nearest deadline, which must be equal to or less than the period [19]. EDF assigns priorities dynamically and the computations with the earliest deadlines have the highest priorities. EDF allows more efficient exploitation of computational resources and better responsiveness of non-periodic activities.

The IC technique [13] also has been applied for scheduling algorithms that try to improve performance under transient overloads. Scheduling policies with IC provide better results in comparison with their original versions [25]. In [7, 3, 19], the performance of EDF, RMS, Most Execution Time First (MEF) and Least Execution Time First (LEF) were compared. The results showed that EDF had the best performance. Simulation environments as MAST [8], AgaPé-TR [24] and Times [2], have been used to study scheduling algorithms based on the simulation of RTSs scheduling. These environments allow schedulability testing, introducing a complex analytical problem discussed in the following section.

2.2. Schedulability Analysis of Real-Time Systems

Schedulability analysis of RTSs [10, 27] is useful to predict whether a set of computations will meet their deadline. The success depends on whether all the computations can be guaranteed to complete before their deadlines. If this can be guaranteed, the set of computations is said *schedulable*. Schedulability analysis methods [20] have been classified according to the approach used, if the schedulability tests are executed, if they are done statically or dynamically, and if they are used to plan the schedule. Based on this, the algorithms can be considered static or *dynamic*. The first type performs the tests statically, and the resulting schedule is used at runtime. The second type checks the schedulability dynamically, which has a higher computational cost.

Baker [4] presents schedulability analysis approaches over EDF, addressing monotonic schedulability of RTSs. Sun and Lipari [23] presents a schedulability test for sporadic RT tasks using Global Fixed Priority on a multiprocessor system. The probabilistic guarantee [11] improves the results of scheduling of *soft* RT applications where the arrival times of processes and the processing times are random. However, as it does not guarantee that all the computations meet their deadlines, it is not acceptable for schedulability in hard RTS. Kuo [10] presents an online test for the periodic and multiframe process in uniprocessor environments for online admission control of newly arrived computations. Schedulability analysis for fixed priority scheduling based on WRT are exact, dynamic schemes seem to be more effective [27].

It is important to consider both IC and schedulability analysis on the design phases of RT systems. However, as of today, no M&S environment combines these. In order to contribute in this field, we proposed the Imprecise DEVS framework, which integrates IC with DEVS and opened new perspectives towards the improvement of predictability and feasibility of scheduling in M&S environments [16]. We extended Imprecise DEVS in order to integrate schedulability analysis, as the integration of schedulability analysis on the early phases of the design can improve the scheduling algorithms efficiency affecting the quality of the results.

2.3. Imprecise DEVS

Imprecise DEVS (I-DEVS) [15, 16, 17] integrates IC with DEVS in order to model RT systems. The main objective is to provide an imprecise framework for applications where the job arrival times are not known a-priori. The approach balances the computation when the system is busy while keeping the runtime overhead as low as possible. This load balance is based on the computations priorities.

I-DEVS added *d* to the atomic model and *c* to the definition of states as following:

d: $S \rightarrow R^1_{0,\infty}$ is the **relative** deadline of each state for output production

S:{(s,c) | s ∈ Z^1_0 and c ∈ (mandatory, optional)}

The coupled model is defined as DEVS where:
CM=<X,Y,D, {Mi | i ∈ D}, EIC, EOC, IC>
and D is a set of components and for each *i* in D
and $M_i = \{X_i,Y_i,S_i,\delta_{exti},\delta_{inti},\delta_{coni},\lambda_i,ta_i,d_i\}$ is the I-DEVS basic structure.

Despite the I-DEVS definitions of *d* and *c* are enough to manage the mandatory and optional computations, the schedulability analysis approach requires a known set of computations at a given state time. For each computation, it is essential to know the execution time, the release time, the deadline and if the computation is mandatory or optional. We extended the formal definition of I-DEVS to allow the execution of the schedulability tests, presented following.

3. EXTENDING I-DEVS

In I-DEVS, whenever there is more than one internal event (computation) 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 cannot be used to apply schedulability tests because it is done by the DEVS *Simulator*. Instead, schedulability tests should be done by the *Coordinators*, as the optional computations that will not meet their deadline must be discarded earlier (before sending them to

the Simulator). We will discuss the definition of set of computations managed *Coordinators*.

Coupled models connect basic models hierarchically, and they are defined as in Section 2.3. We will define the set of computations which is based on the definition of set of components where:

for each D, M_i is a component defined by $M_i = \{X_i, Y_i, S_i, \delta_{exti}, \delta_{inti}, \delta_{coni}, \lambda_i, ta_i, d_i\}$

Then, $K=(C_1, C_2, \ldots, C_n)$ is the *set of computations* of components D.

In the following, we say a *computation* is an internal event with finite worst case execution time (w_i) and it can be said to be a *mandatory* or *optional* computation (c_i). Each computation has a *release time* (r_i) defined by the beginning of the computation time, which, according to I-DEVS, it is equal to the arrival of an input to the coupled model or an input on the input port of the atomic model. The execution time of each computation is limited by its *relative deadline* $d(s_i)$. The set of components D can have multiple sets of computations of atomic models M_i defined by the set K_i.

Then $C_i=(r_i, c_i, w_i, d(s_i))$ denotes a computation of the atomic model i, where:

r_i is the release time,
$c_i \in$ (*mandatory, optional*),
w_i is the worst case execution time,
$d(s_i)$ is the relative deadline of the state s_i based on I-DEVS.

Each computation C_i is subject to timing constraints given by the *release time* (r_i), its worst case execution time w_i, and its maximum ending time defined by $d(s_i)$.

If there is a set of $C \in K$ and $\{(s, e, d(s)) \mid s \in S, 0 \le e \le ta(s)$ and $d(s) \ge ta(s)\}$, then for each $i \in M$, C_i is **schedulable** if $C(w_i) + w(hp(C_i) \in K) + e_i \le d(s_i)$.

where:

e_i is the elapsed time of C_i since its release time r_i

$C(w_i)$ is the worst case execution time of computation C_i

$w(hp(C_i) \in K)$ is the sum of all $C(w_i) \in K$ with higher priority (*hp*) than C_i where $(hp(C_i) \in K)$ is the set of all $C \in K$ with the priority greater than the priority of C_i

$d(s_i)$ is the relative deadline from the last state transition before s_i.

As a computation C_i is schedulable only if $C(w_i) + w(hp(C_i) \in K) + e_i \le d(s_i)$, then the set K of computations is said schedulable only if all $C_i \in K$ are schedulable on a single CPU at a given current state (instant of time).

The set K must include all C_i of the model at a given time to execute the schedulability analysis. However, as each Coordinator manages only the computations of its own Simulators, we need to flatten the hierarchical structure to have the set of computations of all Coordinators in the set K. The Flattened Coordinator [9, 22] strategy transforms a hierarchical structure of the coupled model to a flattened structure with depth one by eliminating intermediary Coordinators. The transformation must preserve the original port linkage relationship among atomic models.

4. SYSTEM MODEL

The internal events of I-DEVS models are mapped into computations $(C_1, C_2, \ldots, C_n)$ to represent the hard RTS on a single CPU and they must meet their deadlines defined by $d(s)$. The computations are considered sporadic, and they have irregular arrivals. Their minimum interarrival time is denoted as T_i. If a computation C_i has its arrival time set based on a given elapsed time e_i from the last transition, the next arrival of C_i must occur on or after e_i+T_i.

As each computation C_i is performed on a given state s where $s \in S$, and $S=\{s,c\}$, and $c \in$ (*mandatory, optional*), then each computation can be mandatory or optional according to its current state s. All computations must be completed before their deadlines $d(s)$ to each state s.

The release time of each computation is equal to its arrival time. The arrivals are defined by content (Q,Y) or synchronization (@,*,*done*) messages according to the DEVS simulator. Before the atomic model executes an optional computation, it must have its schedulability analyzed at a level higher than the atomic model level to verify if the computation will meet its deadline. The WRT of the computation is given by $C(w_i) + w(hp(C_i) \in K) + e_i$. If the schedulability test shows that the computation will meet its deadline, then it will be sent to the Simulator. Otherwise, it will be discarded.

When there are multiply arrivals of computations to one or more atomic models simultaneously, all the computations of the set $K=(C_1, C_2, \ldots, C_n)$ must be verified. Computations that not meet their deadline are discarded. To do so, the Coordinator does not trigger the computation on the Simulator level by discarding the messages q, * or @. The results of the schedulability tests are assumed to be feasible. It means that all optional computations sent to the atomic model will meet the deadlines. Mandatory computations are always sent to the atomic models.

The scheduling is performed at the same level. As the current schedulability tests solution is not be able to deal with preemptive scheduler, we assumed that the scheduler is non-preemptable. After each new schedulability test, the set of computations with a new configuration of earliest deadlines is known, and the priorities are assigned dynamically. The scheduler follows the priority-driven based EDF algorithm.

The results are considered *precise* when all optional computations are executed. The maximum accepted (graceful) *degradation* of the results happen when all optional computations are discarded. The system is considered to *fail* when one or more mandatory computations do not meet their

deadline. Mandatory computations have higher priority over the optional ones, and we assume non-hierarchical Simulation structures (the flat Coordinator communicates directly with all Simulators). In order to perform the schedulability analysis on the Coordinator level as discussed in this section, the Coordinator algorithms were adapted. The next section presents these changes.

5. COORDINATOR ALGORITHMS

According to the DEVS simulation algorithms [6], the messages received by the Coordinator from top to a child *i* are sent to the child *i*, and then the identification of the child *i* is cached into a ***synchronization set*** (*SyncSet*). After, the Coordinator sends the '*' message to all the children in *SyncSet*, and waits until all (done, t_N)'s are received.

We proposed to change the order of these steps to make the Coordinator able to perform the schedulability tests over the *SyncSet* before sending the messages to the children. The messages received by the Coordinator (and their child destination) are cached into *SyncSet*. The state transition (or internal event) that each message represents is mapped to a computation based on $C_i=(r_i, c_i, w_i, d(s_i))$. Then, the schedulability tests are performed over the mapped computations in the *SyncSet* that are ready to be sent to the children at current time *t*. Based on the schedulability test result, the non-schedulable computations are discarded, and the scheduler works with the schedulable subset of computations to send them to the children.

Figure 1 illustrates the Coordinator algorithm when it receives a '*' message. If the Coordinator receives an internal message * from the parent Coordinator and there are inputs $q \in Bag$, instead of sending them immediately to the child (as in other DEVS Coordinators), it processes each q_j message stored in the Bag to be sent to a child *j*, and *j(q,t)* is cached into the *SyncSet* (lines 5-6). Following, the schedulability test is run over all the components of the *SyncSet.* The idea is to verify if each computation of *j(q,t)* (mapped to $C_i=(r_i, c_i, w_i, d(s_i))$) at current time *t* for all child $j \in$ *SyncSet*, and whose current state s_j is optional, will meet their deadlines (line 10). This is based on the worst response time of *i(q),* or (*wrt(i(q))*), given by $C(w_i) + w(hp(C_i) \in K) + e_i$ according to the definition presented in the section 3. If s_i is optional and $wrt(i(q)) > d(s_i)$ then *j(q,t)* is not schedulable because it will not meet its deadline, then the Coordinator will discard *j(q,t)* from the *SyncSet* (lines 12 and 13). If *j(q,t)* is schedulable, then the Coordinator sends it to the child (line 11) followed by * (line 22). Mandatory computations are always sent to the children (lines 16-18). When a Simulator does not receive the message due to the discarding, time is saved for mandatory computations. Keep in mind that for Real-Time simulators, the timing information is tied to the CPU's Real-Time clock, thus the algorithm can run in simulated mode (discrete-event) or Real-Time. The * message is always sent to the child to execute the internal transition. The atomic model executes δ_{int} in response to a * message, and returns its next internal event time by a *done* message (line 28 in Figure 1).

```
1 when a ( * , t ) msg is received from parent Coordinator
2     if tL ≤ t ≤ tN
3         for all q ∈ bag
4             for all receivers of q, j ∈ Iself
5                 q := zself, j (q)
6                  cache j(q,t) in the SyncSet
7             end for
8         endfor
9         for all j(q,t) in SyncSet
10            if sj is optional and j(q,t) is schedulable
11                send (q,t) to j
12            else
13                discard j(q,t) of the SyncSet
14                send (*,t) to j /* state transition
15            endif
16            if sj is mandatory
17                send (q,t) to j
18            endif
19        endfor
20        empty bag
21        for all i in the SyncSet
22            send ( *, t ) to i
23        end for
24        wait until all ( done, tN)'s are received
25        tL :=t
26        tN := minimum of components' tN's
27        clear the SyncSet
28        send ( done, tN ) to parent Coordinator
29    else raise an error
30    endif
31 end when
```

Figure 1. Coordinator algorithm: internal message

Similar changes were made in the Coordinator algorithms when receiving a collect (@) and output messages (*y*) from children. Whenever the Coordinator receives a @ message to be sent to the child *i*, *i*(@) is also cached into the *SyncSet*. Following, the Coordinator applies the schedulability test to verify if the computations of *i*(@) at the time *t* for all child $i \in$ *SyncSet* will meet the deadline based on *wrt*(*i*(@)). If discarded, the @ message is not sent to the Simulator. If the @ message is sent to the target Simulator, the Simulator responds to the @ message by executing the λ function and returning the output value through an output (*y*) message.

When the Coordinator receives an output message *y* from child *i* to be sent to the child *j*, the Coordinator translates the Output Message *y* into the External Message *q* at first, and then caches *j*(*q,t*) into the *SyncSet*. Before sending it to all of its receiving Simulators, the Coordinator verifies if the state of the *j* is optional and if *j*(*q,t*) is schedulable among all $j \in SyncSet$ at the current time *t*. If *j* is schedulable then *j*(*q,t*) is sent to its receiving Simulator. If not schedulable, the Coordinator discards *j*(*q,t*) from *SyncSet* and the message *q* will not be sent to the child.

Each Coordinator has its own *SyncSet* to manage the messages to the target Simulators. The schedulability tests are performed based on the *SyncSet* at the Coordinator level and each Coordinator sees only its own Simulators. Be-

cause of this, the schedulability tests and scheduling are not effective. There are two main strategies to avoid this limitation: a message passing protocol to centralize all *SyncSets* of Coordinators in the topmost Coordinator which will be responsible for schedulability testing and scheduling, or applying the Flattened Coordinator strategy [9, 22] to transform a hierarchical structure of the coupled model to a **flattened structure** with depth one by eliminating intermediary Coordinators.

This work assumes non-hierarchical structures where intermediary Coordinators are eliminated and the flattened top most Coordinator communicates directly with all simulators. The literature presents algorithms to transform a hierarchical structure of the model to a flattened structure by eliminating Coordinators and transforming hierarchical coupled model into a coupled model with depth one. Section 6 shows an example to discuss the schedulability test based on the worst execution time and EDF scheduling.

6. SCHEDULING AND SCHEDULABILITY

This section describes the schedulability testing process based on the EDF scheduling. The main EDF based scheduling aspects are discussed and, following, we present the process to verify the schedulability of a set of computations dynamically using the definition of *SyncSet*.

6.1. Scheduling

The scheduler works to determine the best order for scheduling of computations, and schedulability analysis helps the scheduler to minimize the number of discarded optional computations. In general, static scheduling algorithms are suitable for periodic computations with hard deadlines and dynamic algorithms are more suitable for sporadic or aperiodic computations. The schedule is said *feasible* if the timing constraints of all computations are satisfied. Most algorithms for IC scheduling are based on EDF. Here, we adopted EDF based strategies for scheduling IC (mandatory and optional) and for integrating schedulability analysis with I-DEVS.

The EDF algorithm is priority-driven and the computations with the earliest deadlines have the highest priorities. We follow the mandatory first approach to manage the priorities of mandatory and optional computations. We assume that all computations defined by a set $K=(C_1,C_2,..,C_i)$ on a given time t execute on a single CPU. Mandatory computations are always executed and optional ones are executed only if their scheduling is feasible according to the results of the schedulability test. The system is considered to fail when one or more mandatory computations do not meet their deadline. The EDF based scheduling algorithm works dynamically to assign the priorities as described on the following general steps:

- whenever there is a set K of computations with release time r ready to be executed at a given time t, the scheduler updates the priorities of the computations according to the value of c (mandatory computations have higher priority than optional computations);
- Considering the subset of $K(C(c_i))$ computations where $c = mandatory$, the computations with earliest deadlines have higher priority;
- Considering the subset of $K(C(c_i))$ computations where $c = optional$, let the computations with earliest deadlines have assigned highest priorities.

EDF assigns the priorities at runtime whenever there are new arrivals of computations. Here, the weight of all the computations is considered equal. Computations with different weights will be explored in future works.

6.2. Schedulability Analysis

If the *utilization factor* U of a set K of computations is equal to or less than 1 ($U \leq 1$, where 1 is the maximum capacity of a CPU), then the set K is schedulable by EDF. The Utilization Factor represents the fraction of CPU time used by the computation and it is defined by the following equation [5]:

$$U_i = \frac{w_i}{P_i} \quad (1)$$

Where w_i is the WCET of the computation i and P_i is the period. The minimum interval T_i between the arrivals of a sporadic computation i [27] is used as the period P_i for schedulability testing. As the sporadic computations behave like periodic computations with period T and deadline $d(s)$ (where $d(s) \leq T$) and the minimum interval can be set to be equal to the deadline, then $\boldsymbol{T_i = d(s_i)}$ and $\boldsymbol{P_i = T_i}$. In addition, the deadline can be equal to or less than the period. As we are considering the relative deadline from the time t of the last transition and this could be different for distinct $C_i \in K$, then the elapsed time e cannot be considered to analyze the schedulability. Then, the minimum interval is given by the equation:

$$P_i = d(s_i) - e_i \quad (2)$$

Then, given a set K of computations, K could be considered schedulable if $U \leq 1$ for all $U_i(C_i \in K)$ according the equation:

$$\sum_{i=1}^{n} U_i \leq 1 \quad (3)$$

The schedulability analysis requires previous knowledge of the WCET of each computation. The WCET provides information about the worst possible execution time of the computation before running it. Usually, the WCET can be defined by strategies like code analysis [25] or probabilistic techniques [11]. Here, we consider that the execution time w_i is specified at the project time for each computation to be used as the WCET on the schedulability testing.

The utilization factor is seen as a non-exact test to verify if the set K is entirely schedulable or not. If the schedulability test is negative, the set K is definitely not entirely schedula-

ble. If non-negative, it does not guarantee that all $C_i \in K$ are schedulable. In addition, it does not help the scheduler to identify which of optional computation could be discarded in case of transient overloads. This test is based on the following equations:

$$R^0 = w_i \quad (4)$$

$$R^{m+1} = w_i + \sum_{j \in hp(i)} \left\lceil \frac{R^m}{P_j} \right\rceil . w_j \quad (5)$$

The R is the interval between the release time and the end of the execution of the computation and it defines the WRT. The WRT [21] is one of the main EDF based methods used to test the exact schedulability of computations. Given a set K of computations, it permits to verify if each computation of the set K is schedulable or not. The equation (4) of this test is applied to the first calculation of R and the equation (5) is applied iteratively until: $\boldsymbol{R^n_i = R^{n+1}_i}$. The result is the maximum response time R_i of the computation i. The computation is schedulable when the maximum response time R_i is less than or equal to its deadline ($R_i \leq D_i$) and it considers the situation where the deadline is equal to or less than the minimum interval between the arrivals of the computation ($D_i \leq P_i$).

If there is a set K of computations to be executed at a given time t, then we assume the greatest minimum interval between the arrivals among all current computations of the set K to be the P_i to perform the schedulability tests of all C_i. Then, for all $C_i \in K$ at a given time t, $P_i = \{max[K(P_i)] \mid P_i = d(s_i) - e_i\}$.

This is to avoid pessimistic estimation [18] on the schedulability tests without losing the feasibility.

Figure 2 illustrates the example where three models have their computations executed on a single CPU. The execution is for a non hierarchical structure and there is one flattened topmost Coordinator that communicates directly with all Simulators.

In Figure 2, for instance, the computation $A(\lambda 2I2)$ is on the state $A2$ and its release time is equal to 2 ($r_{A2} = 2$), the computation is mandatory ($c_{A2} = mandatory$), and the execution time (WCET) is equal to 2 ($w_{A2} = 2$). The deadline is 4 ($d(s_{A2}) = 4$), and the elapsed time from the last transition of the model A is 1 ($e_{A2} = 1$). As $R_{A2} = 2$ (equation 4) and $RA_{A2} + e_{A2} \leq d(s_{A2})$ or (2+1≤4) then $A2$ is schedulable.

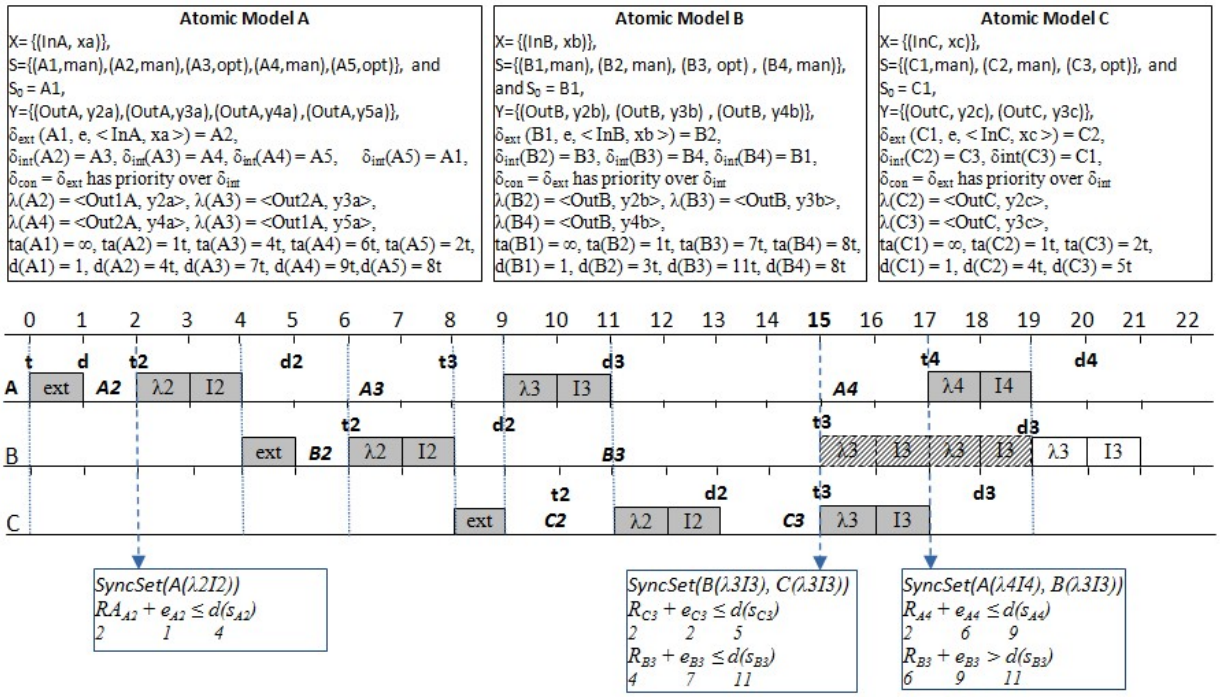

Figure 2. Sample of schedulability testing of the computations *B3*, *C3* and *A4* on a single CPU.

At time 15, the set K includes the computations $B(\lambda 3I3)$ and $C(\lambda 3I3)$, where $C(\lambda 3I3)$ has the highest priority, and there is no computation with higher priority (hp) than $C(\lambda 3I3)$ in K:

- $B(\lambda 3I3)$ is on the state $B3$, and r_{B3}=15, and c_{B3}=optional, and w_{B3}=2, and $d(s_{B3})$=11
- $C(\lambda 3I3)$ is on the state $C3$, and r_{C3}=15, and c_{C3}=optional, and w_{C3}=2, and $d(s_{C3})$=5

Then, applying the equation (4) as following:

$R^0_{C3} = w_i = 2$

Result: as the $R_{C3} + e_{C3} < d(s_{C3})$, then $C(\lambda 3I3)$ is schedulable at $t=15$ as shown in Figure 2.

Considering that the $C(\lambda 3I3)$ has the highest priority in the set K, it belongs to the set of computations with higher priorities than $B(\lambda 3I3)$. It is defined by $C(\lambda 3I3) \in hp(B(\lambda 3I3))$,

where *hp(i)* is the set of computations with higher priority than *i*. Then, the R_{B3} is therefore given by:

$$R^0_{B3} = w_i = 2$$

$$R^1_{B3} = w_i + \sum_{j \in hp(i)} \left\lceil \frac{R^0}{P_j} \right\rceil . w_j = 2 + \left\lceil \frac{2}{4} \right\rceil .2 = 4$$

$$R^2_{B3} = w_i + \sum_{j \in hp(i)} \left\lceil \frac{R^1}{P_j} \right\rceil . w_j = 2 + \left\lceil \frac{4}{4} \right\rceil .2 = 4$$

Result: as $R^1 = R^2$ at the second iteration and $R_{B3} + e_{B3} \leq d(s_{B3})$ then $B(\lambda 3I3)$ is schedulable and, considering the schedulability testing at time 15, it can be executed at 17.

However, before executing $B(\lambda 3I3)$, $A(\lambda 4I4)$ is released at time 17, so there are two computations, $A(\lambda 4I4)$ and $B(\lambda 3I3)$, ready to be executed at time 17 and $A(\lambda 4I4)$ has higher priority than $B(\lambda 3I3)$, where:

- $A(\lambda 4I4)$ is on state 4, and r_{A4}=17, and c_{A4}=mandatory, and w_{A4}=2, and $d(s_{A4})$=9
- $B(\lambda 3I3)$ is on state 3, and r_{B3}=15, and c_{B3}=optional, and w_{B3}=2, and $d(s_{C3})$=11

As there is no computation with higher priority (hp) than A(λ4I4) in K, then:

$$R^0_{A4} = w_i = 2$$

Result: as the $R_{A4} + e_{A4} < d(s_{A4})$ than $A(\lambda 4I4)$ is schedulable.

Considering that $A(\lambda 4I4)$ has the highest priority in the set *K*, it belongs to the set of computations with higher priorities than $B(\lambda 3I3)$, or: $A(\lambda 4I4) \in hp(B(\lambda 3I3))$. Then, R_{B3} is therefore given by:

$$R^0_{B3} = w_i = 2$$

$$R^1_{B3} = w_i + \sum_{j \in hp(i)} \left\lceil \frac{R^0}{P_j} \right\rceil . w_j = 2 + \left\lceil \frac{2}{3} \right\rceil .2 = 4$$

$$R^2_{B3} = w_i + \sum_{j \in hp(i)} \left\lceil \frac{R^1}{P_j} \right\rceil . w_j = 2 + \left\lceil \frac{4}{3} \right\rceil .2 = 6$$

$$R^3_{B3} = w_i + \sum_{j \in hp(i)} \left\lceil \frac{R^2}{P_j} \right\rceil . w_j = 2 + \left\lceil \frac{6}{3} \right\rceil .2 = 6$$

Result: as $R^2 = R^3$ at the third iteration and $R_{B3} + e_{B3} > d(s_{B3})$ then $B(\lambda 3I3)$ is not schedulable and it must be discarded as shown in Figure 2.

Whenever the schedulability test shows that the scheduling is feasible for all $C(c_i) \in K$, all computations can be scheduled. Whenever the schedulability test shows that there are optional computations that will not meet the deadline, they are discarded. If at least one computation with c_i = *mandatory* is not schedulable, the system fails.

7. CONCLUSION

We presented an approach to integrate schedulability analysis strategy with I-DEVS in order to improve the predictability and the feasibility for scheduling ICs. The approach is based on the EDF algorithm combined with the mandatory-first approach and schedulability testing is based on the WRT to verify whether each computation is schedulable or not according their timing constraints. This feature makes possible, for instance, to analyze whether the model can be executed on a specific hardware platform.

To integrate this strategy into the I-DEVS, we proposed to use the Synchronization Set as the main resource to implement the schedulability tests and the scheduling on the Coordinator level before triggering the computations on the Simulators.

In the future, we will study new scheduling and schedulability analysis methods. For example, we want to integrate sharing resources in the system models, the scheduling and schedulability analysis process could implement mutual exclusion functionalities in order to deal with deadlocks situations. Moreover, in our approach the priority inversion situations can happen when an optional computation has earlier deadline then a mandatory one and both computations could meet their deadlines if the optional computation is executed before the mandatory one. The proposed schedulability test solution does not deal with priority inversion scenarios. For example, in Figure 2, if *d4 = 21* instead *d4 = 20* then there was time enough to execute both computations *A4* and *B3* if *B3* was executed first. However, *A4* has highest priority according to the scheduling algorithm. This leads to priority inversion.

Finally, we consider that the WCET is defined at project time. A new functionality could be integrated to define the WCET dynamically such as we have done with the WRT.

ACKNOWLEDGMENTS

This research was supported by Universidade Federal da Fronteira Sul and the Brazilian research agency CAPES, process No. 1835-14-9. It was partially funded by NSERC.

REFERENCES

1. Altenbernd, P. Deadline-monotonic software scheduling for the co-synthesis of parallel hard real-time systems. In Proceedings of the 1995 European conference on Design and Test, Washington, DC, USA, 190, 1995.
2. Amnell, T. F., E. Mokrushin, L., Pettersson, P. and Yi, W. “TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems”, Formal Modeling and Analysis of Timed Systems Lecture Notes in Computer Science, Volume 2791, (2004) 60-72.
3. Aydin, H., Melhem, R., and Mosse, D.Optimal scheduling of IC tasks in the presence of multiple faults. In Real-Time Computing Systems and Applications, Cheju Island,South Korea, 2000, 289–296.
4. Baker, T. P. Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In Proceedings of the 24th IEEE International Real-Time Systems Symposium . Washington, DC, 2003, 120-129.

5. Buttazzo, G.C. “Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications”, second ed., Springer-Verlag, New York, (2004).
6. Chow, A. C. “Abstract simulator for the Parallel DEVS formalism”. AI. Simulation, and Planning in High Autonomy Systems, 1994.
7. Guo, C, Zhu, C. Tay, T.T. “Design and Simulation of a Green Broker with Imprecise Computation Scheduling for Energy-efficient Large Scale Computing in Clusters”. In Journal of Emerging Trends in Computing and Information Sciences, Vol. 4, No. 12, Dec. 2013.
8. Harbour, M. G. García, J.J. G. J.C. Gutiérrez, P. and Moyano, J.M. D. MAST: Modeling and Analysis Suite for RT Applications. Proc. 13th Euromicro Con-ference on Real-Time Systems, Delft, Netherlands, 2001.
9. Kim, K., Kang, W., Sagong, B. and Seo, H., Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One. In Proceedings of 33rd Annual Simulation Symposium, Washington, D.C., 2000.
10. Kuo, T.W., Chang, L.P., Liu, Y.H. and Lin. K.J. “Efficient Online Schedulability Tests for Real-Time Systems”. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. 29, No. 8, Aug. (2003).
11. Li, W., Wang, G. and Zhao. W. Stochastic Analysis of Expected Schedulability for Real-Time Tasks on a Single Computing System. In Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, 2008.
12. Lin, K., Natarajan, S. and Liu,J.-S. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In proceedings of the IEEE 8th Real-Time Systems Symposium, San Jose, California, USA, 1987.
13. Liu, J. W. S., Shih, W.-K., Lin, K.-J. R. and Bettati, J.-Y. Chung. Imprecise Computations. In Proceedings of the IEEE, Vol. 82, No. 1, pp. 83-94, Jan. 1994.
14. Liu, J. W. S. “Real-Time Systems”. (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
15. Moallemi, and M. Wainer, G. A. Designing an Interface for Real-Time and Embedded DEVS. In Proceedings of TMS/DEVS Symposium. Orlando, FL. 2010.
16. Moallemi, M. and Wainer, G. A. I-DEVS: Imprecise Real-Time and Embedded DEVS Modeling. In Proceedings of 2011 Spring Simulation Conference, DEVS Symposium, 2011, 95-102.
17. Moallemi, M. Wainer, G. A. "Modeling and simulation-driven development of embedded real-time systems". Simulation Modeling, Practice and Theory. Elsevier, Volume 38, Nov. 2013, 115-131.
18. Nasri, M., Baruah, S., Fohler, G. and Kargahi, M. On the Optimality of RM and EDF for Non-Preemptive Real-Time Harmonic Tasks. In Proceedings of the 22nd International Conference on Real-Time Networks and Systems. New York, NY, 2014, 331-340.
19. Palencia, J. C. and Harbour, G. M. “Response time analysis of EDF distributed real-time systems”. J. Embedded Comput. 1, 2 (April 2005), 225-237.
20. Ramamritham,K., Stankovic,J. A. and Shiah, P. F. Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems. IEEE Trans. Parallel Distrib. Syst. 1, 2, (1990), 184-194.
21. Ripoll, I., Crespo, A. and Mok, A. K. Improvement in feasibility testing for real-time tasks. Real-Time Syst. 11, 1 (July 1996), 19-39.
22. Shang, H. and Wainer, G. A. Dynamic structure DEVS: Improving the real-time embedded systems simulation and design. In Annual Simulation Symposium. IEEE Computer Society, 2008, 271–278.
23. Sun, Y. and Lipari, G. A Weak Simulation Relation for Real-Time Schedulability Analysis of Global Fixed Priority Scheduling Using Linear Hybrid Automata. In Proceedings of the 22nd International Conference on Real-Time Networks and Systems. New York, 2014.
24. Wainer, G. A. Experiencing with AgaPé-TR: a simulation tool for local real-time scheduling. In Proc. 4th IFAC/IFIP Workshop of Algorithms and Archi-tectures for Real-Time Control, Lisbon, Portugal, 1997.
25. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J. and Stenström, P. 2008. The worst-case execution-time problem-overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. 7, 3, Article 36 (May 2008), 53 pages.
26. Zeigler, B., Kim, T. and Praehofer, H. 2000.“Theory of Modeling and Simulation”. Academic Press.
27. Zhang, F. and Burns, A. “Schedulability Analysis for Real-Time Systems with EDF Scheduling”. IEEE TRANSACTIONS ON COMPUTERS, VOL. 58, No.9, Sep. 2009, 1250-1258.