

Integrating I-DEVS and schedulability methods for analyzing real-time systems constraints

Braulio A Mello¹  and Gabriel A Wainer² 

Abstract

The design of embedded real-time systems (RTS) is challenging due to the criticality of the timing constraints of these systems. Various informal and formal methods for RTS design have been proposed, both in the design space and the real-time execution at the hardware level, but many of these methods are not effective when the complexity of the system scales up. Here, we discuss a new method to integrate a modeling (and simulation) formalism that allows designing complex systems specifications for real-time constraints called Imprecise-DEVS (I-DEVS), and the mapping of such high-level models into a real-time task model. This method allows analyzing real-time constraints both at the high level of modeling as well as the low level of the tasks executed by the processing units and the Operating System. A new method to study the schedulability of the task models is proposed. The method provides a design analysis space from the model level, up to the individual tasks, with a focus on the schedulability of real-time constraints under transient overloading conditions.

Keywords

Imprecise-DEVS, real-time systems, schedulability analysis, simulation

1. Introduction

Embedded systems are tightly coupled hardware/software systems designed to perform specific functions. Nowadays, these embedded systems are ubiquitous (medical devices, vehicles, industrial control systems, etc.) and many of them have real-time requirements. They normally combine hardware, software, physical devices and environment they control, and must satisfy critical timing requirements,¹ which makes their design challenging. Different design methods have been proposed for building real-time systems, including informal approaches such as *real-time structured analysis* or formal approaches such as *Petri Nets* or *process algebras*.² These methods can be used to define and study the timing restrictions of the system at a high level; for instance, in Haur et al.³ where model checking is used to study schedulability using a *Stopwatch Petri Net* that allows determining under which temporal conditions the application is schedulable. These methods are useful at the design stage but cannot deal with implementation issues or low-level design at the processor scheduling level, i.e., these formal methods cannot be applied to study scheduling of computations under timing constraints in real-time systems. In addition, the bridge

between formal methods for design, and detailed scheduling problems, has not been explored. Combining methods to mitigate their low effectiveness in earlier or later phases of development of the real-time systems could increase the design cost and complexity.^{4,5} In addition, Modeling and Simulation (M&S) has been shown to be useful in reducing the effort and the cost for the overall designing process of real-time systems bypassing the difficulties of other approaches.⁶ M&S allows the designer to experiment with varied scenarios in terms of system loads, hardware/software configurations, in a virtual environment with reduced risk and improved overall costs. Nevertheless, informal M&S methods cannot be used to provide formal guarantees on the correctness of the system under development.

¹Federal University of Fronteira Sul, Brazil

²Systems and Computer Engineering Department, Carleton University, Canada

Corresponding author:

Braulio A Mello, Federal University of Fronteira Sul, 108 Av. Fernando Machado, Chapeco 89815-899, SC, Brazil.

Email: braulio@uffs.edu.br

This research focuses on the issues discussed above by using a formal M&S approach like the Discrete-Event Systems specifications (DEVS)^{7,8} formalism. DEVS is adequate to model real-time systems as it provides a homogeneous mechanism for analysis based on a combination of formal methods, modeling, and simulation. A formal M&S-driven approach provides a precise method for defining models and provides the tools for analyzing models formally as well as performing risk-free simulation-based tests.

The use of DEVS for development of real-time system has been explored in the past; in this research, we introduce new methods that allow designing real-time systems with real-time deadlines based on the DEVS formalism. The method permits building real-time models that can be later translated to task models for scheduling requirement analysis. We are interested in exploring methods that deal with real-time systems going through conditions that trigger unexpected responses that affect processing capacity (called transient system overloads). We want to study how to guarantee DEVS models that can execute predictably in a target platform using scheduling algorithms that will perform adequately under these transient overloads. The timing faults that inevitably occur whenever the real-time system becomes overloaded can be handled using the well-established theory of imprecise computation (IC).⁹ This formal method showed to be a good solution to deal with overloaded real-time systems by discarding noncritical computations in organized fashion.¹⁰ To organize the dismissal of noncritical computations, IC theory identifies some of the computations as *mandatory*, and others as *optional*. The mandatory parts are those that will affect the correctness of the results,¹¹ and they must be fully executed. The optional parts affect mostly the quality of the result, and they could be discarded if they cannot meet the deadlines. The discarding of optional computations can be done in an organized way, providing graceful degradation.¹²

The Imprecise-DEVS (I-DEVS)¹³ method combines the formal advantages of DEVS and IC, providing a mechanism to define formal models with imprecise computing. New I-DEVS-based techniques to overcome overrun conditions under imprecise computing approach are presented in Wainer and Moallemi.¹⁴ Although I-DEVS provides a mechanism for defining and studying real-time models and their timing constraints, we need to translate the models into executable specifications into running in real-time computing hardware. To make this happen, we convert the models into real-time tasks, and those tasks execute predictably. Since there is no possibility to anticipate the new occurrences of mandatory or optional computations in those real-time systems at runtime, we need to provide schedulability analysis¹⁵ of the executable model with the goal of achieving predictability¹⁶ of the scheduled tasks. Schedulability analysis considers a set of computations and predicts whether each one of the computations

will meet their deadlines. If all the computations can be guaranteed to complete before their deadlines, the set is said *schedulable*. The methods we propose here are the first existing techniques that can provide an integrated method for designing predictable schedules for DEVS models, in particular when the models under execution experience transient overloading. The method composes an I-DEVS-based infrastructure for designing hard real-time systems, which is used to design the real-time system as a DEVS coupled model with IC components. The models can be formally verified, and we propose a technique to convert the high-level models into computational units to execute in real-time hardware. These computational units are then represented as real-time tasks, and a schedulability analysis of the DEVS computational units is provided. The schedulability analysis and modeling techniques are enhanced by including imprecise computing theory. The tests allow studying the system response under overloading conditions by applying schedulability testing over mandatory and optional computations. The schedulability algorithms are based on the worst response time (WRT) and on the worst-case execution time (WCET) methods^{15,17,18} using mandatory-first, priority-based, and earliest deadline first approaches.

The proposed method allows:

- Defining real-time applications as DEVS models, including behavior of the real-time application in atomic models, and integrating various real-time subcomponents into coupled models.
- Enhancing those models using I-DEVS and dividing the behavior of the models into mandatory and optional subcomponents.
- The runtime execution algorithms, as well as the simulation algorithms, act on the I-DEVS models and can be used to simulate system behavior as well as executing the models at runtime. In both cases, when transient system overloads occur (or are triggered in a simulation), the models react according to imprecise computing theory.
- The models are subsequently transformed into computational units that are represented as task models in a real-time timeline. The timeline can be formally analyzed for schedulability, integrating thus the high-level I-DEVS models and the low-level task execution, providing a robust method to analyze and guarantee timeliness of the real-time application, both under steady-state and overloading conditions.

The rest of the paper will discuss these issues as follows: the next section summarizes the scheduling and schedulability analysis of real-time systems under IC, and it introduces the Imprecise-DEVS formal definition. Section 3 presents the assumptions of the scheduling and

schedulability analysis in accordance with the Imprecise-DEVS. Section 4 describes the formulation of schedulability testing, and the DEVS algorithms to support the schedulability procedures. Section 5 presents a schedulability testing scenario. Section 6 concludes the paper with a summary of future work.

2. Background

As discussed in the introduction, independent of their features, purpose, or complexity, all embedded real-time systems must deal with timing constraints in dependable fashion. Timing constraints issues can be addressed at the high-level design process¹⁹ by using modeling methods such as timed automata²⁰ and model checking.^{19,21} Such methods address the design timing constraints at high level, guaranteeing the correctness of the design in terms of the timeliness requirements. Nevertheless, these theoretical results are based on ideal conditions, and when deployed in a hardware target platform, there are numerous real-world constraints that affect the timeliness analysis produced by the high-level designs, ranging from interrupt latency, sensor delays, failures in hardware, processor scheduling, and contention to access memory. These various factors make the timeliness analysis only partially useful. At the low level, we need to consider other aspects, starting by schedulability at the level of tasks (and other resources) at runtime, and this should consider timing and resource constraints² that are different than those studied by a high-level design. Bridging the gap between these two types of models needs exhaustive exploration, and our research introduces a new method to link high-level and low-level models using DEVS formal modeling. We also provide imprecise computing theory to deal with overloading conditions, extending schedulability tests (including mandatory and optional imprecise computations). This section discusses the different concepts related to this area of research, including I-DEVS formal definitions, real-time scheduling, and imprecise computing methods.

2.1. Schedulability analysis and scheduling of real-time systems

Schedulability analysis of real-time systems^{15,22–25} is useful to predict whether a set of computations will meet their deadlines according to their timing constraints. A set of computations is said *schedulable* if all the computations can be guaranteed to complete before their deadlines. One of the main challenges on real-time systems is to find a feasible method for evaluating the schedulability of computations using a well-defined scheduling algorithm. Most schedulability analysis methods are based on a task model

where a computation transforms an initial state with input values to a final state with output values.

Real-time schedulers decide which computations should be executed next. It assumes the WCETs, the deadlines, and the priorities of the computations are known. The most common real-time systems schedulers are based on priority.²⁶ The rate monotonic (RM) and the deadline monotonic (DM)⁹ algorithms work dynamically with static priority. RM uses a periodic scheduler, and it assigns high priorities to the computations with the shortest periods. DM assigns high priorities to the computations with the nearest deadline, which must be equal to or less than the period.²⁷ The earliest deadline first (EDF) works dynamically, with static or dynamic priority, by calculating the priorities at runtime while the scheduling is executed. The computations with the earliest deadlines have the highest priorities. It is considered optimal on single processor systems (when the computing resource is not overloaded).²⁸

The foundational article presented in Ramaratham and Stankovic²⁹ defines four types of scheduling algorithms based on these principles.

- *Static table-driven*: the schedulability analysis is done prior to the execution of the system, in static fashion, and the resulting schedule (in the form of a table used to schedule the system tasks) is used at runtime. The schedule cannot be changed.
- *Static priority-driven preemptive*: the schedulability analysis is done in static fashion, as above, and the computation with the highest priority is always executed first. If a higher priority task becomes available, executing tasks with lower priority can be preempted.
- *Dynamic planning-based*: the schedulability analysis is done at runtime, in dynamic fashion, and a computation is executed only if its WCET is less than its deadline.
- *Dynamic best effort*: there are no guarantees to ensure that the computation will meet the deadline.

In general, static approaches are applicable to periodic computations where the tests are performed statically, and the resulting scheduling table is highly predictable. However, this approach is not flexible since any change on a computation could affect the scheduling table. On the contrary, dynamic approaches can provide flexibility, feasibility, and predictability. When a computation arrives, these approaches are flexible for creating a new feasible schedule including the new computation before its execution begins. We can predict if the computations will meet their deadlines, making dynamic approaches more suitable for integrating schedulability analysis.

Baker³⁰ introduced schedulability tests over EDF and DM scheduling. Given a schedule of a sequence of

computations (defined as a window) with their release times and WCET, the tests can identify the earliest possible missed deadline. The WCET can be specified by methods such as code analysis¹⁸ or probabilistic techniques.^{31,32} The feedback-directed optimization process³³ uses compiler technology called to derive a WCET profile for a program that can be implemented in mainstream compilers, instead of using specialized compilers. A survey discussing recent contributions on estimation and optimization of WCET is presented in Meng et al.³⁴ Zhang and Burns¹⁵ showed that dynamic schemes using WCET are effective for sufficient schedulability analysis under EDF scheduling with arbitrary relative deadlines. Kuo et al.²³ presented new schedulability tests for single processor systems with heavy loads. Gunzel et al.³⁵ proposed two schedulability analysis methods for EDF on uniprocessor systems to keep the correctness when a computation task assigned to the processor may suspend itself. Sun and Lipari³⁶ presented a schedulability test for sporadic computations scheduled by global fixed priority on a multiprocessor system using the linear hybrid automata (LHA) formalism to represent the scheduler and the computations. The missed deadline conditions are modeled as errors in the automata. Guo et al.³⁷ use directed acyclic graphs to model parallel tasks, and they present a methodology to order the vertices in the graph, combined with schedulability analysis, and a priority-based scheduling algorithm. A review of uniprocessor real-time scheduling algorithms and schedulability analysis techniques is presented in Davis.⁵

The WCET can be applied to compute the WRT of the computations for evaluating their schedulability.³⁸ The WRT is an essential technique to verify the behavior of real-time systems, and simulation-based approaches have been applied to provide WRT estimation.³⁹ The WRT is the worst possible response time of a computation in a priority-driven environment. For example, if the set of computations (C_1, C_2, C_n) are ready to be executed at the same time on a single processor system, the WRT of the computation C_1 is given by the sum of its WCET and the WCETs of the computations with priority higher than C_1 . Therefore, the computation C_1 is said schedulable if its WRT is not larger than its deadline.

2.2. Scheduling of imprecise computations in real-time systems

Depending on the features of the system environment, real-time computations can produce different results for the same input values. These varied results do not mean that the execution sequences are incorrect. Likewise, in real-time systems, computations could be interrupted on their deadlines before completing their execution. Interrupting the computation early does not mean that the

results are not correct; nevertheless, the results could be imprecise.⁹ When imprecise results are acceptable, discarding of some computations could be advantageous to guarantee that the remaining computations meet the deadlines. To do so, the scheduling algorithms should be built in order to decide which of the computations should be executed (mandatory), and which could be discarded (optional).⁴⁰ For instance, the milestone and the sieve approaches¹¹ can obtain results from incomplete computations. In the milestone approach, partial results are saved at the end of each distinct phase of the computation. Each imprecise result saved is closer to the precise form. If the deadline arrives, the computation is interrupted, and the result saved in the last imprecise computation is used. In the sieve approach, the non-critical parts or states of computations are designed using the concept of sieve functions. The sieves are not critical to the success of the computation. Instead, they are useful for improving or refining the precision of the results and can be discarded if needed to meet the deadlines. A scheduling algorithm needs to decide which of the parts should be executed on the available time. Other methods include optimization techniques⁴¹ to minimize the errors from discarding low-criticality (or optional) tasks subject to schedulability constraints.

Different priority-driven algorithms have been combined with imprecise computation to improve their performance under transient overloads.¹ Imprecise computation also has been applied for scheduling algorithms to improve the performance under transient overloads. The results presented in Stavrinides and Karatza⁴² showed that EDF scheduling policies combined with imprecise computation improve the overall system performance under heavy workload when compared with EDF without imprecise computation.

Heuristic algorithms based on EDF are also proposed in Huang et al.⁴³ for scheduling periodic tasks. The authors show how to improve non-preemptive real-time scheduling on a single processor by applying imprecise computation. The first proposed algorithm can decide if each computation of a set of computations can be executed in accurate or imprecise mode. If the algorithm guarantees that the computation will meet its deadline, then the computation is executed in accurate mode. Otherwise, the computation is executed in imprecise mode. The authors presented methods to maximize the execution of computations in accurate mode. These methods change computations from imprecise to the accurate mode at runtime. The second algorithm is an online heuristic scheduler. It decides if each computation starts in accurate or imprecise mode. Online schedulability tests keep checking the computations to decide if they could be adjusted to the accuracy mode.

The advantages of EDF to deal with high workloads in real-time systems are also demonstrated in Guo et al.⁴⁴ The results showed that EDF algorithms with imprecise

computation could improve the energy efficiency for large-scale systems by providing a strategy to enhance QoS for real-time systems. Esmaili et al.⁴⁵ proposed a heuristic for scheduling imprecise computations using directed acyclic task graphs to maximize the QoS under energy-constrained computing devices. The QoS issues of real-time applications under imprecise computation are also addressed on the cloud systems. A scheduling heuristic based on imprecise computation for real-time applications of the heterogeneous cloud is proposed in Stavrinides and Karatza.⁴⁶ This solution guarantees that the applications meet their deadlines, and it works to minimize the global execution time. A generalized weakly hard analysis of real-time on uniprocessor systems for limiting the number of deadline misses is proposed in Pazzaglia et al.⁴⁷

Our solution is based on the principle of the EDF scheduling, priority-driven, and imprecise computation with the mandatory-first approach to integrate I-DEVS with schedulability analysis approaches. In Section 3, we introduce the I-DEVS schedulability model based on the definition of WCET, and in Section 4 we present the I-DEVS schedulability formulation based on the definition of WRT. Section 2.3 introduces the I-DEVS main definitions.

2.3. Imprecise-DEVS

The DEVS formalism⁷ was defined for modeling and simulating discrete event systems. A DEVS model is built by compositing basic atomic and coupled models for modeling real systems. Formal M&S techniques provide efficient design and verification of these systems. Atomic models can be composed by building a coupled model.

DEVS was extended in Hong et al.⁴⁸ to support real-time systems, providing the capability to simulate real-time models. This extension, named here as Real-Time DEVS, allows estimating the timeliness constraints while simulating the system and it opens perspectives for simulation-driven development of real-time systems. For supporting the simulation of timing constraints, Real-Time DEVS defines the concept of time interval function, executable activities, and state of an activity. For instead, if the time interval (ti) of an activity x in a given state s is defined by $ti(s, x)$ must be finished before the end of the $ti(s)$. It means that the end of the time interval is seen as the time upper bound to execute the activity.

A new definition to extend DEVS to provide real-time simulation capability, called RT DEVS, was proposed in Moallemi and Wainer.⁴⁹ RT DEVS assigns a deadline to each output in the atomic component, and it verifies the deadline when the associated output is produced. Unlike Real-Time DEVS defined in Hong et al.,⁴⁸ RT DEVS

applies minor modifications to DEVS, allowing for easy reuse of the previous models.

The atomic component of RT DEVS is formally defined as follows:

$$AMRT = \langle X^b, Y^b, S, \delta_{ext}, \delta_{int}, d_{con}, \lambda, ta, d \rangle$$

where

X^b is a bag of inputs

Y^b is a bag of outputs

S : is the set of sequential states

$\delta_{ext}: Q \times X^b \rightarrow S$, is the external state transition function

$\delta_{int}: S \rightarrow S$, is the internal state transition function

$\delta_{con}: Q \times X^b \rightarrow S$, is the confluent transition function

$\lambda: S \rightarrow Y^b$, is the output function

$ta: S \rightarrow R^+_{0,\infty}$, is the time advance function (which is tied to physical clock of the system); with

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ the set of total states.

$d: S \rightarrow R^+_{0,\infty}$ is the relative deadline of each state for output production. The deadline starts at the end of the associated state when the output function is invoked to produce an output (i.e., considered the release time of the output task). The deadline is allocated to each output generated by the output function. Management of the deadline is done by the time advance function.

Coupled models define the connections between basic atomic and coupled models and are defined as following (as in DEVS formal specifications):

$$CM = \langle X^b, Y^b, D, EIC, EOC, IC \rangle$$

X^b is a bag of inputs

Y^b is a bag of outputs

D is an index to the component

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.

Building on RT DEVS, Moallemi and Wainer¹³ introduced Imprecise-DEVS (I-DEVS), which combines imprecise computation with DEVS and opened new perspectives

toward combining real-time modeling methods with modeling and simulation environments.

Imprecise DEVS (I-DEVS) uses RT DEVS definition and adds a mandatory or optional condition for each state, as follows:

$$AM = \langle X^b, Y^b, S, \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta, d \rangle$$

where

$X^b, Y^b, \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta,$ and d are the same as in RT DEVS,

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

The states of the atomic model are categorized as mandatory and optional. A mandatory state will have a mandatory output function (represented as an output task), and an optional state will produce an optional output task. Although the I-DEVS definitions can be used to study systems with enough to manage the mandatory and optional computations, schedulability analysis requires a known set of computations at each given 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.⁵⁰

3. I-DEVS scheduling and the schedulability model

This section introduces the assumptions considered for scheduling and schedulability analysis of real-time computations. The idea is to improve the predictability and feasibility of I-DEVS scheduling.

In our definitions, a real-time computation C_i is defined by $C_i = \langle r_i, c_i, w_i, d(s_i) \rangle$ where:

- r_i is the release time
- $c_i \in \{\text{mandatory}, \text{optional}\}$
- w_i is the worst-case execution time
- $d(s_i)$ is the relative deadline.

Then, a set of computations K is defined by $K = \{C_1, C_2, \dots, C_n\}$.

The variable w_i denotes the WCET of the computation C_i and it provides information about the worst possible execution time of C_i before running it. Using the WCET allows the scheduling algorithm to ensure that no real-time constraint is missed by assuming that every computation always runs according to its WCET.⁵¹ We assume that the w_i of each computation is specified correctly by the systems engineers at the project requirements phase.

According to the I-DEVS definition,¹³ the computation release time (r_i) of an input to the model is equal to the arrival time. According to the formal semantics of DEVS atomic models, the computation release time of the output

(λ) and internal transition computation (I) functions are considered to execute together (λI). When this happens, the state $s \in S$ might change. The execution of these functions is triggered by the time advance function $ta(s)$. As each computation C_i is performed on a state s where $s \in S$, and $S = \{s, c\}$, and $c \in (\text{mandatory}, \text{optional})$, each computation can be mandatory or optional according to its current state s .

The relative deadline is specified by $d(s)$, as seen in the I-DEVS atomic model definition. The relative deadline defines the time between the beginning of the state S and the time limit to complete the λI computation. A single computation C_i is said schedulable if its relative deadline $d(s_i)$ is equal to or greater than the time advance plus the computation time of the λI , which is defined by w_i . Then, $ta(s) + w_i \leq d(s_i)$.

If λI is not schedulable and it is defined as optional, then the output (λ) could be discarded. The internal transition computation (I) is not discarded as we need to perform an internal transition function to the next state. For multiple consecutive optional λI computations, only the I component of the last computation is required to perform the internal transition. Discarding the optional computations can save time to execute the mandatory computations.

In our proposed method, the scheduling of the mandatory and optional computations is based on the EDF algorithm. We consider a single processor system. The priority of each computation is defined according to its relative deadline $d(s_i)$. The computations with earliest deadlines have higher priority, and the mandatory computations receive a higher priority than the optional computations independently of their $d(s_i)$. The system is considered to fail when one or more mandatory computations are not schedulable. The scheduling algorithm assigns priorities dynamically to $C_i \in K$, where $K = (C_1, C_2, \dots, C_n)$, as follows:

If $C_i \wedge C_j \in K$
then
if $c_i = \text{mandatory}$ and $c_j = \text{optional}$
 C_i is higher priority than C_j // even if $d(s_i) > d(s_j)$
else if $c_{i,j} = \text{mandatory}$ and $d(s_i) < d(s_j)$ then
 C_i is higher priority than C_j
else if $c_{i,j} = \text{optional}$ and $d(s_i) < d(s_j)$ then
 C_i is higher priority than C_j .

The EDF scheduler assigns priorities at runtime whenever there are new arrivals of new computations at a given time. We assume sporadic time arrivals of computations, which are like the aperiodic arrivals,²⁷ but with a minimum inter-arrival time denoted by T_i . If the arrival time of a computation C_i is r_i , then the next computation arrives on or after $r_i + T_i$, and $d(s_i) - r_i \leq T_i$. As we can study schedulability considering that the minimum inter-arrival

time can be used to define the period P_i of a computation C_i , we assume that $P_i = T_i$.

As discussed in Nasri et al.,⁵² schedulability analysis for systems with non-deterministic arrival times is decidable for non-preemptive schedulers, and it could be undecidable for systems with preemptive schedulers. Moreover, preemption could increase memory requirements, reducing the number of feasible schedules.^{31,51} Therefore, we decided to use a non-preemptive scheduler (extensions for preemptive schedulers will be studied in future work).

4. I-DEVS schedulability formulation

The schedulability model defined in the previous section was used to specify a schedulability analysis method for I-DEVS based on the WRT of the computations. The WRT allows verifying if each computation which belongs to a known set of computations (ready to be executed) is schedulable or not by performing the schedulability testing.

This section presents a definition of a *set of computations*, a method for determining the WRT of each computation and how it works with the relative deadline for schedulability analysis. Following, we present a technique to implement a schedulability test according to the EDF scheduling algorithm. Finally, we present an adaptation of the DEVS execution algorithms to manage the set of computations and to execute the schedulability tests on the coupled models, as well as discussing a method to flatten of DEVS hierarchical structure.

4.1. The set of computations and schedulability analysis specification

The I-DEVS basic components are modeled hierarchically. However, as each component knows only its own computations, the hierarchical structure needs to be flattened. Flattening of the model hierarchy^{53,54} is a method that transforms the hierarchical structure of a coupled model to a flat structure of depth one. Therefore, all computations are managed by the topmost coupled component. Then, if M_i is an I-DEVS atomic component and the set of components is defined as $\{M_i | I \in D\}$ where each i in D is a component, then K is the set of computations C_i of all i in D where $K = \{C_1, C_2, \dots, C_n\}$. And C_i is the computation of the component M_i at a given time. All the arrived computations at a given time define the set K .

The schedulability analysis^{1,27} is performed for all the computations i of the set K before releasing them to be executed. We need a known set K to evaluate schedulability. If there is a set K of computations at a given time, then:

$$C_1, C_2, \dots, C_n \in K \\ \{(s, e, d(s)) | s \in S, 0 \leq e \leq ta(s) \text{ where } d(s) \geq ta(s)\}$$

for each $C_i \in K$, C_i is schedulable if:

$$w_i + \sum_{C_j=1 \in (hp(C_i))}^{|K|} w_j + e_i \leq d(s_i) \quad (1)$$

where

$C_i \wedge C_j \in K$

e_i is the elapsed time of the s_i

w_i and w_j are the WCET of the computations C_i and C_j , respectively

$hp(C_i)$ is all C_j computations of K with higher priority (hp) than C_i

$d(s_i)$ is the relative deadline of C_i from the beginning of the state s_i .

Equation (1)¹ verifies if the WRT³⁸ of each computation C_i plus its elapsed time is no larger than its deadline. The WRT of a C_i is defined by the sum of the w_j of all C_j with higher priority (hp) than C_i plus the w_i . For each $C_i \in K$, if $WRT(C_i) + e_i \leq d(s_i)$, then C_i is assumed schedulable. Otherwise, if $WRT(C_i) + e_i > d(s_i)$, then C_i is assumed not schedulable. If $c_i = optional$, then it is discarded, and the results are considered imprecise. Mandatory computations are always performed.

As the results of the schedulability tests are assumed to be feasible, all such schedulable computations will meet their deadline. The schedulability analysis of each $C_i \in K$ is accomplished by their WRT that include the WCET of all computations with priority higher than C_i .

4.2. Schedulability testing formulation

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:²²

$$U_i = \frac{w_i}{P_i} \quad (2)$$

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 ¹⁵ is used as the period P_i for schedulability test. The sporadic computations behave like periodic computations with period T and deadline $d(s)$, and the minimum interval can be settled to be equal to the relative deadline.²⁷

Therefore, we assumed $T_i = d(s_i)$ and $P_i = T_i$. As the relative deadline is defined according to the time of the last transition and it can change for distinct $C_i \in K$, the elapsed time e cannot be considered as part of the period to analyze the schedulability. Then, the minimum interval of all $C_i \in K$ is given by the following equation:

$$P_i = d(s_i) - e_i \quad (3)$$

The largest minimum interval (P_i) between the arrivals among all the computations of the set K of computations at a given current time is settled as the current P_i for performing the schedulability tests of the set K of computations. Equation (3) shows how P_i is calculated. Therefore, for all $C_i \in K$, let $P_j = \{ \max_i [P_i] | P_i = d(s_i) - e_i \}$. The largest minimum interval is adopted to avoid a pessimistic estimation⁵² on the schedulability tests without losing feasibility.

Therefore, by calculating the utilization factor (Equation (2)) based on w_i and P_i , a set K of computations could be considered schedulable if $U \leq 1$ for all $U_i (C_i \in K)$ according to the equation:

$$\sum_{i=1}^n U_i \leq 1 \quad (4)$$

However, the utilization factor is seen as a non-exact test to verify whether the set K is entirely schedulable or not. If the schedulability test is negative, K is not entirely schedulable. If the schedulability test is non-negative, it does not guarantee that all $C_i \in K$ are schedulable. Moreover, it does not help the scheduler to identify which of the optional computations could be discarded in case of transient overloads.

Different to the utilization factor, the WRT analysis presented by Liu¹ allows the evaluation of the exact schedulability of computations for EDF. Given a set K of computations, it permits to verify if each $C_i \in K$ is schedulable or not. In addition, it permits to identify the optional computations that could be discarded to save time for the mandatory computations. This test is based on the following equations:

$$R_{C_i}^m = w_i \quad (5)$$

Assuming the set K of computations with one computation C_i , and the worst-case execution time of C_i is defined by w_i , the WRT R of C_i ($R_{C_i}^m$ where $m = 0$) is equal to the w_i from its release time given by r_i .

Equation (5) is applied to the schedulability test where the set K has one computation. When the set K has more than one computation, Equation (5) is applied to the first iteration for calculating R and m starts at 0:

$$R_{C_i}^{m+1} = w_i + \sum_{j \in hp(C_i)} \frac{R_j^m}{P_j} \cdot w_j \quad (6)$$

where

R is the interval between r_i and the end of the execution of the C_i (WRT)

P is the period

w is the WCET.

Assuming the set $K = (C_i, C_{i+n})$ where $n > 0$, and the worst-case execution time of (C_i, C_{i+n}) is defined by (w_i, w_{i+n}) , the WRT R_{C_i} of each $C_i \in K$ is equal to the w_i plus the worst-case execution time of all computations with higher priority than C_i , represented by w_j in Equation (6), into the K .

Equation (6) is applied after Equation (5) iteratively until $R_{C_i}^m = R_{C_i}^{m+1}$ to reach the WRT of each computation C_i into the K . Each iteration is identified by m . The result is the maximum (or worst) response time (R_{C_i}) of the computation C_i . Equations (5) and (6) for calculating the WRT of real-time computations are demonstrated in Liu¹ and Burns and Wellings.²⁷

According to the I-DEVS definition introduced in section 2.3, $d(s_i)$ is the relative deadline of C_i from the beginning of the state s_i , and $S: \{(s, c) | s \in Z_0^+ \text{ and } c \in \{\text{mandatory} | \text{optional}\}\}$. As the R_{C_i} is calculated from the release time of C_i , defined by r_i , then the elapsed time (e_i) from the initial of the state s_i until r_i is applied on the schedulability test and C_i is assumed schedulable only if the condition of Equation (7) is satisfied:

$$R_{C_i} + e_i \leq d(s_i) \quad (7)$$

Following the assigning priority method introduced in section 3, the mandatory computations always have higher priority than optional computations, independent of their deadlines. If the condition of Equation (7) is false for one or more C_i , where $c_i = \text{mandatory}$, then the scheduling of $K = (C_i, C_{i+n})$ is not viable. Optional computations are discarded only when the condition defined on Equation (7) is false.

4.3. Extending the I-DEVS coordinator algorithms

This section describes the algorithms, which are associated with the coupled models that integrate the functionalities for schedulability analysis. We extended the I-DEVS algorithms (which are based on RTDEVS and P-DEVS⁵⁵) to perform the schedulability testing procedures. Execution processes (Engines) associated with atomic models were also adapted.

The coordinator algorithms manage the messages exchange among Engines. A simulation starts, as defined on I-DEVS, by sending an initialization message to all Engines.

Whenever there is an external input, instead of routing it through an external message (X) to the destination

ALGORITHM 1: Modified Engine algorithm

```

Receive X msg (s, e, x)
    push x in the  $Q_{ext}$ 
end external
Receive * msg (s)
    if (internal event)
        Run  $\delta_{int}$  function
        if (external input in  $Q_{ext}$ )
            Run  $\delta_{ext}$  function
        elseif (both external and internal events)
            Run  $\delta_{con}$  function
        endif
    endif
    send done msg
end internal
Receive @ msg (s)
    Run  $\lambda$  function
    Send y msg
    Send done msg
end collect

```

Engine, as happen in the I-DEVS algorithms, we cache it in the set K of computations (called *SyncSet* into the algorithms). The coordinator performs the schedulability testing over the *SyncSet* and the optional computations that will not meet their deadlines are discarded from the *SyncSet*.

Whenever there is an internal event (λI), instead of sending collect (@) and internal (*) messages to the target Engine, we cache the attributes (as defined in Section 3) of the computation associated with this event in the *SyncSet* in order to perform the schedulability testing. If this computation will meet the deadline, then @ and * messages are sent to the respective Engine. Otherwise, only the * message is sent to execute δ_{int} . When the @ message is canceled, the coordinator does not expect to receive done message. Whenever a sequence of optional computations of an atomic model is discarded, only the * message of the last event must be sent to the Engine to execute the δ_{int} . This strategy will reduce the number of messages between the coordinator and the Engine and the number of δ_{int} executions. In this way, the Engine is no longer responsible for controlling the imprecise computation issues.

ALGORITHM 1 shows the modified version of the Engine. The original I-DEVS Engine functions control the imprecise computation by including the following condition:¹³ *if (state is optional AND $ta(s) < now$ OR $d(s) < now$)*. This condition is true when an optional computation is to be serviced later than its release time (given by $ta(s)$ in the original algorithm) or when the computation does not meet the deadline. Hence, whenever this condition is true, its output will be discarded. This condition was

ALGORITHM 2: Coordinator algorithm: internal message

```

while  $t \neq \infty$ 
    if  $t_L \leq t \leq t_N$ 
        for all  $q \in bag$ 
            for all receivers of  $q, j \in I_{self}$ 
                 $q := z_{self,j}(q)$ 
                cache  $j(q, t)$  in the SyncSet
            endfor
        endfor
        for all  $j(q, t)$  in SyncSet
            if  $s_j$  is optional and  $j(q, t)$  is schedulable
                send ( $q, t$ ) to  $j$ 
            else
                discard  $j(q, t)$  of the SyncSet
                send (*,  $t$ ) to  $j$  // state transition
            endif
            if  $s_j$  is mandatory
                send ( $q, t$ ) to  $j$ 
            endif
        endfor
        empty bag
        for all  $i$  in the SyncSet
            send (*,  $t$ ) to  $i$ 
        endfor
        wait until all (done,  $t_N$ )'s are received
         $t_L := t$ 
         $t_N :=$  minimum of components'  $t_N$ 's
        clear the SyncSet
    else raise an error
    endif
end while

```

removed from our modified version because the schedulability testing is performed by the topmost coordinator and only the schedulable computations are sent to the Engine. Hence, the functions for controlling the imprecise computation are not needed in the Engine.

ALGORITHM 2 illustrates the topmost coordinator algorithm when it receives a * message where t_L is the time of the last transition, t_N is the time of the next transition, and $t_L \leq t \leq t_N$. If the coordinator receives an internal message * to be sent to an Engine and there are inputs $q \in Bag$, instead of sending them immediately to the child, the $j(q, t)$ are cached into the *SyncSet*. Following, the schedulability test is performed over each one of the computations cached in the *SyncSet*. The tested computations C_i that will meet their deadlines (according to the schedulability test) are sent to the children. Otherwise, they are discarded from the *SyncSet*. The sent messages are followed by the internal message *.

Mandatory computations are always sent to the children. When an Engine does not receive the message due to the discarding; time is saved for mandatory computations. The atomic model executes δ_{int} in response to a *

ALGORITHM 3: Coordinator algorithm: collect message

```

while  $t \neq \infty$ 
  if  $t = t_N$  then
     $t_L := t$ 
    for all imminent child processors  $i$  with
      minimum  $t_N$ 
      cache  $i$  in the SyncSet
    endfor
    for all  $i$  in SyncSet
      if  $s_i$  is optional and  $i(\lambda, t)$  is schedulable on
        SyncSet based on the  $WRT(i(\lambda, t))$ 
        send ( $@, t$ ) to child  $i$ 
      else
        discard  $i$  from the SyncSet
      endif
      if  $s_i$  is mandatory
        send ( $@, t$ ) to  $i$ 
      endif
    endfor
    wait until (done,  $t$ )'s have been received from
      all imminent processors
  else raise error
endif
end while

```

message and returns its next internal event time by a *done* message.

Similar changes were made in the coordinator algorithms when receiving a collect ($@$) and output y messages from the children. As illustrated in ALGORITHM 3, whenever the coordinator receives a $@$ message to be sent to the child i , $i(@)$ is also cached into the *SyncSet*. Afterward, the coordinator applies the schedulability test to verify if the computations of $i(@)$ at the time t for all child $i \in \text{SyncSet}$ will meet the deadline. If discarded, the $@$ message is not sent to the Engine. If the $@$ message is sent to the target Engine, the Engine 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 , as illustrated in ALGORITHM 4, 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 its receiving Engines, the coordinator verifies if the state of the j is optional and if $j(q, t)$ is schedulable among all $j \in \text{SyncSet}$. If j is schedulable then $j(q, t)$ is sent to its receiving Engine. If not schedulable, the coordinator discards $j(q, t)$ from *SyncSet* and the message q will not be sent to its child.

The messages to trigger mandatory computations are always sent to the Engines by all the coordinators. As the schedulability tests and scheduling are performed by the coordinator, the Engines were not changed.

ALGORITHM 4: Coordinator algorithm: output message

```

when a ( $y, t$ ) message is received from child  $i$ 
  for all influences,  $j$  of child  $i$ 
     $q := z_{ij}(y)$ 
    cache  $j(q, t)$  in the SyncSet
  endfor
  for all  $j(q, t)$  in SyncSet
    if  $s_j$  is optional and  $j(q, t)$  is schedulable on SyncSet
      based on the  $WRT(j(q, t))$ 
      send ( $q, t$ ) to  $j$ 
    else
      discard  $j(q, t)$  from the SyncSet
    endif
    if  $s_j$  is mandatory
      send ( $q, t$ ) to  $j$ 
    endif
  endfor
endwhen

```

4.4. DEVS hierarchy

The set K of computations must include all computations C_i of the model at a given time to execute the schedulability test. However, as each coordinator manages the computations of its own Engines only, the hierarchical structure must be flattened to keep all computations in one single set K . The Flattened coordinator strategy^{53,54} transforms a hierarchical structure of a coupled model into a flat structure with depth one by eliminating intermediate coordinators. In addition, the direct messaging communications between the Flattened coordinator and the Engines reduce overhead and improve stability. The transformation must preserve the original port linkage relationship among atomic models. The *SyncSet* structure in the coordinator algorithm implements the definition of the set K of computations.

Therefore, the schedulability algorithms are performed based on the *SyncSet* at the topmost coordinator. There are different algorithms to transform the hierarchical structure of the model into a flattened structure by eliminating coordinators and transforming the hierarchical coupled model into a coupled model of depth one.

Figure 1 illustrates the model transformation from a hierarchical structure to a flattened structure with depth one. It shows a simple graph to identify the multiply *SyncSet* due to the hierarchical structure and a single *SyncSet* after flattening. The structure illustrated in Figure 1(a) does not permit to perform the schedulability tests for the entire model. After flattening the coordinator's structures, as shown in Figure 1(b), there is one unique *SyncSet* and the schedulability tests and scheduling can be performed.

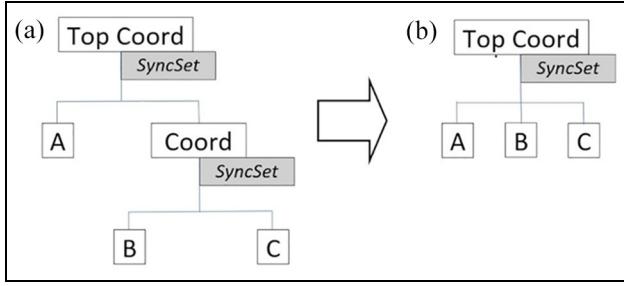


Figure 1. From a hierarchical structure to a flattened structure with depth one: (a) *SyncSet* in a hierarchical structure and (b) *SyncSet* in a flattened structure.

5. Schedulability test scenario and results

This section shows scenarios of schedulability testing based on an I-DEVS model including three atomic models. This I-DEVS model definition is presented in Moallemi and Wainer,¹³ and its original hierarchical structure was flattened. The scenarios explore four schedulability situations where each testing step is discussed in detail. The three atomic models A , B , and C are coupled into the top-most coordinator. Figure 2 shows the I-DEVS graph of each atomic model.

Figure 2(a) shows a flattened I-DEVS model where three atomic models are coupled into the Top model. The atomic models A , B , and C are connected by *input/output* ports. Figure 2(b)–(d) shows a graphic representation of the atomic models. The continuous lines indicate external transitions and dashed lines indicate internal transitions. The graphs show the state identification, the time advance, the relative deadline, and if the state is mandatory or optional for each state of each atomic model. For instance, the state $A2$ of the atomic model A (Figure 2(b)) is mandatory (M), its $ta(s) = 1t$ and its $d(s) = 4t$. Therefore, the release time of the state $A2$ (r_{A2}) is given by $ta(A2)$, hence $r_{A2} = 1$, and the output $y2a$ must be produced until the deadline of the state $A2$ ($d(s_{A2}) = 4$). The flattened model (Figure 2(a)) works as follows: it is initially in state $A1$ ($ta(A1) = inf$) until an input xa is received on port InA when the external transition changes the states to $A2$. After $1t$ it produces the output $y2a$ and transitions to $A3$. Similar behavior can be seen in the other states of A and in the states of B and C .

Figure 3 shows an analytical specification for the atomic models A , B , and C according to the diagrams shown in Figure 2. Each box specifies the ports, states, transitions, release time, and the relative deadline for each state. These specifications are used to represent the behavior of the schedulability testing scenario presented in this section. The following figures represent the computations by X (input), λ (output), and I (internal transition computation). Boxes with gray background represent executed

computations. The first line shows the current time t . The release time and the deadline of the computations are represented by r_n and d_n , respectively.

Figure 4 shows a scenario where computations of the atomic models A , B , and C are performed. First, an input X enters the system at time zero of the model A . Assuming the X takes $1t$ at time 1, the atomic model A moves from the initial state $A1$ to $A2$. As $r_{A2} = 1$ (given by the $ta(A2)$ in Figure 3), the computation C_{A2} ($A(\lambda 2I 2)$) (Figure 4) is executed at time 2 producing the output $y2a$ (illustrated in Figure 2(b)) and the internal transition from the state $A2$ to $A3$. The output $y2a$ is translated to an input for B (Figure 2), and the computation X on B is performed after $A(\lambda 2I 2)$ causing the internal transition on B from $B1$ to $B2$. The next steps to execute the computations are defined on the configuration presented in Figures 2 and 3. From now on, a computation $A(\lambda nIn)$ will be named as C_{An} and X will be named as C_X .

The schedulability test is performed whenever a new computation is ready to be executed. If there are multiple computations ready to be executed at the current time t , they all belong to the set K for testing purposes.

Figure 4 demonstrates a scenario where the release time of the computation C_{A2} is given by $r_{A2} = 1$ from the end of the computation C_X , and only C_{A2} is in the set $K = \{C_{A2}\}$. Considering that the current time t is 1:

If $K = \{C_{A2}\}$ and:

$r_{A2} = 1$ (from the initial of the state on time 1)

$c_{A2} = \text{mandatory}$

$w_{A2} = 2$

$d(s_{A2}) = 4$

$e_{A2} = 1$ (the elapsed time from the last transition of the atomic model A is 1)

The release time of C_{A2} is 1 ($r_{A2} = 1$), C_{A2} is mandatory, the WCET of C_{A2} is 2 ($w_{A2} = 2$), its relative deadline ($d(s_{A2})$) is 4, and the elapsed time from the last transition of the atomic model A is 1 ($e_{A2} = 1$). Because the set of computations K has one computation at time 1 (Figure 4), only Equation (5) is applied.

Then, applying Equation (5) to the computation C_{A2} :

$$R_{A2}^0 = w_{A2}$$

$$R_{A2}^0 = 2$$

After calculating the worst execution time where $R_{A2}^0 = 2$, and considering that $r_{A2} = 1$, and $d(s_{A2}) = 4$, then by applying Equation (7) where $R_{A2}^0 + e_{A2} \leq d(s_{A2})$, the result $2 + 1 \leq 4$ is true, and C_{A2} is schedulable.

Note that only Equation (5) was applied because the set K of computations has only the computation $A2$ at time $t = 1$, and the WRT of the computation $A2$ ($R_{A2}^0 = 2$) is equal to its WCET ($w_{A2} = 2$).

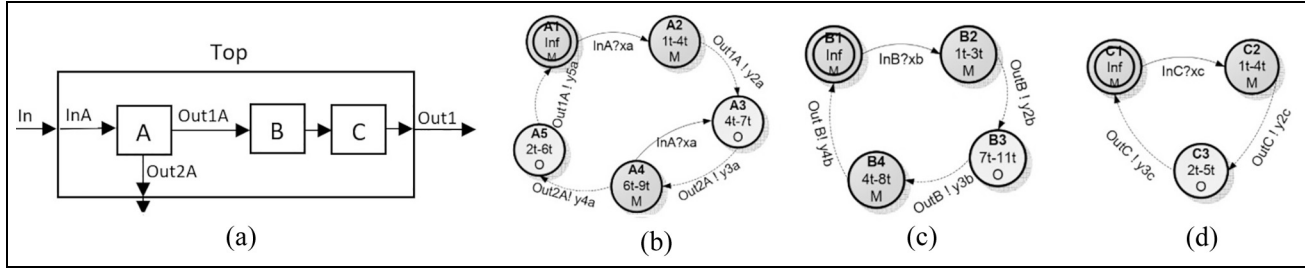


Figure 2. DEVS graph of the atomic models: (a) DEVS flattened model, (b) atomic model A, (c) atomic model B, and (d) atomic model C.

Source: Moallemi and Wainer.¹³

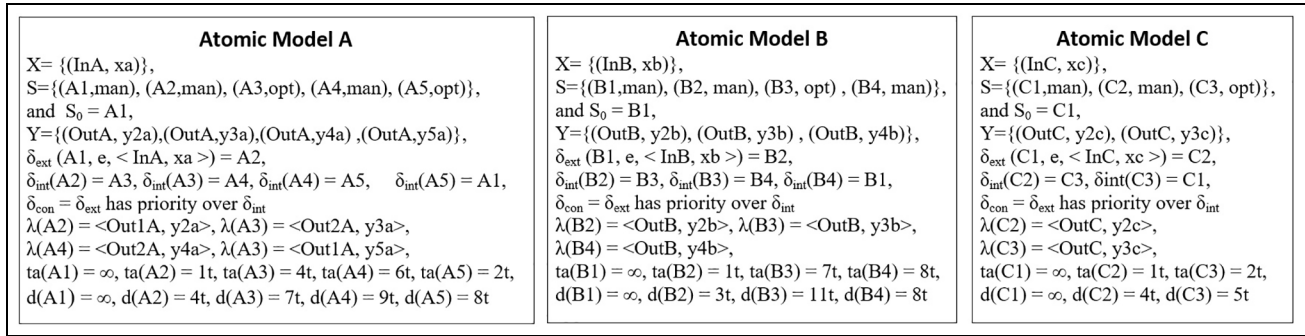


Figure 3. Configuration of the atomic models A, B, and C.

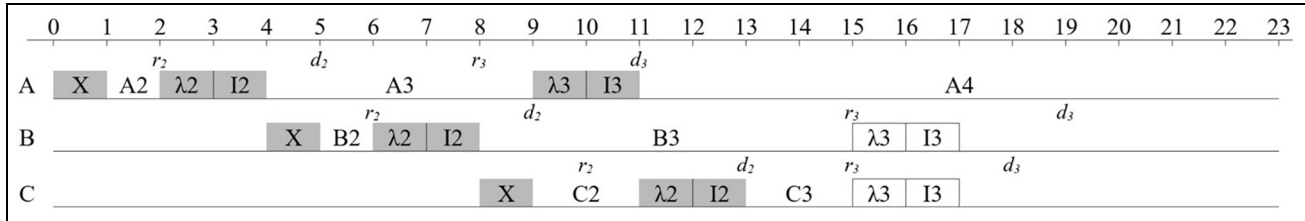


Figure 4. Schedulability test scenario: $K = \{C_{B3}, C_{C3}\}$.

Figure 4 also shows an overload scenario where the computations C_{B3} and C_{C3} are in the set of computations K . Considering that the current time t is 15:

If $K = \{C_{B3}, C_{C3}\}$ and:

$$r_{B3} = 7 \text{ (from time 8)} \wedge r_{C3} = 2 \text{ (from time 13)}$$

$$c_{B3} = \text{optional} \wedge c_{C3} = \text{optional}$$

$$w_{B3} = 2 \wedge w_{C3} = 2$$

$$d(s_{B3}) = 11 \wedge d(s_{C3}) = 5$$

$$e_{B3} = 7 \wedge e_{C3} = 2$$

Both computations are optional. The deadline of C_{B3} is at time 19 (as shown in Figure 4), and it is given by its relative deadline ($d(s_{B3}) = 11$) from the end of the last transition at time 8. The deadline of C_{C3} is at time 18, and it is given by its relative deadline ($d(s_{C3}) = 5$) from the

end of the last transition at time 13. Because the deadline of C_{C3} is earlier than the deadline of C_{B3} , the computation C_{C3} is higher priority than C_{B3} . Therefore, C_{C3} executes first.

Then, applying Equation (5) to the computation C_{C3} :

$$R_{C3}^0 = w_{C3}$$

$$R_{C3}^0 = 2$$

Applying Equation (7), $2 + 2 \leq 5$ is true, and C_{C3} is schedulable.

Because C_{C3} is higher priority than C_{B3} ($C3 \in hp(B3)$), C_{B3} executes after C_{C3} . The WCET of C_{C3} ($w_{C3} = 2$) is considered to analyze the schedulability of C_{B3} by applying Equation (5) at first and following Equation (6)

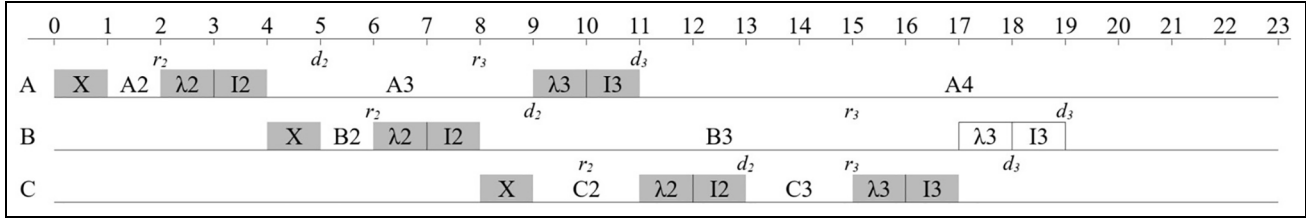


Figure 5. Schedulability test scenario: $K = \{C_{B3}\}$.

iteratively until $R_{C_i}^m = R_{C_i}^{m+1}$, where P_{C3} is given by $d(s_{B3}) - e_{B3}$ according to Equation (3).

Applying Equations (5) and (6) to the computation C_{B3} with $K = \{C_{B3}, C_{C3}\}$:

$$\begin{aligned} R_{B3}^0 &= w_{B3} = 2 \\ R_{B3}^1 &= w_{B3} + \sum_{C3 \in hp(B3)} \frac{R^0}{P_{C3}} \cdot w_{C3} = 2 + \frac{2}{4} \cdot 2 = 4 \\ R_{B3}^2 &= w_{B3} + \sum_{C3 \in hp(B3)} \frac{R^1}{P_{C3}} \cdot w_{C3} = 2 + \frac{4}{4} \cdot 2 = 4 \end{aligned}$$

As $R_{B3}^1 = R_{B3}^2$ at the second iteration, and applying Equation (7), $4 + 7 \leq 11$ is true, and C_{B3} is also schedulable.

Figure 5 shows that after the end of the execution of C_{C3} , the computation C_{B3} can be executed before its deadline at time 19. As $d(s_{C3})$ is earlier than $d(s_{B3})$, C_{C3} is higher priority than C_{B3} (according to the EDF scheduling algorithm). Therefore, w_{B3} is not considered to assess the schedulability of C_{C3} (only Equation (5) is applied). Otherwise, the w_{C3} must be considered to evaluate the schedulability of C_{B3} and Equations (5) and (6) must be applied. In this test, C_{B3} also is schedulable because it will meet its deadline even if executed after C_{C3} . This scenario can be seen in Figure 5 where the current time $t = 17$.

Figure 6 shows a new computation C_{A4} ready to be executed before executing C_{B3} . As $r_{A4} = 6$ from the initial of the state $A4$ (configuration in Figure 3), the execution time of the computation C_{A4} is settled to time $t = 17$. Therefore, the set K is defined by $K = \{C_{B3}, C_{A4}\}$ at $t = 17$ for evaluating the schedulability.

Figure 6 also shows another overload scenario where the computations C_{B3} and C_{A4} are in the set of computations K . Considering that the current time t is 17:

$$\begin{aligned} &\text{If } K = \{C_{B3}, C_{A4}\} \text{ and:} \\ r_{B3} &= 7 \text{ (from time 8)} \wedge r_{A4} = 6 \text{ (from time 11)} \\ c_{B3} &= \text{optional} \wedge c_{A4} = \text{mandatory} \\ w_{B3} &= 2 \wedge w_{A4} = 2 \\ d(s_{B3}) &= 11 \wedge d(s_{A4}) = 9 \\ e_{B3} &= 9 \wedge e_{A4} = 6 \end{aligned}$$

The deadline of C_{B3} is at time 19 (as shown in Figure 6), and it is given by its relative deadline ($d(s_{B3}) = 11$) from the end of the last transition at time 8. The deadline of C_{A4} is at time 20, and it is given by its relative deadline ($d(s_{C3}) = 9$) from the end of the last transition at time 11. However, C_{A4} is higher priority than C_{B3} because $c_{A4} = \text{mandatory}$ and $c_{B3} = \text{optional}$ even if the deadline of C_{B3} is earlier than the deadline of C_{A4} . Therefore, according to the schedulability model presented in section 3, C_{A4} executes first.

Then, applying Equation (5) to the computation C_{C3} :

$$\begin{aligned} R_{A4}^0 &= w_{A4} \\ R_{A4}^0 &= 2 \end{aligned}$$

Applying Equation (7), $2 + 6 \leq 9$ is true, and C_{A4} is schedulable.

Because C_{A4} is higher priority than C_{B3} ($A4 \in hp(B3)$), C_{B3} executes after C_{A4} . The WCET of C_{A4} ($w_{A4} = 2$) is considered to analyze the schedulability of C_{B3} by applying Equation (5) at first and following Equation (6) iteratively until $R_i^m = R_i^{m+1}$.

Applying Equations (5) and (6) to the computation C_{B3} as following:

$$\begin{aligned} R_{B3}^0 &= w_{B3} = 2 \\ R_{B3}^1 &= w_{B3} + \sum_{A4 \in hp(B3)} \frac{R^0}{P_{A4}} \cdot w_{A4} = 2 + \frac{2}{3} \cdot 2 = 4 \\ R_{B3}^2 &= w_{B3} + \sum_{A4 \in hp(B3)} \frac{R^1}{P_{A4}} \cdot w_{A4} = 2 + \frac{4}{3} \cdot 2 = 6 \\ R_{B3}^3 &= w_{B3} + \sum_{A4 \in hp(B3)} \frac{R^1}{P_{A4}} \cdot w_{A4} = 2 + \frac{6}{3} \cdot 2 = 6 \end{aligned}$$

As $R_{B3}^2 = R_{B3}^3$ at the third iteration, and applying Equation (7), $6 + 9 \leq 11$ is false, and C_{B3} is not schedulable. Figure 7 shows that after the end of the execution of C_{A4} , the computation C_{B3} cannot be executed before its deadline at time 19.

As C_{A4} is mandatory and C_{B3} is optional, C_{A4} is higher priority than C_{B3} . Therefore, w_{B3} is not considered to

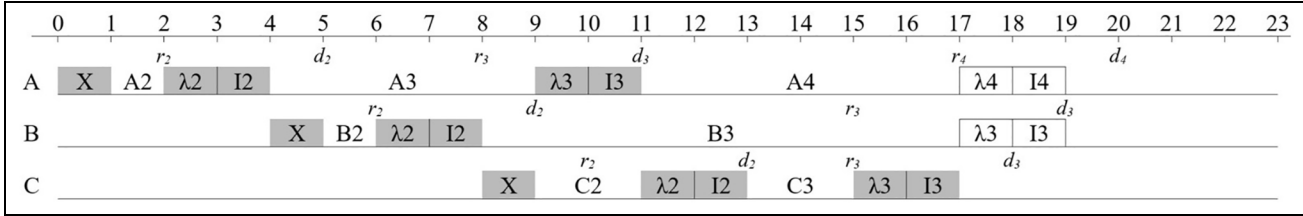


Figure 6. Schedulability test scenario: $K = \{C_{B3}, C_{A4}\}$.

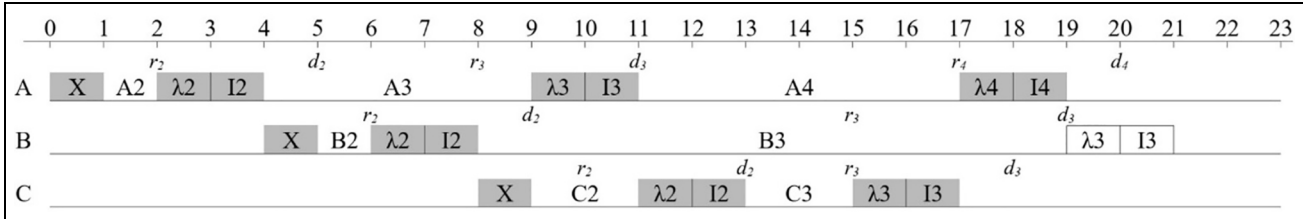


Figure 7. Schedulability test scenario illustrating the execution of C_{A4} .

assess the schedulability of C_{A4} (only Equation (5) is applied). Otherwise, the w_{A4} must be considered to assess the schedulability of C_{B3} and Equations (5) and (6) must be applied. In this test, C_{B3} is not schedulable because it will not meet its deadline, and it is discarded. This scenario can be seen in Figure 7 at the current time $t = 19$. Even if C_{A4} is mandatory, the schedulability test must be executed because the set K could have another mandatory computation.

In summary, we showed how the proposed methods allow the analysis of overloading conditions under I-DEVS formalism, which integrate imprecise computations of real-time systems and DEVS, by schedulability testing on designing phases.

6. Discussion and perspectives

The research results presented above show how to integrate DEVS formal models, imprecise computing, simulation, and schedulability analysis to define and study real-time systems models including cases of transient overloading conditions in the designing phases. Using this solution, the designer configures the atomic models of a real-time systems using I-DEVS atomic models and coupling them to build complex applications. Then, the different DEVS functions are converted into transitions, r_i , c_i , w_i , and $d(s_i)$ for each computation, as illustrated in Figures 2 and 3. Then, we can apply the proposed schedulability test (based on WCET and WRT methods), to evaluate the feasibility of the timeline for the RT system. Our approach allows the designer to keep attention on the modeling aspects, using

simulation and schedulability analysis prior implementation, and considering IC model development. The proposed methods guarantee the timeliness requirements by applying schedulability tests over mandatory and optional computations.

I-DEVS would discard optional computations whenever it is to be serviced later than its release time. For instance, in Figure 4, $A(\lambda3I3)$ was serviced at time $t = 9$ while its release time is at time $t = 8$ ($rA3$) and this computation would be discarded even if there is no overload. On the contrary, introducing schedulability tests allows analyzing the timing constraints at runtime. In Figure 4, the computation $A(\lambda3I3)$ is considered feasible to be scheduled even if it is to be serviced later than its release time. Consequently, the number of executed optional computations can be increased.

The scenarios in section 5 show how our solution analyzes the schedulability of real-time constraints under different overloading conditions. Our solution is based on WRT classical methods that were integrated with I-DEVS. The WRT methods were demonstrated in Liu¹ and in Burns and Wellings.²⁷ The effectiveness of I-DEVS by combining imprecise computation technique and DEVS is demonstrated in Moallemi and Wainer.¹³ The principles of imprecise computation are introduced in Liu et al.⁹ In this work, we demonstrate the feasibility and effectiveness of our solution where a modeling and simulation formalism (I-DEVS) allows analyzing the schedulability of real-time constraints of RTS under overload conditions by simulation and the mapping of models into a real-time task model. Exhaustive experiments for comparing the

performance and optional computations discarding measures of the previous and current approaches are needed in the future.

The error of the optional computation is calculated as the distance between the imprecise result, when the optional computations are executed partially, and the precise result, when the optional computations are executed completely. The maximum error happens when all the optional computations are discarded.¹⁸ To minimize the total error, the scheduler and schedulability test can consider the weight of each computation. The weight is a positive number for measuring the relative importance of the computation to the result where, if z_i is the weight of the computation i , then $0 \geq z_i \leq 1$ and $\sum_{i=1}^n z_i = 1$. Given a schedule of a computation set K where ε_i is the error of each computation i and z_i is the weight of each computation i , the maximum error ($\max_i [z_i, \varepsilon_i]$) or average error can be minimized by scheduling the subset of K with the smallest maximum or average error.

7. Conclusion



We presented an approach to integrate schedulability analysis strategy with I-DEVS 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 test is based on the WRT to verify whether each computation is schedulable or not according to 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 a method based on the *SyncSet* on the coordinator level as the main resource to implement the schedulability tests and the scheduling before triggering the computations on the Engines.

As future work, we plan to study new scheduling and schedulability analysis methods to manage new features. The main interests include the integration of sharing resources in the system models, and the scheduling and schedulability analysis process could implement mutual exclusion functionalities to avoid deadlock situations. Moreover, in our approach, the priority inversion and the weight of the computations are not being considered. The discarding of optional computations unnecessarily can be avoided by managing the priority inversion situations. The weight of the optional computations is useful to improve the accuracy of the results. Analysis considering the definition of error measures of the optional computations will also be considered in the future.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was supported by Federal University of Fronteira Sul and the Brazilian research agency CAPES, process No. 1835-14-9. It was partially funded by NSERC.

ORCID iDs

Braulio A Mello  <https://orcid.org/0000-0003-3430-809X>
Gabriel A Wainer  <https://orcid.org/0000-0003-3366-9184>

References

1. Liu J. *Real-time systems*. 1st ed. London: Pearson, 2000.
2. Felder M and Pezzè M. A formal design notation for real-time systems. *ACM T Softw Eng Meth* 2002; 11: 149–190.
3. Haur I, Béchenec J and Roux O. Formal schedulability analysis based on multi-core RTOS model. In: *29th international conference on real-time networks and systems*, Nantes, 7–9 April 2021, pp. 216–225. Nantes: Association for Computing Machinery.
4. Davis R and Burns A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput Surv* 2011; 43: 1–44.
5. Davis R. A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Rev* 2014; 11: 8–19.
6. Moallemi M and Wainer G. Designing an interface for real-time and embedded DEVS. In: *Spring simulation multiconference (SpringSim '10)*, Orlando, FL, 11–15 April 2010, pp. 1–8. San Diego, CA: Society for Computer Simulation International.
7. Zeigler B. *Multifaceted modelling and discrete event simulation*. 1st ed. London; Orlando, FL: Academic Press Professional, 1984.
8. Zeigler B, Praehofer H and Kim T. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed. San Diego, CA: Academic Press, 2000.
9. Liu J, Shih W, Lin K, et al. Imprecise computations. *P IEEE* 1994; 82: 83–94.
10. Ramanathan P. Graceful degradation in real-time control applications using (m, k)-firm guarantee. In: *IEEE 27th international symposium on fault tolerant computing*, Seattle, WA, 24–27 June 1997, pp. 132–141. New York: IEEE.
11. Lin K, Natarajan S and Liu J. Imprecise results: utilizing partial computations in real-time systems. In: *IEEE 8th real-time systems symposium*, San Jose, CA, 1–3 December 1987, pp. 210–217. New York: IEEE.
12. Liu J, Lin K, Shih W, et al. Algorithms for scheduling imprecise computations. *Computer* 1991; 24: 58–68.
13. Moallemi M and Wainer G. I-DEVS: imprecise real-time and embedded DEVS modeling. In: *Symposium on theory of modeling & simulation: DEVS integrative M&S symposium*, Boston, MA, 3–7 April 2011, pp. 95–102. Boston, MA: The Society for Modeling and Simulation International.

14. Wainer G and Moallemi M. Designing real-time systems using imprecise discrete-event system specifications. *Softw: Pract Exper* 2020; 50: 1327–1344.
15. Zhang F and Burns A. Schedulability analysis for real-time systems with EDF scheduling. *IEEE T Comput* 2009; 58: 1250–1258.
16. Buttazzo G. *Hard real-time computing systems: predictable scheduling algorithms and applications*. 3rd ed. Boston, MA: Springer Publishing Company, 2011.
17. Ripoll I, Crespo A and Mok A. Improvement in feasibility testing for real-time tasks. *Real-Time Syst* 1996; 11: 19–39.
18. Wilhelm R, Engblom J, Ermedahl A, et al. The worst—case execution—time problem—overview of methods and survey of tools. *ACM T Embed Comput S* 2008; 7: 1–53.
19. Alshareef A and Sarjoughian H. Simulation, model checking, and execution of activity models. arXiv:2105.11851v1 [cs.SE], 2021, <https://arxiv.org/abs/2105.11851>
20. Fersman E, Pettersson P and Yi W. Timed automata with asynchronous processes: schedulability and decidability. In: *International conference on tools and algorithms for the construction and analysis of systems*, Grenoble, 8–12 April 2002, pp. 67–82. Berlin: Springer.
21. Saadawi H and Wainer G. Principles of DEVS models verification. *Simul: T Soc Mod Sim* 2013; 89: 41–67.
22. Liu C and Layland J. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J ACM* 1973; 20: 46–61.
23. Kuo T, Chang L, Liu Y, et al. Efficient online schedulability tests for real-time systems. *IEEE T Software Eng* 2003; 29: 734–751.
24. Zhang F and Burns A. Schedulability analysis of EDF-scheduled embedded real-time systems with resource sharing. *ACM T Embed Comput S* 2013; 12: 1–19.
25. Lee H and Choi J. Constraint solving approach to schedulability analysis in real-time systems. *IEEE Access* 2018; 6: 58418–58426.
26. Sha L, Abdelzaher T, Arzén K, et al. Real time scheduling theory: a historical perspective. *Real-Time Syst* 2004; 28: 101–155.
27. Burns A and Wellings A. Implementing analysable hard real-time sporadic tasks in Ada 9X. *ACM Sigada Ada Lett* 1994; XIV: 38–49.
28. Buttazzo G. Rate monotonic vs. EDF: judgment day. *Real-Time Syst* 2005; 29: 5–26.
29. Ramamritham K and Stankovic J. Scheduling algorithms and operating systems support for real-time systems. *P IEEE* 1994; 82: 55–67.
30. Baker T. Multiprocessor EDF and deadline monotonic schedulability analysis. In: *24th IEEE real-time systems symposium*, Cancun, Mexico, 5 December 2003, pp. 120–129. New York: IEEE.
31. Bernat G, Colin A and Petters S. WCET analysis of probabilistic hard real-time systems. In: *23rd IEEE real-time systems symposium*, Austin, TX, 3–5 December 2002, pp. 279–288. New York: IEEE.
32. Edgar S and Burns A. Statistical analysis of WCET for scheduling. In: *22nd IEEE real-time systems symposium (RTSS 2001)*, London, 3–6 December 2001, pp. 215–224. New York: IEEE.
33. Becker M and Chakraborty S. 21st international workshop on software and compilers for embedded systems. In: *21st international workshop on software and compilers for embedded systems*, Sankt Goar, 28–30 May 2018, pp. 10–13. New York: Association for Computing Machinery.
34. Meng F, Sun H and Wang J. Survey on estimation and optimization of worst-case execution time with energy consumption constraint. In: *12th international conference on computational intelligence and communication networks (CICN)*, Bhimtal, India, 25–26 September 2020, pp. 316–320. New York: IEEE.
35. Gunzel M, von der Bruggen G and Chen J. Suspension-aware earliest-deadline-first scheduling analysis. *IEEE T Comput Aid D* 2020; 39: 4205–4216.
36. Sun Y and Lipari G. A weak simulation relation for real-time schedulability analysis of global fixed priority scheduling using linear hybrid automata. In: *22nd international conference on real-time networks and systems*, Versailles, 8–10 October 2014, pp. 35–44. New York: Association for Computing Machinery.
37. Guo Z, Bhuiyan A, Liu D, et al. Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms. In: *IEEE real-time and embedded technology and applications symposium (RTAS)*, Montreal, QC, Canada, 16–18 April 2019, pp. 156–168. New York: IEEE.
38. Joseph M and Pandya P. Finding response times in a real-time system. *Comput J* 1986; 29: 390–395.
39. Moghadam M, Saadatmand M, Borg M, et al. Learning-based response time analysis in real-time embedded systems: a simulation-based approach. In: *IEEE/ACM 1st international workshop on software qualities and their dependencies (SQUADE)*, Gothenburg, May 2018, pp. 21–24. New York: IEEE.
40. Blazewicz J, Ecker K, Pesch E, et al. Scheduling imprecise computations. In: *Handbook on scheduling*. Cham: Springer, 2019, pp. 527–576, https://link.springer.com/chapter/10.1007/978-3-319-99849-7_14
41. Huang L, Hou I, Sapatnekar S, et al. Graceful degradation of low-criticality tasks in multiprocessor dual-criticality systems. In: *26th international conference on real-time networks and systems*, Chasseneuil-du-Poitou, October 2018, pp. 159–169. New York: Association for Computing Machinery.
42. Stavrinides G and Karatza H. Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations. *SIMULATION* 2009; 85: 525–536.
43. Huang L, Li Y, Sapatnekar S, et al. Using imprecise computing for improved non-preemptive real-time scheduling. In: *55th ACM/ESDA/IEEE design automation conference (DAC)*, San Francisco, CA, 24–28 June 2018, pp. 1–6. New York: IEEE.
44. Guo C, Zhu C and Tay T. Design and simulation of a green broker with imprecise computation scheduling for energy-efficient large scale computing in clusters. *J Emerg Trends Comput Inf Sci* 2013; 4: 900–907.
45. Esmaili A, Nazemi M and Pedram M. Energy-aware scheduling of task graphs with imprecise computations and end-to-end deadlines. *ACM T Des Automat El* 2020; 25: 1–21.
46. Stavrinides G and Karatza H. A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds. In: *3rd international*

- conference on future internet of things and cloud*, Rome, 24–26 August 2015, pp. 231–239. New York: IEEE.
47. Pazzaglia P, Sun Y and Natale M. Generalized weakly hard schedulability analysis for real-time periodic tasks. *ACM T Embed Comput S* 2021; 20: 1–26.
 48. Hong J, Song H, Kim T, et al. A real-time discrete event system specification formalism for seamless real-time software development. *Discrete Event Dyn S* 1997; 7: 355–375.
 49. Moallemi M and Wainer G. Modeling and simulation-driven development of embedded real-time systems. *Simul Model Pract Th* 2013; 38: 115–131.
 50. Mello B and Wainer G. Scheduling predictability in I-DEVS by schedulability analysis. In: *Symposium on theory of modeling and simulation*, Pasadena, CA, 3–6 April 2016, pp. 1–8. New York: IEEE.
 51. Manolache S, Eles P and Peng Z. Schedulability analysis of applications with stochastic task execution times. *ACM T Embed Comput S* 2004; 3: 706–735.
 52. Nasri M, Baruah S, Fohler G, et al. On the optimality of RM and EDF for non-preemptive real-time harmonic tasks. In: *22nd international conference on real-time networks and systems*, Versailles, 8–10 October 2014, pp. 331–340. New York: Association for Computing Machinery.
 53. Kim K, Kang W, Sagong B, et al. Efficient distributed simulation of hierarchical DEVS models: transforming model structure into a non-hierarchical one. In: *33rd annual simulation symposium (SS 2000)*, Washington, DC, 16–20 April 2000, pp. 227–233. New York: IEEE.
 54. Shang H and Wainer G. Dynamic structure DEVS: improving the real-time embedded systems simulation and design. In: *41st annual simulation symposium*, Ottawa, ON, Canada, 13–16 April 2008, pp. 271–278. New York: IEEE.
 55. Chow A, Zeigler B and Kim D. Abstract simulator for the parallel DEVS formalism. In: *Fifth annual conference on AI, and planning in high autonomy systems*, Gainesville, FL, 7–9 December 1994, pp. 157–163. New York: IEEE.

Author biographies

Braulio A Mello is a professor at Federal University of Fronteira Sul, Department of Computer Science, Chapecó-SC, Brazil. He has a PhD in Computer Science from Federal University of Rio Grande do Sul, Brazil. His e-mail address is braulio@uffs.edu.br.

Gabriel A Wainer, FSCS, is professor at Department of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada). He is an ACM Distinguished Speaker and a Fellow of SCS. His current research interests include modeling methodologies and tools, parallel/distributed simulation, and real-time systems. His e-mail and web addresses are [gwainer@sce-carleton.ca](mailto:gwainer@sce.carleton.ca) and <https://www.sce.carleton.ca/faculty/wainer>.