A Qualitative Comparison of Two Hybrid DEVS Approaches

Christina Deatcu*, Thorsten Pawletta

Wismar University of Applied Sciences, Res. Group Computational Engineering and Automation (CEA), PB 1210, 23952 Wismar, Germany; *christina.deatcu@hs-wismar.de

Abstract. A hybrid system is defined as a system with mixed discrete event and continuous parts. Two approaches for modeling and simulation of hybrid systems in the context of the Discrete EVent System Specification (DEVS) formalism are compared. Since the system theoretic DEVS approach has been introduced during the 1970ies, it has been completed by different extensions. The hybrid extensions checked against each other here, are the Quantized State Systems (QSS) method and a wrapper concept based on traditional ODE solvers. For a comparison from engineering practice point of view, these two hybrid DEVS approaches are analyzed within the scientific and technical computing environment MATLAB.

Introduction

There are several formal descriptions of system structures and system behaviors on hand to support the precise and complete mapping of real systems. This work is based on the Discrete EVent System Specification (DEVS) and its associated abstract simulator algorithms. We focus on applying DEVS-based approaches in the engineering field and in doing so try to find reasons why DEVS is relatively unknown and rarely used among engineers.

Initial research work on DEVS formalism was done in 1976 by Zeigler [1]. The DEVS formalism offers comprehensive options for describing systems that change their state according to events that appear at discrete instances of time. A main characteristic of simulation models defined according to the DEVS formalism is their modular hierarchical layout. In the decades that followed, the classic DEVS formalism was enhanced with extensions for hybrid, structure variable, parallel, real time and other system types to provide options for building models of a large range of diversity [2].

Two possible approaches for hybrid DEVS are discussed here. Hybrid systems in this context are systems with mixed discrete event and continuous system parts. It is generally accepted t hat modeling hybrid systems can be approached according to different worldviews. The first, the worldview of continuous systems, maps discrete events to zero-crossing problems. The entire model is computed by a continuous simulator, i.e. by an ODE solver with event detection and event localization. Approaching the issue of hybrid systems from the second, the discrete event worldview, requires the extension of discrete event formalism in a way that allows continuous model parts to be computed. This method is convenient in case the systems that need to be modeled are of a discrete event nature and include only minor continuous parts. The simulation can be controlled by a discrete event-based simulator that calls an ODE solver between event times. The Prerequisite is the availability of a closed representation of all continuous model parts as a differential equation system. However, the usual flattening of a modular hierarchical model to achieve a closed equation-oriented representation causes the loss of important model structure information during simulation runtime.

This contribution focuses on analyzing methods for hybrid system simulation within the DEVS formalism where model structure information is kept during simulation runtime. Keeping structure information is necessary to allow modular hierarchical systems with structure variability following [3, 4] to be investigated.
One way to integrate continuous model parts in the DEVS formalism without model flattening is the *wrapper concept* proposed here. Thereby, an extended DEVS simulator controls the overall simulation and cyclically calls an ODE solver between discrete event times. The so-called wrapper generates the required closed representation of the differential equations and in doing so does not affect structure information of the modular hierarchical DEVS model.

In contrast to the wrapper concept, the second approach discussed here, the Quantized State Systems (QSS) method solves the problem on the model level instead of on the simulator level. With the QSS method Kofman [5, 6, and 7] proposes a quantization of the state variables instead of the traditionally accepted discretization of time to solve a system of differential

SNE Simulation Notes Europe – Print ISSN 2305-9974 | Online ISSN 2306-0271
SNE 22(1), 2012, 15-24 | doi: 10.11128/sne.22.tn.10107

equations by numerical approximation. This approach allows asymptotic solutions by special DEVS model components. Adjusting the DEVS formalism itself is not necessary. Therefore, a hybrid model based on the QSS approach can be computed by any standard DEVS simulator.

In the following, after a short introduction to DEVS formalism, we discuss the two hybrid approaches in more depth. Based on the detailed examination, the suitability of the approaches for engineering problems is evaluated from the practitioner's point of view. Both approaches are studied by means of prototypical implementations within the scientific and technical computing environment (SCE) MATLAB. Numerical aspects like accuracy or stability problems of the examined methods are explicitly not subject to discussion.

1 Overview of DEVS

Models designed as formal DEVS models are composed in a modular hierarchical manner. They consist of two basic model types. The first type, in which dynamic behavior is defined, is the *atomic* DEVS model. The second type, the *coupled* DEVS model, describes a set of interacting components. These components are other coupled DEVS models or atomic DEVS. Beside model definitions, the DEVS formalism comprises data structures and algorithms that allow the hierarchical DEVS model to be simulated. For simulation purposes, a *simulator* is assigned to each atomic model and a *coordinator* is assigned to each coupled model as depicted in Figure 1. Simulators and coordinators are responsible for executing the model.

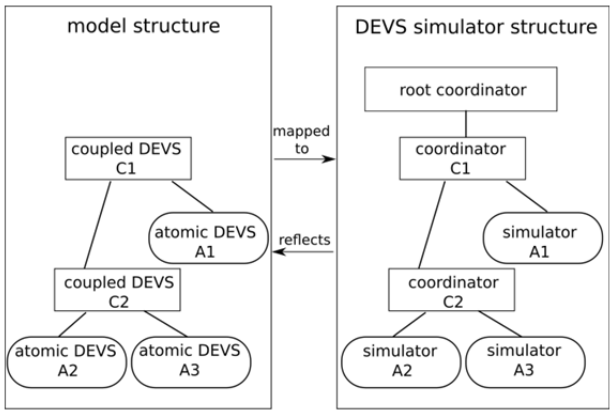

Figure 1. Model components and associated simulators / coordinators according to Zeigler.

The conjunction of a model component and a simulator or a coordinator is depicted as a *simulation object* below. The superordinate *root coordinator* initializes all simulation objects and starts and controls the simulation run. The root coordinator and the associated simulators and coordinators communicate with one another through messages.

1.1 Classic DEVS

A classic atomic DEVS A and a classic coupled DEVS N are defined as follows [2]:

$$A = (X, Y, S, \delta_{\text{int}}, \delta_{ext}, \lambda, ta) \quad (1)$$

$$N = (X_N, Y_N, D, \{M_d \mid d \in D\}, EIC, EOC, IC, Select) \quad (2)$$

An atomic DEVS A is described by the set of input events X, the set of output events Y, the set of states S, and by the four characteristic functions: internal transition function δ_{int}, external transition function δ_{ext}, output function λ and time advance function ta. The characteristic functions specify the dynamics of the atomic DEVS. Accordingly, sets of input and output events X_N and Y_N are defined for the coupled DEVS N. The set M_d specifies the subcomponents, which may be atomic or coupled DEVS. The set D is the corresponding index set.

The way in which subcomponents are connected is described by the coupling relations EIC (External Input Coupling), EOC (External Output Coupling) and IC (Internal Coupling). The $Select$ function controls model execution if simultaneous events appear. Hence, an atomic DEVS is responsible for describing of the dynamics of each component, whereas a coupled DEVS represents the composition of components.

1.2 PDEVS

To break up conflicts caused by simultaneous events solely at the level of atomic models, the classic DEVS formalism has been enhanced mainly by Chow and Zeigler to parallel DEVS (PDEVS) formalism since 1994. Atomic and coupled PDEVS are defined as follows:

$$A_P = (X, Y, S, \delta_{\text{int}}, \delta_{ext}, \delta_{conf}, \lambda, ta) \quad (3)$$

$$N_P = (X_N, Y_N, D, \{M_d \mid d \in D\}, EIC, EOC, IC) \quad (4)$$

In classic DEVS simultaneous events in subcomponents of a coupled system need a definition of priorities within the $Select$ function of the coupled model at superordinate level. The definition of priorities and therefore the $Select$ function can be omitted in PDEVS. Instead, the dynamic description of the atomic PDEVS is complemented by the confluent function δ_{conf}. This function is called, if for an atomic PDEVS A_P one or more external events and an internal event occur at the same time. Additionally, simulator and coordinator algorithms were comprehensively modified for executing PDEVS models.

1.3 DEVS with structure variability

Structure variability in the context of this work is defined as the possibility of modifying the hierarchical model structure during simulation runtime. It includes the creation, deletion and exchange of model components or the modification of coupling relations.

Barros [3] proposes the addition of a special atomic model called network executive to each coupled model which holds the structure information and describes the structure dynamics. In contrast to this approach, we favor extending the PDEVS coupled model definition in a way that allows holding structure dynamics specification directly in the coupled model according to [4].

This preference of keep structure information at the coupled model level is based on the principles of the primal DEVS theory where the atomic level specifies dynamics and the coupled level specifies structure issues. In both approaches the atomic model definitions remain unchanged.

We call the current composition of a coupled system, which means its set of subsystems and its coupling relations, a structure state $s_i \in S_N$. A structure variable PDEVS coupled model can have different structure states $s_0, s_1, ..., s_n \in S_N$. Furthermore, structure dynamics information, e.g. the number and kind of structure changes already achieved, need to be stored. The set of structural variables H_N holds this information. A structure state s_i, which is an element of the set of sequential structure states S_N is defined as follows:

$$s_i = (H_N, D, \{M_d \mid d \in D\}, EIC, EOC, IC) \quad (5)$$

The set of sequential structure states extends the formal definition of coupled PDEVS without structure variability so that structure variable PDEVS can be defined as follows:

$$N_{pdyn} = (X_N, Y_N, \{d_N\}, S_N, \delta_{N\,\mathrm{int}}, \delta_{Next}, \delta_{Nconf}, \lambda_N, ta_N) \quad (6)$$

It should be noted that coupling information is now encapsulated in the set of structure states S_N. The name of the coupled system is stored in d_N. Furthermore, the functions $\delta_{N\,\mathrm{int}}$, δ_{Next}, δ_{Nconf}, λ_N, and ta_N for coupled PDEVS systems are introduced to provide operations similar but not identical to atomic systems.

In analogy to event-oriented dynamic behavior of atomic PDEVS systems, dynamic structure changes in coupled DEVS are induced by events. The characteristic functions describe reactions on structure events. An application example for dynamic structure DEVS in engineering area that deals with structure and parameter optimization is given in [8].

2 Wrapper Concept

Scientific and technical computing environments (SCEs) such as the widely-used SCE Matlab offer a large number of established numerical methods for solving differential equations. The wrapper concept aims to make these methods applicable to hybrid, modular hierarchical DEVS models. A wrapper concept which is adapted to PDEVS is presented here. PDEVS is chosen because it offers a way of achieving an easy and realistic mapping of simultaneous events.

To keep structure information available and modifiable, the *flattening* which is typically done before computation of hybrid models must not be realized. This is a strong motivation for the wrapper concept. The issue of the structure variable PDEVS extension is sketched out here but not discussed in detail for reasons of clarity. However, it should be kept in mind that the wrapper approach is applicable to static structure as well as to structure variable models.

2.1 Formal Model Definitions

A hybrid classic atomic DEVS was defined for the first time in the 1990ies by Prähofer [9]. To allow computation of hybrid atomic DEVS Prähofer added the explicit

Euler method to the DEVS simulator.

Advanced numerical methods as they are common in engineering applications like implicit integration algorithms, predictor corrector methods, or ODE solvers with adaptive step size cannot be used. The wrapper concept starts from Prähofer's formal hybrid atomic DEVS definition and adapts it to PDEVS formalism.

For the wrapper concept, just the formal model specification resembles Prähofer's definition, whereas the simulation approach and algorithms are completely different. We define a hybrid atomic PDEVS as follows:

$$A_{hybridp} = \left(X, Y, S, f, c_{se}, \lambda_c, \delta_{state}, \delta_{\text{int}}, \delta_{ext}, \delta_{conf}, \lambda_d, ta\right) \quad (7)$$

The sets of inputs X, outputs Y and states S of hybrid DEVS may contain discrete as well as continuous values. The continuous dynamics of a component is mapped by the rate of change function f and the continuous output function λ_c. Discrete events can be external or internal events, or can be state events caused by discontinuities in the continuous parts. The state event condition function c_{se} defines the conditions under which a state event is generated.

External events, internal events, and state events alter the system's state stored in S. External events achieve this via the external transition function δ_{ext} while state events are computed within the state event transition function δ_{state}. Internal events activate the discrete output function λ_d first and subsequently the internal transition function δ_{int}, which then alters the system's state by manipulating S.

For hybrid PDEVS not just external and internal events may occur simultaneously. So can state events in continuous variables with discontinuities detected by the ODE solver. Therefore, the confluent function δ_{conf} needs to be adapted to handle three different types of potentially simultaneous events. After any change of state the time advance function ta calculates the time interval until the next internal event.

The specification for coupled models needs to be modified marginally. A coupled structure variable PDEVS N_{pdyn} following the specification in equation (6) is suitable for hybrid models after extending its δ_{Next} function in a way that it can react on state events occurring in continuous parts of submodels, too. Furthermore, coupling relations encapsulated in the set of structure states S_N need to be differentiated between discrete and continuous relations. In addition, the sets of inputs X_N and outputs Y_N may contain discrete as well as continuous values.

2.2 Extended Simulator Algorithms and Data Structures

In a hybrid, modular hierarchical model, continuous model parts are typically distributed over several model components. To allow the deployment of advanced numerical integration methods for computation without a previous model flattening, these distributed continuous model parts need to be collected, united and provided in a way suitable for the chosen ODE solver's interface. The *ODE wrapper* function supplies the closed representation of continuous model parts required by the ODE solver. Coevally, structure information remains available. The modular hierarchical model itself is not modified. Rather, a closed representation of continuous model parts is generated additionally during simulation runtime.

Figure 2 and Listing 1 show how the DEVS simulation environment is extended by adequate data structures, an adapted *root coordinator*, and the *ODE wrapper* function.

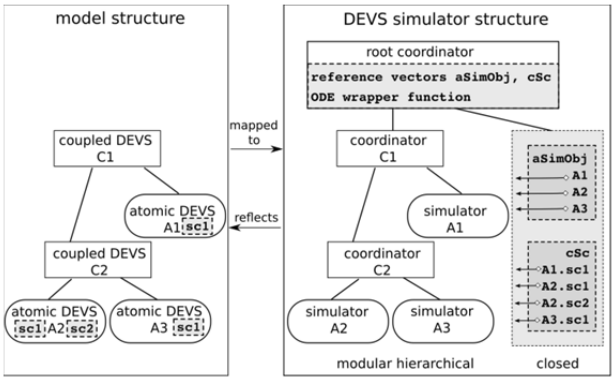

Figure 2. Modular hierarchical model structure and simulator structure with wrapper components.

The *root coordinator* needs to be extended to operate in three different phases: i) initialization phase, ii) discrete phase, and iii) continuous phase.

```
begin root coordinator

set current simulation time to start time
set phase to 0 # start the initialization phase

while simulation time not over

 switch phase

  ----------------------------------------------------

  case 0  # initialization phase
  # determine time of first internal event
  tnext <- send i-message to outermost coupled

  if tnext == current simulation time
   set phase to 1 # start a discrete phase
  elseif tnext > current sim. time
   set phase to 2 # start a continuous phase
  end

  ----------------------------------------------------

  case 1   # discrete phase
  # carry out internal event and
  # determine time of next internal event
  tnext <- send *-message to outermost coupled

  if tnext == current sim. time
   set phase to 1 # remain in discrete phase
  elseif tnext > current sim. time
   set phase to 2 # start a continuous phase
  end

  ----------------------------------------------------

  case 2   # continuous phase
  # establish references to atomic sim. objects and
  # cont. state variables in vectors aSimObj and cSc
  [aSimObj, cSc] <- send z-message to outermost coupled

  # establish direct links between outputs and inputs
  send z2-message to outermost coupled

  sort vector aSimObj depending on moore/mealy behavior

  # specify ODE-wrapper with flag=0 as event function
  set options for ODE-solver

  # call ODE-solver with ODE-wrapper, flag=1,
  # as model function
  # integration interval:
  # [current sim. time, tnext or tfinal]
  [times,values,events] <-
        ODE-solver(ODE-wrapper(1,aSimObj,cSc),interval)

  # set current sim. time to biggest value in times
  current sim. time = max(times)

  if ODE-solver has returned state events
   # send state-event-message to
   # localize atomic obj. where event has occured.
   # simulator of this atomic calls its deltastate func.
   # determine time of next internal event
   tnext <- send se-message to outermost coupled
  end

  if tnext == current simulation time
   set phase to 1 # start a discrete phase
  else
   set phase to 2 # remain in continuous phase
  end

  ----------------------------------------------------

 end switch

end while

end root coordinator
```

Listing 1. Root coordinator algorithm.

In addition to the well-known messages from traditional DEVS simulators, i-message, *-message, x-message, and y-message, we introduce a z- and a z2-message for vector configuration purposes and an se-message for handling state events of the continuous model parts. For exact mapping of simultaneous events we use an interpellation y-message as proposed by [10].

At startup of a simulation run, the initialization phase is entered and the model is initialized by an initialization message (i-message) sent by the *root coordinator* to the outermost coupled model. The initialization message is passed down the hierarchy among the simulation objects until it reaches the leaves of the object tree.

Based on the minimum of time stamps for the next internal events (as they are returned by the subcomponents), the *root coordinator* determines whether a discrete or a continuous simulation phase has to be started next. If there is no imminent internal event at current simulation time, a continuous cycle is initiated and starts with an update of the wrapper data structures. This step is necessary for modular hierarchical systems with structure variability at the beginning of each continuous phase because the model structure may have changed due to a previous event.

For this purpose the *root coordinator* sends recursive configuration messages (z- and z2-message) to the associated coordinator of the outermost coupled model. Those messages are passed down the hierarchy tree depicted in Figure 2. References to all hybrid atomic simulation objects (`aSimObj`) as well as direct references to all continuous state variables contained (`cSc`) are returned. Furthermore, direct links between continuous output and input variables are established by interpreting the coupling relations. This linking information is written back to atomic simulation objects with the z2-message.

The behavior of atomic models containing continuous variables resembles either Moore automata $\lambda_c : S_c \rightarrow Y_c$ or Mealy automata $\lambda_c : S_c \times X_c \rightarrow Y_c$. Depending on the type of atomic models to which the continuous variables belong, the references to the atomic simulation objects in `aSimObj` are divided into two groups in a first step. As the next task, inside of the mealy group, references need to be sorted based on interdependencies. The sorted reference vectors with objects of mealy and moore type are passed to the *ODE wrapper* function.

The *ODE wrapper* function is handed over as the closed representation of all continuous model parts to the ODE solver. Via the references to the atomic simulation objects the hybrid atomic model functions λ_c, f and c_{se} can be called by the *ODE wrapper*. For each integration step the *ODE wrapper* calls the continuous output functions λ_c and the rate of change functions f of all hybrid atomic subcomponents referenced in vector `aSimObj` to calculate continuous outputs and derivatives. Additionally, the state event condition function c_{se} of each subcomponent referenced in `aSimObj` is evaluated to check for state events of continuous values that may have occurred. The *ODE wrapper* distinguishes between the calls of λ_c and f or the call of c_{se} by evaluating the flag parameter.

The continuous cycle is iterated as long as either the next discrete internal event becomes imminent or the cycle is interrupted by a state event. If the ODE solver returns a state event, an se-message is passed down the tree to find out which atomic subcomponent is affected. The simulator of this atomic then calls the associated state event transition function δ_{state}. Based on the time for next internal discrete event returned by the se-message, the *root coordinator* decides whether to start a discrete or a continuous phase next.

A prototype of an advanced PDEVS simulation environment including the wrapper algorithms sketched out has been implemented in the SCE Matlab. It could be proven that, except the `ode15i` for fully implicit differential equations, the entire set of ODE solvers provided by the SCE Matlab, including variable-step solvers as e.g. `ode45` and implicit solvers as e.g. `ode15s`, can be used without modifications for simulating hybrid PDEVS systems with structure variability. A consistent separation of model definitions and numerical solver algorithms is also ensured.

3 Quantized State Systems

Zeigler and Lee defined Quantized Systems (QSS) and their representation in conjunction with DEVS models in the 90ies [11]. The QSS method was improved by Kofman and Junco [5] and offers a completely new innovative approach to numerically solve differential equations.

This approach does not need to be implemented within DEVS necessarily, but it is closely connected to this formalism from the beginning. In contrast to conventional methods for numerically calculating differential equations, time is not discretized within the QSS method, but the states. That point in time is calculated where the continuous state variable has changed more than a certain quantum D.

According to [12] the QSS method is applicable to any category of differential equation systems. Based on selected examples, in [7] Kofman compares solutions obtained by the QSS methods QSS1 and QSS2 with solutions calculated by conventional ODE solvers of the commercial SCE Matlab/Simulink. For the examples studied, he states a considerably better performance of the QSS solutions. The basic approach of the QSS method is introduced below with references to the advanced QSS methods.

3.1 QSS1

For one ordinary differential equation given as $\dot{x}(t) = f(x(t))$ Nutaro [13] specifies the following simulation algorithm, which is independent of the DEVS formalism:

$t \leftarrow 0$
$x \leftarrow x0$
`while terminating condition not fulfilled`
  `print` t, x
  `if` $f(x) = 0$ `then`
    $h \leftarrow \infty$
  `else`
    $h \leftarrow \frac{D}{|f(x)|}$
  `end if`
  `if` $h = \infty$ `then`
    `stop simulation`
  `else`
    $t \leftarrow t + h$
    $x \leftarrow x + D * \mathrm{sgn}(f(x))$
  `end if`
`end while`

D is the value of the quantum. The step size h is variable and depends on the rate of change $f(x(t))$. Thus, h defines the time interval after which the system's state variable x has changed by D. The state values x result as the sum of the start value $x0$ and a multiple k of the quantum D. Thereby, the discrete character of the approximatively calculated state variable becomes apparent.

If not just one differential equation $\dot{x}(t) = f(x(t))$ but a system of differential equations $\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t))$ is computed, the derivatives of all state variables dependent on x_i need to be recalculated at time $t_{x_i} + h_{x_i}$. This modus operandi basically remains the same – also for the advanced QSS algorithms.

3.2 Advanced QSS Algorithms

[6, 7] introduce a wide variety of advanced QSS methods. Second order QSS (QSS2) applies a different quantization (*First Order Quantizer*) to the continuous state variables. The quantized state variable is not constant between two integration steps as it is for QSS1 but follows a linear function. Thereby, the number of event times required for computing a differential equation system is reduced.

Furthermore, there are several implicit QSS methods. The Backward QSS (BQSS) method is designed especially for stiff systems. A combination of BQSS and QSS the Centered QSS (CQSS) is suitable for computating permanently oscillating systems. The Linearly Implicit QSS (LIQSS) are an enhancement of BQSS.

3.3 QSS in DEVS

To permit computation of differential equation systems using QSS in a DEVS environment, the QSS algorithms (QSS integrators) need to be mapped to atomic DEVS models. As far as model design is concerned, this means that one integrator and thus one atomic DEVS are required for each continuous state variable. Detailed specifications for DEVS-based QSS integrators are published in [5, 12, and 13]. Furthermore, each differential equation needs to be mapped to a separate atomic DEVS model.

The consequence of this approach is a rather complex model structure. Figure 3 illustrates the basic model structure of a DEVS model for simulating a differential equation system with two interconnected state variables using the QSS1 method. The atomic DEVS of a differential equation in the case of QSS1 employment receives the quantized variables q_i as external events from the associated QSS1 integrators. The δ_{ext} and ta functions of the atomic DEVS of the ODE then generate an instantaneous internal event. Thus, its output function λ is immediately executed and computes the actual rate of change value $\dot{x}_i$ and sends it as an output event y_i to the connected atomic DEVS coding the QSS integrator.

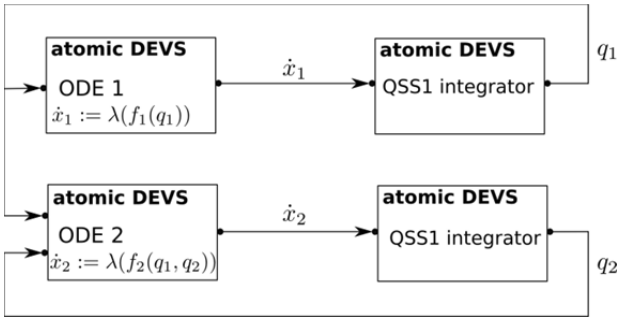

Figure 3. DEVS model structure of a differential equation system with two state variables and associated QSS1 integrators.

Figure 4 shows the basic DEVS model structure of the same ODE system as it needs to be designed, if you aim to use the QSS2 method for computation. According to that not only the atomic DEVS models of the QSS integrators have to be exchanged. The atomic DEVS specifying the differential equations need to be adapted, too. It is essential that the atomic DEVS of the differential equations and the atomic DEVS of the QSS integrators are attuned to one another.

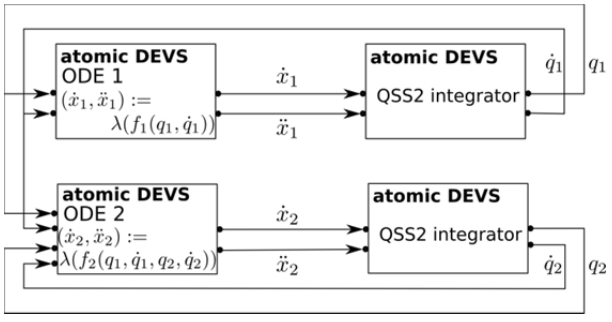

Figure 4. DEVS model structure of a differential equation system with two state variables and associated QSS2 integrators.

Hence, before specifying the atomic DEVS models for the differential equations it needs to be known which QSS method will be used for computation [12]. At this point it becomes clear that there is no strict separation between model description and numerical processing for the QSS method.

The discussed examples are pure continuous systems. But of course, the quantization of the continuous variables and the discrete event-oriented computation allows a seamless specification of hybrid systems too. Event detection and localization functionality has to be implemented in atomic DEVS of the QSS integrators or added as separate atomic DEVS models to the system. The model structure information is kept during simulation runtime.

For software engineering practice, purpose-made templates of e.g. atomic DEVS models for specification of differential equations can be stored in a model library. The modeler simply takes a template file and adapts the concrete algorithm for the derivative in the output function. QSS method specific integrators can be stored in the model library as predefined atomic DEVS models as well. Here, the user can apply those methods without detailed knowledge of the QSS algorithms.

An example of a DEVS-based model library and simulation environment that offers support for QSS based hybrid models is the *PowerDEVS* software suite by E. Kofman [14], which is integrated into the freely available SCE *Scilab* [15].

4 Qualitative Comparison

A key aspect of the comparison subsumed in Table 1 is the quality of integration of the DEVS formalism for hybrid systems into a SCE like Matlab and the resulting potential for engineering application area from user's point of view. Both approaches ensure structure information of the modular hierarchical model during runtime is completely conserved and are therefore both suited for computating structure variable problems in engineering applications.

The QSS approach incorporates the numerical integration algorithms directly into the model description. Model components specifying the differential equations cannot be implemented independently from the integration algorithms the analyst has chosen. In contrast, the wrapper concept is based on a strict separation of model and solution algorithm. Furthermore, the wrapper concept allows the straight, unadapted use of the established ODE solvers provided by the selected SCE.

.

	Wrapper Concept	**Quantized State Systems**
Preservation of structure information	yes	yes
Computation of continuous solution	at simulator level	at model level
Adapted DEVS simulator algorithms	yes	no
Usage of established numerical integration algorithms	yes	no
Event detection and localization	yes (performed by ODE solver)	yes (bounded to quant. level)
Model complexity	lower	higher
Reusability of models	yes	yes
Reality equivalence of models	very high	high
Numerical characteristics	not studied	not studied
Runtime / performance	not studied	not studied

Table 1. Application oriented comparison of two hybrid DEVS approaches.

With the wrapper concept and its prototypal implementation for MATLAB, using varying solvers to execute the same hybrid model is effortless. As all Matlab ODE solvers, except the `ode15i` for fully implicit differential equations, function with the same interface - just one line of code within the *root coordinator* has to be modified.

Due to the fact that the wrapper concept requires separate data structures, additional messages and an interface to the ODE solver, extended DEVS simulator algorithms need to be implemented. In contrast, QSS models can be computed by the standard DEVS simulator algorithms.

Event detection and localization is an essential task in hybrid system simulation. The wrapper concept delegates this task to the ODE solvers of the SCE, while, if following the QSS approach, the atomic models of QSS integrators either need to be extended or additional atomic DEVS models for event detection and localization have to be added to the model. Event detection and localization are not integral capabilities of QSS integration algorithms but can be implemented as extensions to handle events within the bounds of quantization levels.

Continuous model parts designed for computation following the QSS approach consist of at least two atomic models per state variable: One atomic DEVS to specify the differential equation and another atomic DEVS implementing the QSS integration algorithm.

The wrapper concept allows more than one continuous state variable per atomic DEVS model to be defined. Moreover, additional discrete functionality can be added to the same atomic DEVS model. Hence, model structure of models following the wrapper concept appears to be more compact. Model complexity of QSS models compared to models following the wrapper concept increases unavoidably since integration algorithms and event detection and localization methods become part of the model.

However, both methods provide an excellent setting for the modeling and simulating hybrid systems whose modular hierarchical structure may change during simulation runtime. Modeling problems of this type is elaborate and difficult to impossible using standard simulation tools in SCEs such as the toolboxes Simulink and SimEvents for the SCE Matlab.

Reusability of models defined once is given for both approaches. The modular hierarchical characteristic of the DEVS formalism supports this feature per se.

In discrete event modeling and simulation for engineering applications as e.g. modeling of manufacturing systems, it is common practice to represent objects of the real world, e.g. work pieces, as entities [16]. One single atomic hybrid DEVS according to the wrapper concept can represent such an entity, including its entire discrete event behavior, and its continuous dynamic behavior. Modeling the same characteristics using the QSS approach requires more than one atomic DEVS.

The wrapper concept has benefits with respect to the degree of reality equivalence of the models over the QSS approach. For the wrapper concept one work piece can be represented by one atomic DEVS model. Thus, one element in the model equates one element in reality. To improve reality equivalence for QSS models, the multiple atomic DEVS components needed to model one work piece could be combined to become a coupled DEVS model. In this case, an 1-to-1-mapping is achieved from the user perspective.

Detailed studies regarding numerical aspects, performance and runtime of both approaches within a SCE have not yet been carried out.

5 Summary

Both approaches offer the potential to make the DEVS formalism for modeling and simulation of hybrid modular hierarchical systems accessible to engineers and support them in modeling and simulating their hybrid engineering applications more effectively. From an academic point of view, the wrapper concept and QSS both lead to comparable, correct results. But today, the DEVS formalism and the QSS method are fairly unknown in the engineering community and are rarely applied to real engineering problems.

Especially for the QSS approach this may be caused by the fact that people like to continue using tools and algorithms with which they have been familiar for years and intuitively refuse to use unknown concepts. An engineer who is familiar with traditional integration algorithms could maybe be convinced more easily to try out the wrapper approach in which he only has to study one new concept, namely the DEVS theory.

The main benefits of the wrapper concept are the integration of established numerical integration methods as they are familiar to engineers, and the consequent separation of the model from the simulation or rather integration algorithms.

The QSS approach is a methodically new approach. It allows the further use of the existing DEVS simulation algorithms for computation of models of hybrid systems. The lack of acceptance for the QSS approach among engineers is probably also related to the lack of available implementations of the integration algorithms for commercial SCEs like Matlab. Presently, QSS implementations such as the PowerDEVS suite [14] are only available for the non-commercial SCE Scilab or implemented as libraries for other simulation environments such as *ModelicaDEVS* [17]. Acceptance of the DEVS formalism itself as well as the acceptance of the QSS method or the wrapper concept is obviously linked to the availability of adequate tools.

References

[1] B. P. Zeigler. *Theory of Modeling and Simulation (first ed.)*. Wiley Interscience, New York, 1976.

[2] B. Zeigler, H. Praehofer and T. G. Kim. *Theory of Modeling and Simulation (second ed.)*. Academic Press, New York, 2000.

[3] F. Barros. *The Dynamic Structure Discrete Event System Specification Formalism*. Trans. Soc. Comput. Simul. Int. 13:35–46, 1996

[4] T. Pawletta, B. Lampe, S. Pawletta and W. Drewelow. *A DEVS-Based Approach for Modeling and Simulation of Hybrid Variable Structure Systems*. In Modelling, Analysis, and Design of Hybrid Systems, edited by G. F. S. Engell and E. Schnieder, 107–129. Springer Verlag, 2002.

[5] E. Kofman, S. Junco. *Quantized-State Systems: a DEVS Approach for Continuous System Simulation*. Trans. Soc. Comput. Simul. Int. 18, 123–132, 2001.

[6] E. Kofman. *Discrete Event Systems and Control of Continuous Systems*. Phd thesis, Universidad Nacional de Rosario, Argentinien, 2003.

[7] E. Kofman. *Discrete Event Simulation of Hybrid Systems*. SIAM Journal on Scientific Computing, 25(5), 1771–1797, 2004. doi:10.1137/S1064827502418379

[8] O. Hagendorf and T. Pawletta. *A Framework for Simulation Based Structure and Parameter Optimization of Discrete Event Systems*. In: Discrete-Event Modeling and Simulation: Theory and Applications, Editors: G.A. Wainer and P.J. Mosterman, CRC Press Inc. of Tailor & Francis Group, USA, 199-222, 2011.

[9] H. Prähofer. *System Theoretic Foundations for Combined Discrete-Continuous System Simulation*. Phd thesis, VWG, Vienna, Austria, 1992.

[10] T. Schwatinski and T. Pawletta. *An Advanced Simulation Approach for Parallel DEVS with Ports*. In Proceedings of Spring Simulation Multiconference 2010, Book 4 - Symposium on Theory of Modeling & Simulation, 132-139, 2010.

[11] B. P. Zeigler and J. Lee. *Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simualtion Environment*. In SPIE proceedings, 49–58, 1998.

[12] F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer Verlag, 2006.

[13] J. Nutaro, J. *Discrete Event Simulation of Continuous Systems*. In Handbook of Dynamic System Modeling, edited by P. A. Fishwick, 11–1–11–23. Chapman & Hall/CRC. 2007.

[14] PowerDEVS by E. Kofman. *PowerDEVS Website*. Accessed Apr. 2, 2012. http://www.fceia.unr.edu.ar/lsd/powerdevs/. 2003.

[15] Scilab by INRIA. *Scilab Website*. Accessed Apr. 2, 2012. http://www.scilab.org. 2012

[16] C. Deatcu, T. Pawletta, O. Hagendorf, and B. Lampe. *ConsideringWorkpieces as Integral Parts of a DEVS Model*. In Proceedings of 21st European Modeling & Simulation Symposium, Volume 1, 27–35. 2009.

[17] ModelicaDEVS by T. Beltrame. *ModelicaDEVS at Modelica Association Website*. Accessed Apr.2, 2012. https://www.modelica.org/libraries/ModelicaDEVS

Submitted: October 2011
Revised: January 2012
Accepted: March 17, 2012