

Ph.D. Thesis Proposal

Mapping Continuous-Time Formalisms Onto the DEVS Formalism: Analysis and Implementation of Strategies

Jean-Sébastien Bolduc
(jseb@cs.mcgill.ca)
School of Computer Science
McGill University
Montréal

Supervisor: prof. Hans L. Vangheluwe
Co-Supervisor: prof. Denis Thérien

Abstract

We are interested in mapping *Ordinary Differential Equations* (ODEs) models onto a behaviourally equivalent representation in the *Discrete Event system Specification* (DEVS) formalism. The mapping from continuous-time models onto discrete-event models is based on *quantization*, an approach dual to the common *discretization* methods. As compared to discretization-based methods, the alternative approximation technique offers increased parallelization possibilities, as well as an easier approach to simulating multi-formalism models (particularly, hybrid models) through the use of a single common formalism. Our goals include the *design* of approximation methods, their *analysis*, and the *implementation* and optimization of a modelling and simulation package. In this proposal we present an overview of the relevant material, as well as a detailed exposition of the projected work. Some early results are also discussed, which confirm the usefulness of the idea. Our work will contribute to both practical and theoretical aspects of modelling and simulation.

1 Introduction

The *Discrete Event system Specification* (DEVS) formalism was first introduced by Zeigler [10, 11] in 1976, to provide a rigorous common basis for discrete-event modelling and simulation. By this we mean that it is possible to express in the DEVS formalism, popular discrete-event formalisms such as *event-scheduling*, *activity-scanning* and *process-interaction*. The main merit of the formalism is that a DEVS simulator is highly suitable as a Virtual Machine for the effective, and possibly parallel simulation of discrete-event models.

The class of formalisms denoted as *discrete-event* is characterized by a continuous time base where only a finite number of events can occur during a finite time-span. This contrasts with *Discrete Time System Specification* (DTSS) formalisms where the time base is isomorphic to \mathbb{N} , and with *Differential Equation System Specification* (DESS, or *continuous-time*) formalisms in which the state of the system may change continuously over time.

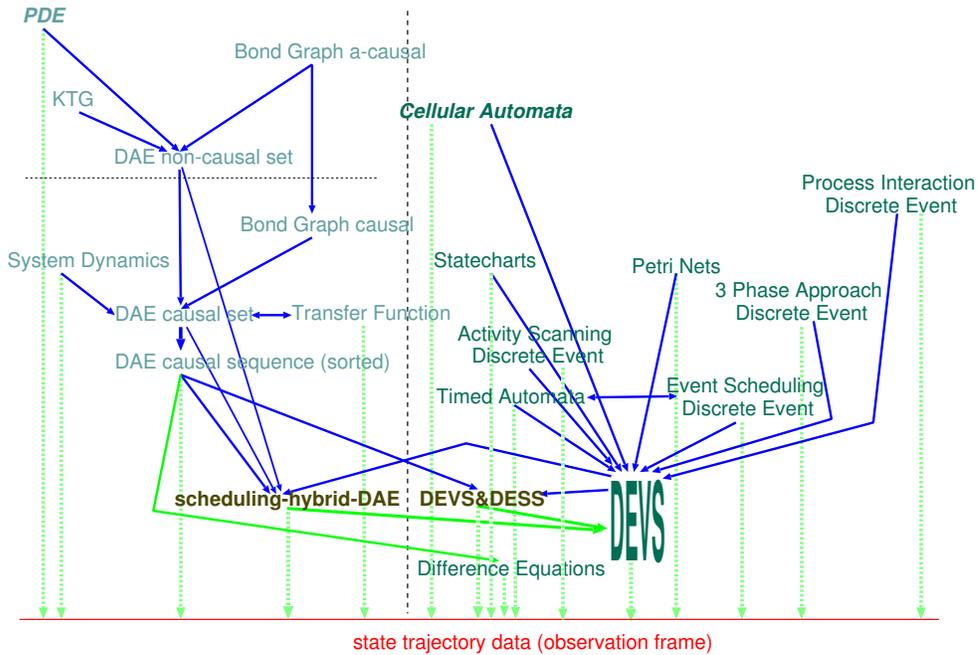


Figure 1: Formalism Transformation Graph

The *Formalism Transformation Graph* (FTG) [8, 9] shown in Figure 1, depicts behaviour-conserving transformations between some important formalisms. The graph distinguishes between continuous-time formalisms on the left-hand side, and discrete formalisms (both discrete-time and discrete-event) on the right-hand side. Although the graph suggests that formalisms can be mapped onto a common formalism on their respective sides, very few transformations allow crossing the middle-line: this illustrates why *hybrid systems* (those that bring together both discrete and continuous systems) are difficult to solve.

The traditional approach to solve continuous-time problems is based on *discretization*, which approximates a continuous-time model by a discrete-time system (difference equations). A partitioning of the time-axis, as is the case in discretization, is however hard to harmonize with a partitioning of the state space, as is performed in discrete-event systems (see [5]).

In this work we propose to map continuous-time formalisms (ODEs and semi-explicit DAEs) onto the DEVS formalism (this corresponds to the arrow going from “scheduling-hybrid-DAE” to “DEVS” on the FTG) through *quantization*. This mapping will be an approximate morphism and will thus require a detailed analysis. In doing this, our prime goal is to use a single “base” formalism for all simulations. To be usable, this formalism must be both simple (which is not the case for the DEV&DESS formalism [6]), and have performance comparable to those of traditional approaches.

As mentioned above, succeeding in our goal would result in a simpler, more elegant way of simulating *hybrid systems*, or more generally, *multi-formalism systems*. Another consequence would be the increased possibilities for distributed (parallel) implementations:

The closure property (under composition, or coupling) of systems such as DEVS offers the possibility to describe a model as a hierarchical composition of simpler sub-components. Apart from the obvious advantages associated with modularity (conceptual level, component reusabil-

ity), a significant gain in the efficiency of simulating large, complex dynamic systems can also be achieved by using *multi-rate integration* (employing different integration frame rates for the simulation of fast and slow sub-components), either on a single or on multiple processors (parallelization). Although some continuous-time formalisms (*e.g.*, causal-block diagram simulation tools) allow model hierarchization, multi-rate integration mixes poorly with traditional approaches where discretization along the time-axis forces the simulator to work explicitly with the global time base. This, in contrast to discrete-event formalisms where the simulator is concerned with local state space changes, and the time base is dealt with implicitly. Discrete event concepts are thus better suited for parallel/distributed simulation, and much effort has been devoted to the development of conservative (*e.g.*, Chandy-Misra approach), optimistic (*e.g.*, Time-Warp) and real-time (*e.g.*, DARPA’s Distributed Interactive Simulation) parallel discrete event simulation techniques. The relevance of DEVS in that context is illustrated by the concept of the *DEVS bus* which concerns the use of DEVS models as “wrappers” to enable a variety of models to inter operate in a networked simulation. The DEVS bus has been implemented on top of the DoD’s *High Level Architecture* (HLA) standard, itself based on the *Run-Time Infrastructure* (RTI) protocol.

This report is structured as follows: The next section gives a very brief overview of numerical analysis concepts that will be required to assess the validity of the quantization schemes. In section 3 we describe the classic DEVS as well as the P-DEVS formalisms, and section 4 introduces the concept of *quantization*, by which an ODE can be mapped onto a DEVS model. Finally, section 5 presents in more detail the objectives and the specific questions we would like to answer.

2 Continuous-Time Formalisms

Ordinary Differential Equations (ODEs) and *Differential Algebraic Equations* (DAEs) are ubiquitous in all fields of science. Continuous problems can be classified into two broad categories: *Initial Value Problems* (IVPs) and *Boundary Value Problems* (BVPs). We will only be interested in the former here (though some BVPs can be solved interactively by using shooting methods over IVPs), and will further assume that we only have one independent variable t (for all practical purposes, time): Numerical methods such as finite elements are used for *Partial Differential Equations* (PDEs), which are not subject to that last constraint. We will further only be interested in *semi-explicit DAEs*, which can be transformed into ODEs with constraints, and pure ODEs; this is not that a severe restriction, as many continuous-time models of interest fall under the semi-explicit DAE scheme.

A succinct overview of the numerical aspects of discretization is presented below. This material will be crucial in establishing the validity of quantization approaches, introduced in section 4, as an alternative to traditional discretization.

2.1 Discretization of ODEs and DAEs

The most general form of an initial value ODE is

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(\mathbf{y}, t) \\ \mathbf{y}(t_0) &= \mathbf{y}_0, \end{aligned}$$

where boldface denotes a vector, and \mathbf{y}_0 is the given initial condition. We say that the system is *autonomous* if the function \mathbf{f} does not depend on t .

We are interested in finding the solution $\mathbf{y}(t)$. Such a solution is guaranteed to exist and to be unique provided that $\mathbf{f}(\mathbf{y}, t)$

1. is continuous in the infinite strip $\mathcal{D} = \{t_0 \leq t \leq t_f, |\mathbf{y}| \leq \infty\}$,

2. and is, more specifically, *Lipschitz continuous* in the dependent variable \mathbf{y} over the same region \mathcal{D} , *i.e.*, \exists a positive constant K such that $\forall (\mathbf{y}, t), (\tilde{\mathbf{y}}, t) \in \mathcal{D}$,

$$|\mathbf{f}(\mathbf{y}, t) - \mathbf{f}(\tilde{\mathbf{y}}, t)| \leq K|\mathbf{y} - \tilde{\mathbf{y}}|.$$

Actually, these (sufficient) conditions also guarantee that the solution depends Lipschitz continuously on the initial condition. *i.e.*, if $\mathbf{y}(t)$ is the solution to the original problem and now $\tilde{\mathbf{y}}(t)$ also satisfies the ODE but with a different initial condition $\tilde{\mathbf{y}}(t_0)$, then \exists a positive constant L such that

$$|\mathbf{y}(t) - \tilde{\mathbf{y}}(t)| \leq L|\mathbf{y}(t_0) - \tilde{\mathbf{y}}(t_0)|.$$

This condition together with the existence and uniqueness of a solution defines a *well-posed problem* [1].

An approximate solution to an ODE is typically obtained by iterating a set of *difference equations* that “approximate” the original system. For this we need to *discretize* the independent variable t . Let $\{t_i | i = 0, 1, \dots, n\}$ be a mesh over the interval of integration, $t_0 < t_1 < \dots < t_{n-1} < t_n = t_f$; then $h_i = t_i - t_{i-1}$ is the i^{th} *step-size*. We assume for simplicity a uniform mesh, $h = \frac{t_f - t_0}{n}$. The recursive application of the difference equation defines a mesh $\{\mathbf{y}_i\}$, with each \mathbf{y}_i an approximation of the exact solution $\mathbf{y}(t_i)$.

The simplest difference approximation scheme is the so-called *Euler-Cauchy* (or *polygonal*) method,

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h\mathbf{f}(\mathbf{y}_i, t_i),$$

which can be obtained by truncating the Taylor expansion of $\mathbf{y}(t)$ after the first-order term, or by approximating the derivative by a forward difference quotient. This is an *explicit* method, in that the new value \mathbf{y}_{i+1} is readily obtained from the old value \mathbf{y}_i .

Compared to ODE theory, DAE theory is more recent, and offers a much more complex challenge [1]: no general existence and uniqueness condition can be stated for this class of problems, and it follows that very few quantitative results can be deduced.

The most general form of a DAE is

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0}.$$

We will be interested here only in the simpler special case of semi-explicit DAEs, which can be reformulated as *ODEs with constraints*:

$$\begin{aligned} \mathbf{x}' &= \mathbf{f}(t, \mathbf{x}, \mathbf{z}) \\ \mathbf{0} &= \mathbf{g}(t, \mathbf{x}, \mathbf{z}), \end{aligned}$$

where we distinguish between *differential variables* \mathbf{x} and *algebraic variables* \mathbf{z} . We can in principle preprocess the DAE to solve for \mathbf{z} in the latter set of equations. The first step of the preprocessing consists in symbolic reduction. Dependency cycles are then explored with an “algebraic loop” detection algorithm such as Tarjan’s. If loops are present, we can try to solve the implicit set of equations (by numeric or symbolic manipulation, or a nonlinear residual solver). When no loops are present, the equations are sorted and then solved explicitly. Substituting the solutions into the first equation yields an ODE in \mathbf{x} (although no uniqueness is guaranteed).

2.2 Numerical Analysis

Numerical analysis is concerned with the study of convergence and stability, and a suitable choice of the step-size h . For completeness we also mention *sensitivity analysis*, which studies the effects of introducing perturbations (due to roundoffs or measurement errors) in the original ODE. For a difference approximation to be usable for a class of functions $\mathbf{f}(\mathbf{y}, t)$, it is necessary that any function in this class satisfies a number of requirements [3]:

The first requirement concerns the existence and uniqueness of a solution. This is readily satisfied by explicit schemes, and can usually be ascertained for implicit schemes.

Secondly, we expect that for sufficiently small h , $\{\mathbf{y}_i\}$ should be close in some sense to $\{\mathbf{y}(t_i)\}$. Since the scheme we use is an approximation of the original problem, we expect it to introduce an error upon each iteration: assuming infinite precision arithmetic, we call this approximation error the *local truncation error* τ_i (from the truncation of the Taylor expansion). If we can prove for a given scheme that

$$\lim_{h \rightarrow 0} \tau_i = \mathbf{0},$$

then the method is said to be *consistent* (or *accurate*). However, we are interested in the accumulation of these errors: we write $\mathbf{y}_i = \mathbf{y}(t_i) + \mathbf{e}_i$, where \mathbf{e}_i is the *global truncation error* (equivalent to summing up the τ_i 's under the assumption that $\mathbf{e}_0 = \mathbf{0}$). If we can show for a given system that

$$\lim_{h \rightarrow 0} \mathbf{e}_i = \mathbf{0},$$

then the method is said to be *convergent*. For instance, we can find for the Euler-Cauchy method that $|\tau_i| = O(h^2)$ (consistent of order 2), and $|\mathbf{e}_i| = O(h)$ (convergent of order 1).

The third requirement is that the solution should be “effectively computable”. This concerns, on the one hand, the computational efficiency of the implemented method; on the other, since we cannot assume infinite precision arithmetic in practice, we want to estimate the growth of *roundoff errors* in the solution. This is related to *stability* of the method, which is actually a much more general concept: a method is said to be unstable if, during the solution procedure, the result becomes unbounded. This phenomenon arises when the difference equations themselves tend to amplify errors to the point that they obliterate the solution itself. One should not confuse the “stability” of the numerical method we discuss here with the “stability” of the original ODE problem, which is concerned with the analysis of the attractors. However, both concepts are related since the numerical solution is expected to behave similarly to the true solution.

A method is said to be stable (or 0-stable) if the corresponding difference equation is stable. This stability is easy to assess when the difference equation is linear in $(\mathbf{y}_i, \mathbf{y}_{i+1}, \dots, \mathbf{y}_{i+k})$, by studying the eigenvalues associated with its characteristic equation. The *convergence theorem* states that if a method is both stable and consistent (of order p), then it is convergent (of order p).

The concepts introduced above concern the limit behaviour of a method when $h \rightarrow 0$ and $i \rightarrow \infty$. However we are usually interested in finding an expression for the upper bound to the *total error* (truncations and roundoffs) in function of the stepsize h . It turns out that “stability” as defined earlier might not be suitable for that (as they might yield over-pessimistic approximations for h , especially when the original system is stiff). Various other stability schemes are then used (such as *absolute stability*).

3 The DEVS Formalism

We first introduce the original DEVS formalism known as *classic DEVS*. The question whether the formalism describes a “system” (*i.e.*, under which conditions it is well-behaved is a system-

theory sense) is also covered. It turns out that even a well-behaved DEVS model can behave in a counter-intuitive manner. Finally, the P-DEVS formalism, which removes some deficiencies of the original DEVS, is presented.

3.1 The Classic DEVS Formalism

Classic DEVS is an intrinsically sequential formalism that allows for the description of system behaviour at two levels: at the lowest level, an *atomic-DEVS* describes the autonomous behaviour of a discrete-event system as a sequence of deterministic transitions between states as well as how it reacts to external inputs. At the higher level, a *coupled-DEVS* describes a discrete-event system in terms of a network of coupled components, each an atomic-DEVS model (or a coupled-DEVS in its own right, as we will see in section 3.2).

The atomic-DEVS

An atomic-DEVS M is specified by a 7-tuple

$$M = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta),$$

where:

X is the *input set*,

Y is the *output set*,

S is the *state set*,

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

$\delta_{ext} : Q \times X \rightarrow S$ is the *external transition function*, where

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the *total state set*, and
 e is the *elapsed time* since the last transition.

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

$ta : S \rightarrow \mathbb{R}^+ \cup \{0, +\infty\}$ is the *time advance function*.

There are no restrictions on the sizes of the sets, which typically are product sets, *i.e.*, $S = S_1 \times S_2 \times \dots \times S_n$. In the case of the state set S , this formalizes multiple concurrent parts of a system, while it formalizes multiple input and output ports in the case of sets X and Y . The time base T is not mentioned explicitly and is continuous. For a discrete-event model described by an atomic-DEVS M , the behaviour is uniquely determined by the initial total state $(s_0, e_0) \in Q$ and is obtained by means of the following iterative simulation procedure (refer to Figure 2):

At any time t the system is in some state $s \in S$. If no external event occurs, the system will stay in state s until the elapsed time e reaches $ta(s)$. The *time left*, $\sigma = ta(s) - e$, is often introduced as an alternate way to check for the time until the next (internal) transition. The system then first produces the output value $\lambda(s)$ and makes a transition to state $s' = \delta_{int}(s)$ (*e.g.*, at time t_1 and t_3 in Figure 2). If an external event $x \in X$ occurs before e reaches $ta(s)$, the system interrupts its autonomous behaviour and instantaneously goes to state $s' = \delta_{ext}((s, e), x)$ (*e.g.*, at time t_2 in Figure 2). Thus, the internal transition function dictates the system's new state based on its old state in the absence of external events. The external transition function dictates the system's new state whenever an external event occurs, based on this event x , the current state s and how long the system has been in this state, e . After both types of transitions, the elapsed time e is reset to 0.

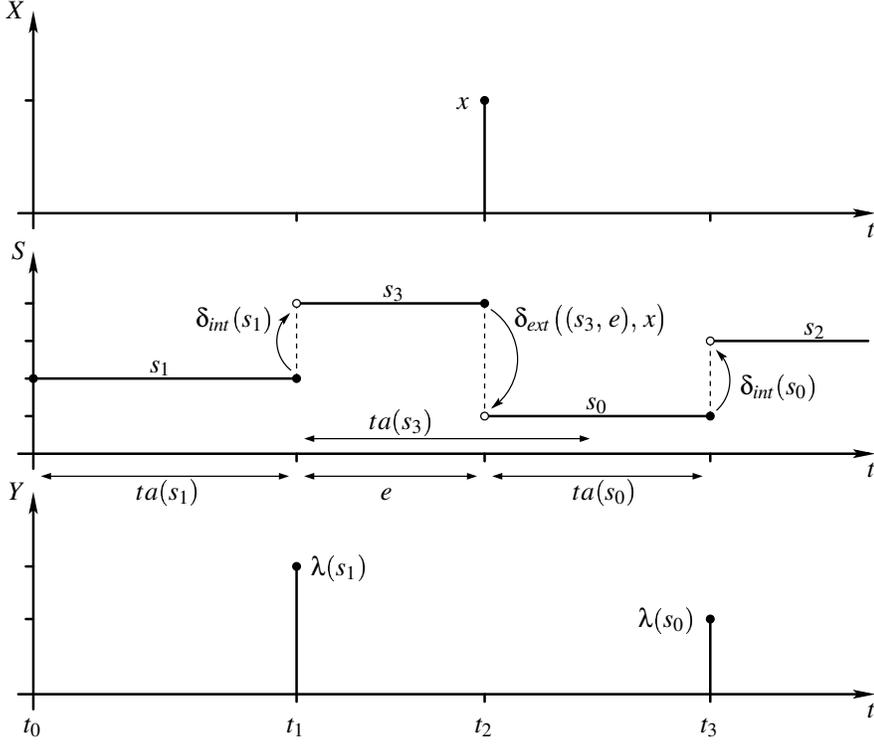


Figure 2: State Transitions of an atomic-DEVS Model

The term *collision* refers to the situation where an external transition occurs at the same time as an internal transition. When such a collision occurs, the atomic-DEVS formalism specifies that the tie between the two transition functions shall be solved by first carrying out the internal, then the external transition function. This is consistent with the way collisions are handled in coupled-DEVS, as will be seen shortly.

Note that $ta(s)$ can take on values 0 and $+\infty$: in the first case, the system stays in state s for a time so short that no external event can intervene, and we say that s is a *transitory state*. In the latter case the system stays in state s forever unless an external event occurs, and we say that s is a *passive state*.

Outputs are associated only with internal transitions to impose a delay on the propagation of events. In a modular construct (see coupled-DEVS below), zero-time propagation could result in infinite instantaneous loops. Such ill-behaved (or ill-defined) systems can of course still be constructed using transitory states (despite only associating outputs with internal transitions). Actually, instantaneous loops are at the heart of the *legitimacy* issue introduced in section 3.2. Since outputs are only generated in the absence of external events, the atomic-DEVS formalism is a Moore machine. From an implementation point of view, it is easy to emulate the effect of generating an output upon entering a state by using $\lambda(\delta_{int}(s))$.

The coupled-DEVS

A coupled-DEVS N is specified by a 7-tuple

$$N = (X, Y, D, \{M_i\}, \{I_j\}, \{Z_{j,k}\}, \gamma),$$

where:

X is the *input set*,

Y is the *output set*,

D is the *set of component indexes*,

$\{M_i \mid i \in D\}$ is the *set of components*, each M_i being an atomic-DEVS:

$$M_i = (X_i, Y_i, S_i, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i),$$

$\{I_j \mid j \in D \cup \{self\}\}$ is the *set of all influencer sets*, where $I_j \subseteq D \cup \{self\}$, $j \notin I_j$ is the *influencer set of j* ,

$\{Z_{j,k} \mid j \in D \cup \{self\}, k \in I_j\}$ is the *set of output-to-input translation functions* where:

$$Z_{j,k} : X \rightarrow X_k \quad \text{if } j = self,$$

$$Z_{j,k} : Y_j \rightarrow Y \quad \text{if } k = self,$$

$$Z_{j,k} : Y_j \rightarrow X_k \quad \text{otherwise.}$$

$\gamma : 2^D \rightarrow D$ is the *select function*.

The sets X and Y typically are product sets, which formalizes multiple input and output ports. To each atomic-DEVS in the network is assigned a unique identifier in the set D . This corresponds to model names or references in a modelling language. The coupled-DEVS N itself is referred to by means of $self \notin D$. This provides a natural way of indexing the components in the set $\{M_i\}$, and to describe the sets $\{I_j\}$ and $\{Z_{j,k}\}$.

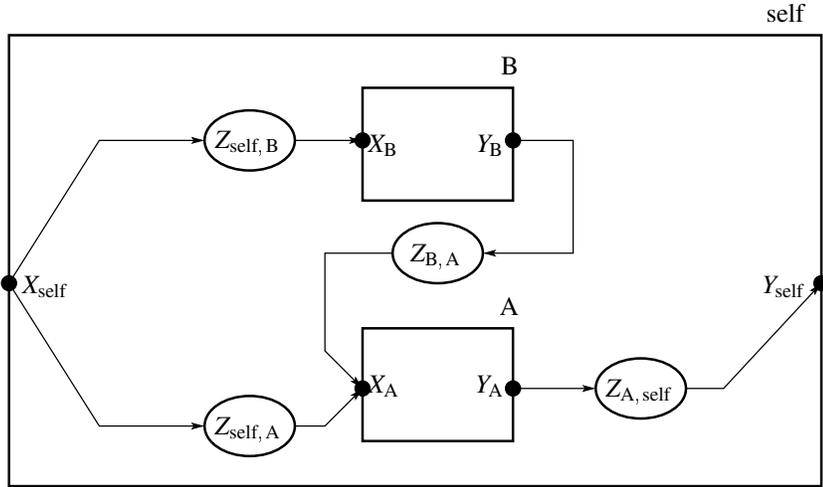


Figure 3: coupled-DEVS

The sets in $\{I_j\}$ explicitly describe the coupling network structure: for instance, in Figure 3 we have $I_A = \{self, B\}$, $I_B = \{self\}$ and $I_{self} = \{A\}$. For modularity reasons, a component may not be influenced by components outside its enclosing scope, defined as $D \cup \{self\}$. The condition $j \notin I_j$ forbids a component to directly influence itself, to prevent instantaneous dependency cycles. The functions $Z_{j,k}$ describe how an influencer's output is mapped onto an influencee's input. The set of output-to-input transition functions implicitly describes the coupling network structure, which is sometimes divided into *External Input Couplings* (EIC, from the coupled-DEVS' input to a

component's *input*), *External Output Couplings* (EOC, from a component's *output* to the coupled-DEVS' *output*), and *Internal Couplings* (IC, from a component's *output* to a component's *input*).

As a result of coupling concurrent components, multiple internal transitions may occur at the same simulation time t . Since in sequential simulation systems only one component can be *activated* at a given time, a tie-breaking mechanism to select which of the components should be handled first is required. The classic coupled-DEVS formalism uses the select function γ to choose a unique component from the set of *imminent components*, defined as $\Pi_t = \{i \mid i \in D, \sigma_i = 0\}$, *i.e.*, those components that have an internal transition scheduled at time t . The component returned by $\gamma(\Pi_t)$ will thus be activated first. For the other components in the imminent set, we are left with the following ambiguity [2]: when an external event is received by a model at the same time as its scheduled internal transition, which elapsed time should be used by the external transition: $e = 0$ of the new state, or $e = ta(s)$ of the old state? These *collisions* are resolved by letting $e = 0$ for the unique activated component, and $e = ta(s)$ for all the others.

3.2 Well-Defined Systems and Legitimacy

It is shown in [12] how the classic DEVS formalism is *closed under coupling*: given a coupled model N with atomic-DEVS components, we can construct an atomic-DEVS M_{res} . The construction procedure is compliant with our intuition about concurrent behaviour and resembles the implementation of event-scheduling simulators. At its core is the total time-order of all events in the system. By induction, closure under coupling leads to hierarchical model construction, where the components in a coupled model can themselves be coupled-DEVS. This means that the results developed for atomic-DEVS in this section also apply to coupled models.

We have emphasized in the previous section how transitory states in a DEVS model could result in an ill-behaved system when zero-time advance cycles are present. *Legitimacy* is the property of DEVS that formalizes these notions.

For an atomic-DEVS M , we define legitimacy by first introducing an iterative internal transition function $\delta_{int}^+ : S \times \mathbb{N} \rightarrow S$, that returns the state reached after n iterations starting at state $s \in S$ when no external event intervenes. It is recursively defined as

$$\begin{aligned}\delta_{int}^+(s, n) &= \delta_{int}(\delta_{int}^+(s, n-1)) \\ \delta_{int}^+(s, 0) &= s.\end{aligned}$$

Next we introduce a function $\Gamma : S \times \mathbb{Z} \rightarrow \mathbb{R}_0^+$ that accumulates the time the system takes to make these n transitions:

$$\begin{aligned}\Gamma(s, n) &= \Gamma(s, n-1) + ta(\delta_{int}^+(s, n-1)) \\ &= \sum_{i=0}^{n-1} ta(\delta_{int}^+(s, i)) \\ \Gamma(s, 0) &= 0.\end{aligned}$$

With these definitions, we say that a DEVS is *legitimate* if for each $s \in S$,

$$\lim_{n \rightarrow \infty} \Gamma(s, n) \rightarrow \infty.$$

Equivalently, legitimacy can be interpreted as a requirement that there are only a finite number of events in a finite time-span. It can be shown that *the structure specified by a DEVS is a well-defined system if, and only if, the DEVS is legitimate*.

For atomic-DEVS M with S finite, a necessary and sufficient condition for legitimacy is that every cycle in the state diagram of δ_{int} contains at least one non-transitory state. For the case where S is infinite however, there exists only a stronger-than-necessary sufficient condition, namely, that there is a positive lower bound to the time advances, i.e., $\exists b$ s.t. $\forall s \in S, ta(s) > b$. In the context of our work this condition is too restrictive, and we will want to refine it as explained in section 5.1.

3.3 The P-DEVS Formalism

Because of the inherent sequential nature of classic DEVS, modelling using this formalism requires extra care. As a matter of fact, resolving collisions by means of the select function γ might result in counter-intuitive behaviours.

The *Parallel-DEVS* formalism [2] (or *P-DEVS*, to distinguish it from parallel *implementations* of both classic DEVS and P-DEVS) was introduced to solve these problems by properly handling collisions between simultaneous internal and external events. As the name indicates, P-DEVS is a formalism whose semantics successfully describes (irrespective of sequential or parallel implementations) concurrent transitions of imminent components, without the need for a priority scheme.

Just as in the case of classic DEVS, P-DEVS allows for the description of system behaviour at the atomic and coupled levels. The formalism is closed under coupling, which leads to hierarchical model construction. The concepts of legitimacy introduced in the previous section also apply to P-DEVS.

The formalism uses a *bag* as the message structure: a bag X^b of elements in X is similar to a set except that multiple occurrences of elements are allowed (e.g., $x^b = \{a, b, a\} \in X^b$). As with sets, bags are unordered. By using bags to collect inputs sent to a component, we recognize that inputs can arrive from multiple sources and that more than one input with the same identity may arrive simultaneously.

The atomic formalism for P-DEVS M is specified by an 8-tuple

$$M = (X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta).$$

The definition is almost identical to that of the classic version, except that we introduce the concept of a bag in the external transition and output functions:

$$\begin{aligned} \delta_{ext} &: Q \times X^b \rightarrow S, \\ \lambda &: S \rightarrow Y^b. \end{aligned}$$

This reflects the idea that more than one input can be received simultaneously, and similarly for the generation of outputs. P-DEVS also introduces the *confluent transition function*

$$\delta_{con} : S \times X^b \rightarrow S,$$

which gives the modeller complete control over the collision behaviour when a component receives external events at the time of its internal transition. Rather than serializing model behaviour at collision times through the select function γ at the coupled level, P-DEVS leaves the decision of what serialization to use to the individual component.

It follows that the coupled formalism for P-DEVS N is specified by a 6-tuple

$$N = (X, Y, D, \{M_i\}, \{I_j\}, \{Z_{j,k}\}).$$

We note the absence of the select function γ . All the remaining elements have the same interpretation as in the classic version, except that here again the bag concept must be introduced in the output-to-input translation functions $\{Z_{j,k}\}$.

The semantics of the formalism is simple: at any event time t , all components in the imminent set Π_t first generate their output, which get assembled into bags at the proper inputs. Then, to each component in Π_t is applied either the confluent or the internal transition function, depending whether it has received inputs or not. The external transition function is applied to those components that have received inputs and are outside the imminent set.

4 Quantization Approach to the Numerical Solution of ODEs

As an alternative to the traditional discretization approach to the solution of ODEs briefly introduced in section 2, Zeigler [12] has proposed an approach based on partitioning of the state space rather than of the time domain. This *quantization* approach requires a change in viewpoint. The question “at which point in the future is the system going to be in a given state” is now asked instead of “in which state is the system going to be at a given future time”. In both questions a numerical procedure to produce the answer is derived from the ODE model.

When applied to a continuous signal, both quantization and discretization approaches yield an exact representation of the original signal only in the limit case where the partition size goes to zero (assuming a well-posed problem). Whereas DTSS seem to match discretized signals well, it turns out that DEVS is an appropriate formalism for quantized systems. Material in this section is derived from chapter 16 of Zeigler *et al.* [12].

4.1 Quantization Principles

We present a simple quantization π_X of the space X , an interval over \mathbb{R} , as follows: we introduce the sets $d_i = \{x \mid x \in X, \frac{D}{2}(2i-1) \leq x < \frac{D}{2}(2i+1)\}$, $i \in \mathbb{Z}$. Each denotes a *quantum* (or *cell*, *block*) of X , where D is the (uniform) quantum-size. In general, the sets d_i represent a tessellation of the space X , *i.e.*, $\bigcup_i d_i = X$ and $\forall i \neq j, d_i \cap d_j = \emptyset$. This could obviously be extended to higher-dimensions, defining “tiles” of arbitrary shapes, or of non-uniform sizes.

In each quantum a *representative* item \tilde{d}_i is designated: in our example, we chose it to be the “middle element” (given by D_i) of the quantum. Then $\pi_X = \{(d_i, \tilde{d}_i) \mid i \in \mathbb{Z}\}$. Under that scheme, we use the notation $[x]$ to refer to the quantum containing the point x . Furthermore, we extend the “tilde” notation to the quantization itself, denoting by $\tilde{\pi}_X$ the set of representatives. A more formal definition of quantization can be found in [7].

For a time base $T = \mathbb{R}$, we define a closed *interval* as $[t_1, t_2] = \{t \mid t \in T, t_1 \leq t \leq t_2\}$, and the open interval $]t_1, t_2[$ is defined in the obvious manner. We generally use the notation $\langle t_1, t_2 \rangle$ to refer to an interval that is either closed or open at either end. Then a function $u : \langle t_1, t_2 \rangle \rightarrow X$, or $u_{\langle t_1, t_2 \rangle}$, is called a *segment* over X and T .

Using the simple quantization scheme introduced above, we define the quantization of a segment $u_{\langle t_a, t_b \rangle}$ as the piecewise-continuous segment

$$u_{\langle t_a, t_b \rangle}^* = u_{1_{\langle t_a, t_1 \rangle}} \cdot u_{2_{\langle t_a, t_2 \rangle}} \cdots u_{n_{\langle t_{n-1}, t_b \rangle}},$$

where each $u_{i_{\langle t_{i-1}, t_i \rangle}}$ is a constant segment of value \tilde{d}_j , such that the range of the corresponding segment $u_{\langle t_{i-1}, t_i \rangle}$ lies entirely in quantum d_j (refer to Figure 4). We introduce a *quantizer* as a I/O-system Q_{π_X} that outputs the quantized version of its input.

Quantization suggests a new approach to solving ODEs, where a system updates its output only when a “sufficiently important” change has occurred in its state.

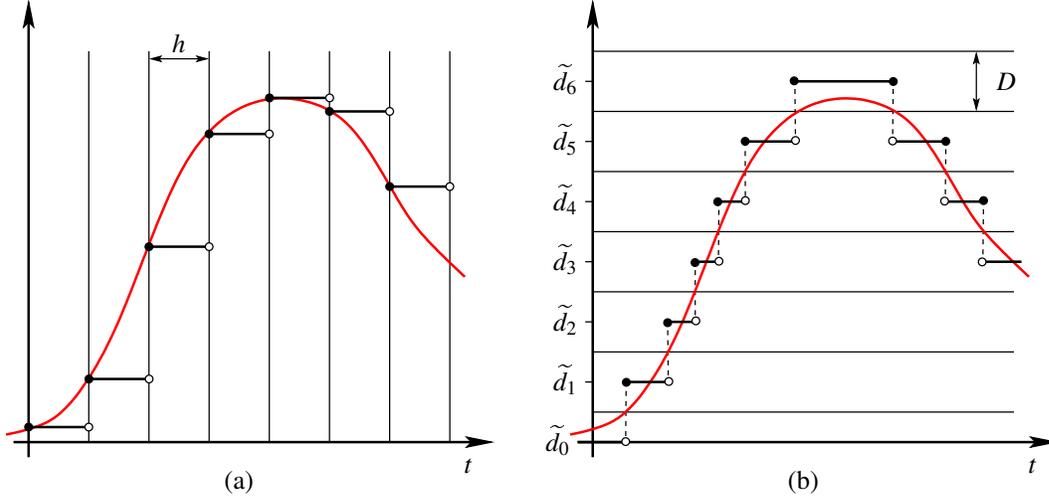


Figure 4: Discretization (a) and Quantization (b) of the same segment

4.2 DEVS Representation of Quantized Systems

By quantizing the input and output of a system S we obtain a *quantized system*, represented as $S^* = \langle Q_{\pi_X}, S, Q_{\pi_Y} \rangle$. This notation should be interpreted as a cascade of systems: the input quantizer Q_{π_X} sends the quantized version (a segment over $\tilde{\pi}_X$ and T) of an input segment (over X and T) to the internal system S , which sends its output (a segment over Y and T) to the output quantizer Q_{π_Y} . Hence, whereas the internal system S has input and output sets X and Y , the quantized system S^* has input and output sets X and $\tilde{\pi}_Y$. Here again we assume the sets X and Y are intervals over \mathbb{R} , and the input and output quantizers use quantum sizes D_1 and D_2 , respectively.

Quantization of systems is a general concept that imposes no constraints on the internal system S . We will assume for our present purpose that it represents a continuous-time system. The quantized system S^* is equivalent to the internal system S only in the limit case where D_1 and D_2 tend to 0.

It turns out that every quantized system can be simulated, without error, by a DEVS. This claim requires a precision regarding the nature of input signals: since discrete-event formalism are not compatible with continuous signals, the *quantized DEVS* equivalent of a quantized system expecting a continuous signal, will itself expect a *piecewise constant approximation* of a signal (actually represented by the equivalent *event signal*). Since we would expect the quantization of both (continuous, and piecewise constant approximation) signals to be identical, it is reasonable to assume the approximation scheme is indeed a quantization scheme. We will use this assumption, which eliminates the need for an input quantizer, when describing a DEVS quantized integrator below. We also observe that the quantization of a continuous signal has the property that each piecewise constant segment is exactly one quantum “above” or “below” its adjacent segments: this characteristic, *quanta-continuity*, could be exploited where applicable to reduce the size of messages to one single bit.

To give some insight in how to represent a quantized system by a DEVS, assume the internal system S has a set of states Q , and input and output sets X and Y . To map S onto an atomic-DEVS M , we let the DEVS’ input and output sets be $X_M = X$ and $Y_M = Y$. The state set is given by $S_M = Q \times \tilde{\pi}_X$, where the second component allows the DEVS to remember its last (quantized)

input. The time advance function ta is then the time to the next change in output produced by the output quantizer Q_{π_y} . The output function λ outputs the representative of the new quanta, while the internal transition function δ_{int} updates the state's first component accordingly. If a new input x' is received, δ_{ext} updates the DEVS' state as specified in the system S .

The first consequence of this is immediate: a (quantized) ODE can be simulated by a DEVS. This opens interesting perspectives. Since DEVS is closed under coupling, a quantized ODE can readily be coupled with, and interact with, purely discrete-event components. However, some care must be taken to avoid sending a quantized signal to a quantizer with a finer quantum size, which could result in unexpected results (as this could break the quanta-continuity property described above). This requirement is called *partition refinement*.

A second consequence stems from the quantized system versus DEVS equivalence. A quantized system S^* can be obtained by literally sandwiching a system S between quantizers. Since this new system behaves as a DEVS, it can readily be coupled with DEVS, or other quantized systems for that matter. This idea, which allows coupling of various formalisms, is referred to as *system wrapping*. It is the basis of the DEVS bus concept.

4.3 Quantization Example

In this section we present some early empirical results obtained by experimenting with a *DEVS quantized integrator* as described in [12].

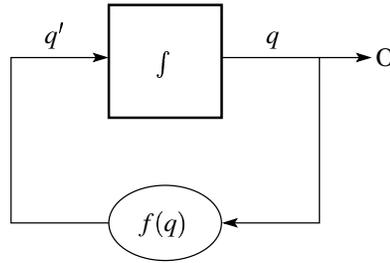


Figure 5: Causal-Block Diagram of an ODE

In section 2 we introduced the canonical form of an ODE. The autonomous, first-order form is

$$\begin{aligned} q' &= f(q) \\ q(t_0) &= q_0. \end{aligned}$$

Integrating both sides of the ODE, we can rewrite it as

$$q(t_f) = q(t_0) + \int_{t_0}^{t_f} f(q(t)) dt.$$

In causal-block diagram simulation systems, this can be simply implemented as an integrator block with feedback, as illustrated in Figure 5. The traditional Euler-Cauchy method can be obtained by discretizing the equation, after approximating the integral on the right-hand side by $(t_f - t_0) \cdot f(q(t_0))$. We adapt this scheme to the DEVS quantized integrator by approximating the integral by ex , where e is the elapsed time and x is the last input. It follows that when entering a new state (either after an internal or external transition), the time of residency in that state (time advance function) is obtained by solving the equation for the time until the current quantum is departed.

Using the simple quantization scheme described in section 4.1, the DEVS quantized integrator is defined as follows:

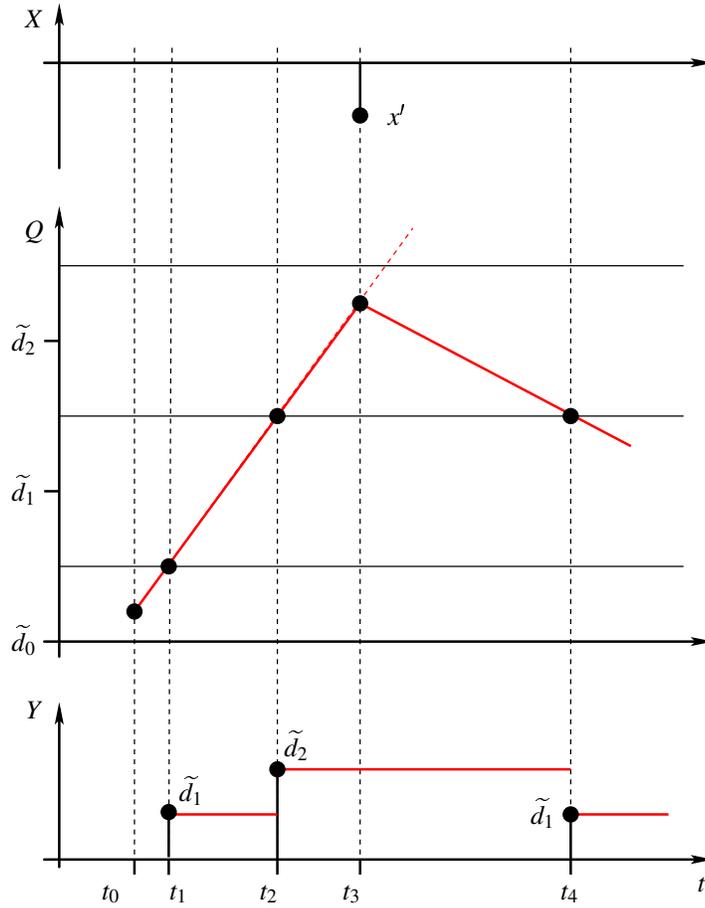


Figure 6: DEVS Quantized Integrator

- The input and output sets X and Y , as well as the set Q , are subintervals of \mathbb{R} . The same quantum size D is used for both;
- We define the state set as $S = Q \times \widetilde{\pi}_X \times \widetilde{\pi}_Q$, such that for a state $s = (q, x, r)$, q is the state itself, x stores the last input received (*i.e.*, the slope) and r is the representative $\widetilde{[q]}$. The latter component is necessary to break ties when q is at a quanta interface;
- The internal transition function brings the state component q to the quanta interface “above” or “below”, depending on the sign of the slope x :

$$\delta_{int}(q, x, r) = \left(r + \frac{D}{2} \cdot \text{sign}(x), x, r + D \cdot \text{sign}(x) \right).$$

Above and below are the only possibilities here, due to the continuity of q . The integral approximation is simplified by substituting the expression for $ta(s)$ in place of the elapsed time e ;

- The external transition function applies the Euler-Cauchy approximation of an integral, and stores the input received:

$$\delta_{ext}((q, x, r), e, x') = (q + ex, x', r);$$

- The output function returns the representative of the quantum the state is entering:

$$\lambda(q, x, r) = r + D \cdot \text{sign}(x);$$

- the time advance function returns the time to the next scheduled internal transition, *i.e.*, the time until the current quanta is departed:

$$\begin{aligned}
 ta(q, x, r) &= +\infty && \text{if } x = 0 \\
 &= \frac{|q - (r + \frac{D}{2} \cdot \text{sign}(x))|}{|x|} && \text{otherwise,}
 \end{aligned}$$

the numerator being the distance between the state q and the relevant quanta interface.

This prototype was implemented using a *classic DEVS modelling and simulation package* prototype that we wrote in Python. The actual code for the DEVS quantized integrator is presented in the appendix, and shows the simple syntax used to define a model.

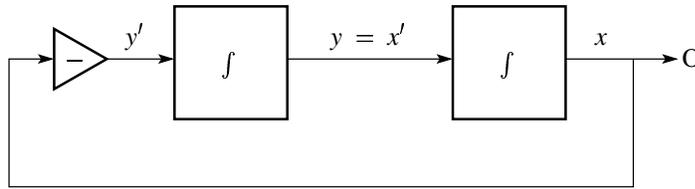


Figure 7: Causal-Block Model for the Circle Test

We performed a few tests with the *circle test* equation,

$$q'' = -q$$

whose analytical solution is

$$q(t) = c_1 \cdot \cos(t) + c_2 \cdot \sin(t),$$

where c_1 and c_2 are constants that depend on the initial conditions. It would be possible to approximate this ODE as an autonomous (*i.e.*, input-free) atomic-DEVS; however, using the DEVS quantized integrator presented above first requires us to rewrite the second order ODE as two coupled first-order ODEs:

$$x' = y \tag{1}$$

$$y' = -x. \tag{2}$$

The corresponding causal-block diagram is depicted in Figure 7. At the coupled level, the DEVS model includes two DEVS quantized integrators arranged in the same feedback pattern (see the appendix for the actual implementation of the coupled-DEVS model). The “gain” block in our case was included in the leftmost atomic-DEVS, and collisions were resolved by inserting passive buffers on the two couplings: although these 0-delay blocks violate the “strong” DEVS legitimacy condition stated in section 3.2, it turns out the model is still legitimate. The input and output quanta sizes D are identical, which seems reasonable because of feedback. Moreover, since the integrator’s internal state q is continuous (piecewise linear) and since the output is the quantization of that state, the output signal has the “quanta continuous” property.

The circle test being a second-order ODE, initial conditions must be specified for both $x(0)$ and $y(0) = x'(0)$. Choosing these to be 1 and 0 respectively, the analytical solution reduces to $x(t) = \cos(t)$. In Figures 8 and 9, we show the evolution of the first integrator’s internal state q as a function of time, as obtained by simulating the model with a quanta size $D = 0.1$. The

analytical solution is shown for comparison, and the first figure also shows the results obtained with a traditional Euler-Cauchy scheme (discretization) with a step-size $h = 0.02$. This resolution was chosen to roughly correspond to the minimum time between two internal transitions in the quantized counterpart.

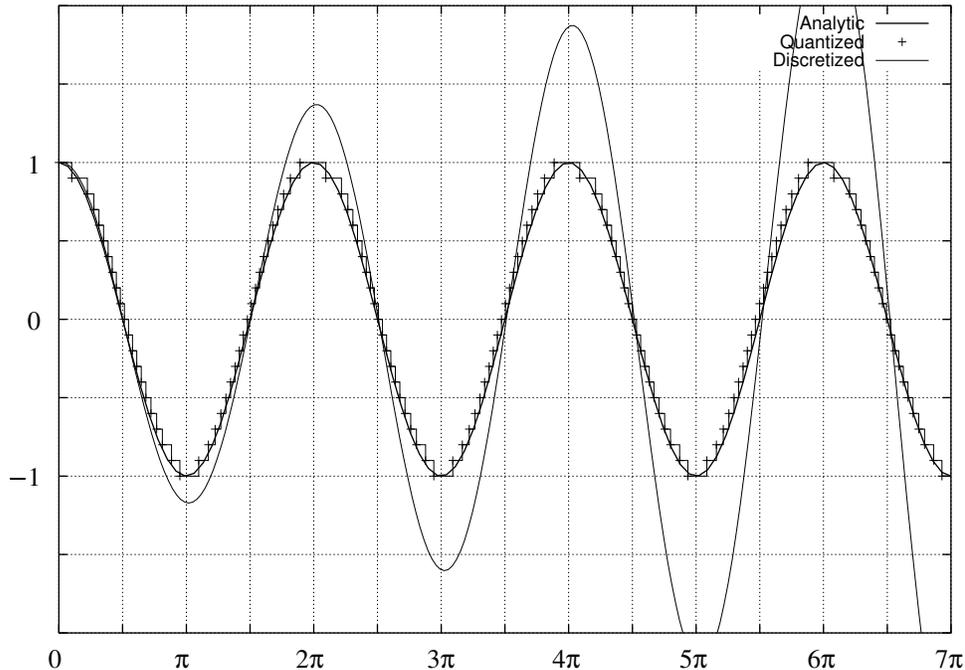


Figure 8: Simulation of the Circle Test Problem

The striking feature is that while the error on the discrete solution increases exponentially, the quantized solution remains bounded. Actually, as evidenced in Figure 9, the quantized solution *lags* behind the analytical solution: it might be intuitively appealing to postulate that whereas a discretization scheme introduces errors in the state space, a quantization scheme should introduce errors in the time domain (phase shift). This would be consistent with the fact that we solve an (approximate) equation for time in the latter scheme. Our empirical results suggest that the lag grows linearly rather than exponentially. A comparison of the errors will require us to define a metric applicable to both (*e.g.*, *difference* in phase-space). A simulation with t varying from 0 to 12π required roughly 480 transitions in the quantized solution, and around 1880 iterations in the discretized solution: however, whereas the cost of an iteration is a single flop, a transition is far more expensive (the functions of the DEVS quantized integrator cost by themselves four flops, and this is without counting all the overhead introduced by the coordination at the coupled-DEVS level). The quantized approach thus seems to be more expensive.

5 Thesis Objectives

The early results presented in the last section tend to confirm the usefulness of the quantization approach for ODEs, and the relevance of the DEVS formalism in that context. The many questions raised by these empirical results will be studied in full detail in our work. We already emphasized that the main advantages of the approach would be: *i*) a more elegant way of building hybrid (and more generally, multi-formalism) systems, and *ii*) increased possibilities for distributed (parallel)

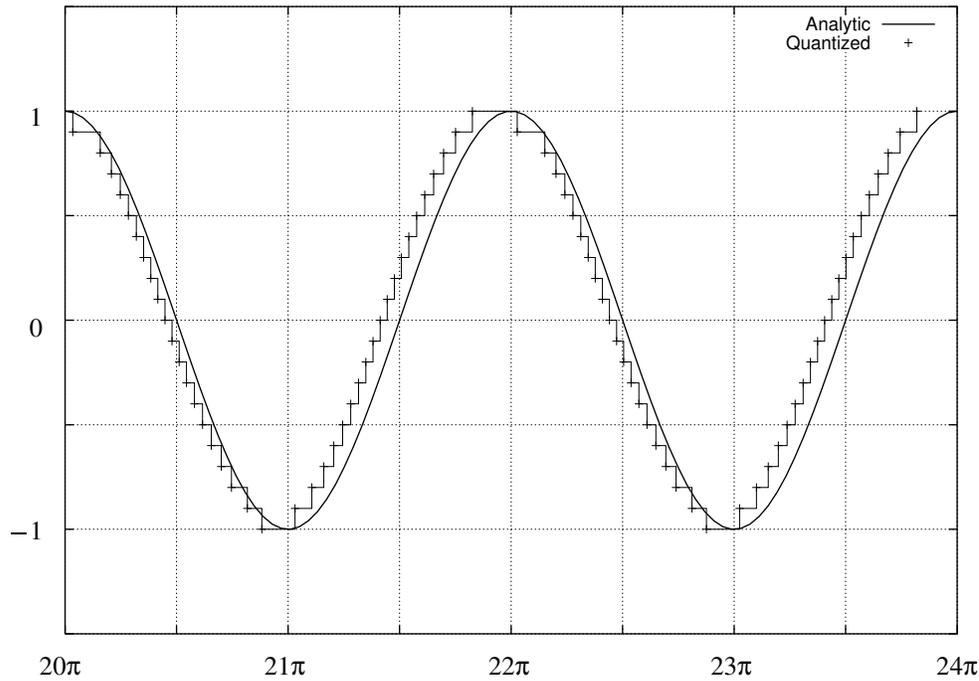


Figure 9: Circle Test Problem after $t = 20\pi$

implementations. We will keep these notions in mind through all steps of our work, as they will define the *context* of our central work.

For the quantization strategy to become an interesting alternative to traditional discretization methods, we will need to provide an elaborate analysis of the approach, as well as usable, efficient algorithms. We thus intend to iterate our work over different aspects pertaining to both the *theoretical aspects of mapping*, and the *implementation of a modelling and simulation package* (based on a sequential implementation of a P-DEVS simulator). These aspects are orthogonal to the context, and are detailed below.

5.1 Theoretical Aspects of Mapping

The study of theoretical aspects of mapping encompasses several key issues for the success of practical implementations. Zeigler *et al.* [12] have done some work in that direction, from defining the DEVS formalism and proving some of its properties, to showing the equivalence of quantized systems and DEVS. However, many points require further exploration, while some other important aspects have not been worked out. We propose to cover the following topics:

- Approximation schemes are validated through numerical analysis: establishing the true value of a method can only be done by applying it to various classes of systems, and then studying the stability and convergence properties of the quantized system that results. This is essential in determining the value of a method, and its field of application (if any). Also, by determining the weaknesses and strengths of a method, we can gain insights into how to improve or design new approaches. To our knowledge, no such rigorous analysis from a quantization perspective has been done yet. We observed in the case of the circle test problem above that the accumulation of errors results in a phase shift, whereas the range of the solution remains

bounded: we will have to further study this result in detail. The numerical analysis of stability and convergence (together with sensitivity analysis) of various schemes will be a crucial part of our work.

- In the last section we introduced a simple, Euler-like approximation method for the DEVS quantized integrator. Although the first empirical results are promising, we will need, in order to satisfy more severe accuracy requirements, to develop higher-order approximation schemes based on the Taylor expansion of $\mathbf{y}(t)$. We note that the Taylor expansion of a function of time t is a polynomial in some Δt : since we want to solve for this variable in quantization schemes (*i.e.*, when is the system going to transition to a given state), we will have to deal with implicit equations. It follows that developing higher-order approximations in the quantization context is not a simple, trivial generalization of those in the discretization context.
- By the closure property of DEVS, it is established that a network of quantized DEVS will be a valid system (under the legitimacy constraint). We have mentioned in section 4.2 the partition refinement requirement, which ensures that a coupled model of quantized systems can component-wise simulate its sub-models. However, this says nothing about the stability of the resulting model: by analogy, it is established that multi-rate integration can greatly impact on the stability of (tightly) coupled discretized models [4]. We thus plan to study the effects of various coupling patterns (*i.e.*, various quanta sizes and coupling densities) on the stability of the resulting system: this is of a particular importance when the error tolerance dictates quanta sizes that are incompatible with the partition refinement requirement (*e.g.*, a stiff coupled system composed of a fast subsystem with large quanta size upstream a slow subsystem with small quanta size).
- The obvious way to map a semi-explicit DAE onto a quantized DEVS would be to first preprocess the DAE (see section 2.1), and to map the resulting set of ODEs in the usual manner. An alternative approach might be considered, if the semi-explicit DAE is derived from a causal-block oriented formalism (*e.g.*, *Simulink*): it would then seem natural to perform a block-wise quantization and couple the resulting DEVS's as in the original model. However, as seen in section 3.2, the current (sufficient) requirements for legitimacy do not allow zero-delay components in the presence of quantized DEVS. We will be interested in finding less-restrictive conditions for legitimacy: *weak-legitimacy* would designate legitimate system where instantaneous transitions are possible. However, well-behavedness could still be compromised by algebraic-loops. We note that we can draw a parallel between the elimination of algebraic loops in the DEVS model, and the preprocessing of the DAE. We will investigate the possibility of performing loop-detection and resolution at the coupled-DEVS level.
- Since the quantized DEVS approach is in direct competition with traditional approaches, it is important to compare the two paradigms: we want to know which are the strengths and weaknesses of each when planning to perform a simulation. When comparing discretization schemes with one another, various metrics can be chosen, such as error or execution time. However, comparing one paradigm with the other might not be so obvious, as a metric in discretization might not have its direct equivalent in quantization (*e.g.*, a small error in timing might correspond to an important error in state space).

5.2 A Modelling and Simulation Package

The ideas developed above will be validated and tested in an actual implementation: “real-life” performance analyses will be conducted, where the quantization approach will be compared with

the traditional discretization approach, and with analytical solutions whenever possible.

We already developed at an earlier stage of our work a DEVS modelling and simulation package: the simulations in section 4.3 were conducted using this prototype, which provides a simple syntax to easily define atomic- and coupled-DEVS models. The *Python* scripting language was used to implement the classic DEVS formalism introduced in section 3. At the present stage of our work, we are ready to start the implementation of a (sequential) P-DEVS package; we will add on top of that the necessary libraries to deal specifically with quantized ODEs as our theoretical work progresses. These tools will allow us to test various optimization ideas:

- The idea of *adaptive step-size* often used with traditional integrators might be transposed to quantization as *adaptive quantas*, to satisfy precision-speed tradeoffs. A related idea involves the use of static, non-uniform quanta sizes. An expression for the error term, obtained from numerical analysis, will be required to evaluate the effect of coarseness.
- The optimization procedures such as the preprocessing of semi-explicit DAEs described in the previous section could be completed by conducting a dependency analysis of the resulting ODE with constraints, to determine appropriate clustering of equations into DEVS sub-models. This would minimize the number of messages sent at the coupled-DEVS level. The idea of clustering have an immediate impact on distributed implementations.
- In hybrid models, monitoring functions are used to trigger events when given conditions are satisfied by a model. These unscheduled events are often interpreted as a signal to change the dynamic *mode* of the system. Our extension libraries will support such requirements by allowing *variable-structure DEVS* to be created: this will be accomplished by sending the relevant events to a dedicated “supervisor” DEVS, responsible for activating and passivating the sub-models.
- In later stages of the prototype, two full-scale benchmark application will be used: on the one hand, Activated Sludge Waste Water Treatment Plant (WWTP) models for the optimal design and control of municipal WWTPs (see BIOMATH’s WEST++, www.hemmiswest.com). On the other hand, Rigid Body Dynamics models (RBD) as used for training simulators and state-of-the-art computer games (see MathEngine’s Dynamic Engine 2, www.mathengine.com). We emphasize that these will allow us to validate the DEVS formalism with both engineering and animation applications, where the focus is on precision and speed respectively.

References

- [1] U.M.Ascher and L.R.Petzold: *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, 1998.
- [2] A.C.-H.Chow: “Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator”. *Transactions of The Society for Computer Simulation International*, vol.13 no.2, pp. 55–67, June 1996.
- [3] E.Isaacson, H.B.Keller: *Analysis of Numerical Methods*. John Wiley & Sons, New York, 1966.
- [4] J.Laffitte and R.M.Howe: “Interfacing Fast and Slow Subsystems in the Real-Time Simulation of Dynamic Systems”. *Transactions of The Society for Computer Simulation International*, vol.14 no.3, pp. 116–126, March 1997.
- [5] T.Park and P.I.Barton: “State Event Location in Differential-Algebraic Models”. *ACM Transactions on Modeling and Computer Simulation*, vol.6 no.2, pp. 137–165, April 1996.

- [6] H.Praehofer: “System Theoretic Foundations for Combined Discrete-Continuous System Simulation”. Doctoral dissertations Johannes Kepler University of Linz, Linz, Austria, 1991.
- [7] O.Stursberg, S.Kowalewski and S.Engell: “On the Generation of Timed Discrete Approximations for Continuous Systems”. *Mathematical and Computer Modelling of Dynamical Systems*, vol.6 no.1, pp. 51–70, March 2000.
- [8] H.L.Vangheluwe and G.C.Vansteenkiste: “A multi-paradigm modeling and simulation methodology: Formalisms and languages”. *European Simulation Symposium (ESS)*, Society for Computer Simulation International (SCS), pp. 168–172, October 1996, Genoa, Italy.
- [9] H.L.Vangheluwe: “DEVS as a common denominator for multi-formalism hybrid systems modelling”. *IEEE International Symposium on Computer-Aided Control System Design*, IEEE Computer Society Press, pp. 129–134, September 2000, Anchorage, Alaska.
- [10] B.P.Zeigler: *Multifaceted Modelling and Discrete Event Simulation*. Academic Press, London, 1984.
- [11] B.P.Zeigler: *Theory of Modelling and Simulation*. Krieger Publishing Company, Malabar, FL, 1984.
- [12] B.P.Zeigler, H.Praehofer and T.G.Kim: *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, 2 ed.* Academic Press, New York, NY, 2000.

Appendix

In this appendix we present the actual code that defines a quantized integrator DEVS. We also include the code that describes how two such integrators are arranged to solve the circle test problem.

DEVS Quantized Integrator

```
class Integrator(AtomicDEVS):
    """ Solves for:  $dy/dt = c*x$ 
    Where y: OUTPUT,
           x: INPUT (slope)
           c: input GAIN (constant)
    Ouput is the quantized state of the integrator.
    """
    def __init__(self, d, c = 1., i = [0., 0.]):
        """ CONSTRUCTOR
        d: quanta size
        c: input gain
        i: initial state
        """
        # Always call parent class' constructor first:
        AtomicDEVS.__init__(self)
        # Make local copies of parameters:
        self.D = d
        self.c = c
```

```

# state is: (actual state, slope, quanta_repres)
# the last input received is stores as the slope
# fruntion 'repres' gives the representative
self.state = i
quant = repres(self.state[0], self.D)
if abs(quant - self.state[0]) == self.D/2.0: # Original state on
    if self.state[1] < 0: # quanta interface
        quant = quant - D
self.state.append(quant)
self.elapsed = 0.
# PORTS:
self.IN = self.addInPort()
self.OUT = self.addOutPort()

def extTransition(self):
    """External Transition Function.
    """
    Nstate = self.state[0] + (self.elapsed * self.state[1])
    Nslope = self.c * self.peek(self.IN)
    return [Nstate, Nslope, self.state[2]]

def intTransition(self):
    """Internal Transition Function.
    """
    Nstate = self.state[2] + sign(self.state[1]) * self.D/2.0
    Nquant = self.state[2]t + sign(self.state[1]) * self.D
    return [Nstate, self.state[1], Nquant]

def outputFnc(self):
    """Output Funtion.
    """
    Nout = self.state[2] + sign(self.state[1]) * self.D
    self.poke(self.OUT, Nout)

def timeAdvance(self):
    """Time-Advance Function.
    """
    if self.state[1] == 0:
        return 100000.0 # Stands for infinity
    Bound = self.state[2] + sign(self.state[1]) * self.D/2.0
    Dstate = abs(Bound - self.state[0])
    return Dstate / abs(self.state[1])

```

Coupled-DEVS for the Circle Test

```

class System(CoupledDEVS):
    """Couples two integrators in a loop
    """

```

```

def __init__(self, d1, d2 = 0):
    """CONSTRUCTOR
       d1 and d2 are the respective quanta sizes
       for the two integrators (typically identical)
    """
    # Always call parent class' constructor first:
    CoupledDEVS.__init__(self)
    # PORTS:
    self.OUT1 = self.addOutPort()
    self.OUT2 = self.addOutPort()
    # Validate quanta sizes
    if d2==0:
        d2 = d1
    # Initial conditions
    start = [1.0, 0.0]
    # Declare sub-models: Note that the 'Processor' atomic-DEVS
    # are simple 0-delay elements
    self.Int1 = self.addSubModel( Integrator(d1, 1.0, start) )
    self.Int2 = self.addSubModel( Integrator(d2, -1.0, [start[1], -start[0]]) )
    self.Del1 = self.addSubModel( Processor() )
    self.Del2 = self.addSubModel( Processor() )
    # Describe couplings:
    self.connectPorts(self.Int1.OUT, self.Del1.IN)
    self.connectPorts(self.Del1.OUT, self.Int2.IN)
    self.connectPorts(self.Int2.OUT, self.Del2.IN)
    self.connectPorts(self.Del2.OUT, self.Int1.IN)
    self.connectPorts(self.Int1.OUT, self.OUT1)
    self.connectPorts(self.Int2.OUT, self.OUT2)

def select(self, imm):
    """Select function.
       imm is the set of imminent components
    """
    return imm[0]

```