Int. J. General Systems, Vol. 19, pp. 219–240
Reprints available directly from the publisher
Photocopying permitted by license only

SYSTEM THEORETIC FORMALISMS FOR COMBINED DISCRETE-CONTINUOUS SYSTEM SIMULATION

HERBERT PRAEHOFER

Department of Systems Theory and Information Engineering, Institute of Systems Sciences, Johannes Kepler University of Linz, A-4040 Linz, Austria

(Received September 1990; in final form December 1990)

In this paper we present an approach to combined discrete-continuous modelling which can be used to model and simulate an intelligent multi-layer control architecture as can be found in high autonomy systems. The modelling approach is based on system theoretical concepts; the three system specification formalisms—differential equation, discrete time, and the discrete event system specification formalism—have been combined to facilitate multi-formalisms modelling. Simulation concepts are based on the abstract simulator concept for discrete event simulation developed by Zeigler. Similar simulation methods have been developed to simulate modular, hierarchical discrete time and differential equation specified systems as well as multi-formalism models. Included in the paper are several examples of multi-formalism models together with the simulation results from the STIMS environment—an implementation of the modelling and simulation concepts in Interlisp-D/LOOPS.

INDEX TERMS: Systems theory, combined discrete-continuous simulation, multi-formalism models, event-based control.

1. INTRODUCTION

High autonomy systems as defined by NASA[1] are artifacts which have the ability to function as independent units over an extended period of time and to perform a variety of actions to achieve predesignated objectives. To fulfill their tasks they have to accept a variety of stimuli produced by integrally contained sensors and, in response to them, they have to effect the environment by a variety of actions in pursuit of the given goal.[2]

According to this definition, a high autonomy system can be understood as a control system where first we have perception, then control, and finally action. The new paradigm of intelligent control[3–5] which tries to incorporate techniques emerged from artificial intelligence into classical control seems to be a promising technology to achieve high autonomy.[2] The intelligent control paradigm uses a hierarchy of layers for a control system which reflects the increasing intelligence with decreasing precision. Saridis[5] distinguishes the execution layer, the coordination layer, and the management layer. The execution layer contains the system to control—the sensors, actuators and effectors. The coordination layer receives the sensor values from the lower layers as well as the high level commands from the management layer and generates sequences of action commands for the lower layer. The management layer now uses artificial intelligence techniques to make the decisions to fulfill tasks in pursuit of varying objectives.

To model and simulate such a layered control architecture different models have to be employed.[2] Lower layers more likely will use differential or difference equation specified models while the higher layers will require symbolic modelling paradigms. Antsaklis[4] proposes a so-called hybrid modelling methodology therefor.

In this paper we will outline an approach to address this issue. We will present an approach to integrate the traditional differential and difference equation specified system formalisms and the newly developed discrete event system (DEVS)[6,7] formalism. In section 2 we will review the traditional differential and difference equation as well as the discrete event modelling paradigm. We will do this from a systems theory point of view. Then in section 3 we will introduce several new systems specification formalisms for combined discrete-continuous modelling. In section 4 simulation concepts for the different types of systems which are based on the abstract simulator for DEVS[7] will be presented.

In section 5 we will demonstrate our modelling and simulation concepts by an example of a combined event-based and analog motor speed control system. The event-based control paradigm is based on the DEVS formalism and has been introduced by Zeigler.[8,9] In this new approach to control, the controller receives threshold sensor inputs from the continuous process. The threshold sensor inputs of the new event-based control paradigm may arrive at arbitrary time instances, indicating the time of event when the threshold sensor flips from one state to another. The event-based controller then reacts whenever such an event input occurs and exerts commands to the process to drive it to a desired goal state. An event-based controller uses time windows to determine if a process run is correct or a failure has occurred. If the reaction of the threshold sensor lies within a predefined time window, it is assumed that the process operation is correct and the next control command can be exerted. Otherwise the process operation is regarded as faulty and corrective action has to be undertaken.

In our example a conventional analog controller has the task to bring the motor to different nominal speeds. An event-based controller issues the nominal speeds as well as different work loads and supervises the analog control process. While the motor and the analog control system will be modeled by a differential equation specified system, the event-based controller will be modeled by a discrete event model. The model has been implemented using the STIMS modelling and simulation environment. STIMS is a prototypic implementation of the modelling and simulation concepts discussed in section 2, 3 and 4 in Interlisp-D/LOOPS[10] and runs on the SIEMENS 5815 (XEROX 1108) workstations. Simulation results will also be included in section 5.

2. A BRIEF REVIEW OF SYSTEM FORMALISMS

In this section we review the traditional system specification formalism for simulation modelling—the *differential equation specified system* (DESS) and the *discrete time specified system* (DTSS) formalism—as well as the system specification formalism for event oriented simulation modelling introduced by Zeigler—the *discrete event system specification* (DEVS) formalism.[6,7] We will do this review from two points of view—first from a systems theory point of view and second from a simulation point of view. In particular, we will take the perspective of Zeigler[6,11] and Ören.[12,13]

	Differential Equations	Discrete Event	Discrete Time
time base T	real numbers	real numbers	isomorphic to integers
basic sets X, Y, Q	real vector spaces	arbitrary	arbitrary
input segments	piecewise continuous segments	discrete event segments	sequences
state and output trajectories	continuous segments	piecewise constant segments	sequences

Figure 1 Constraints imposed by modelling formalisms

A formalism specifies a *Dynamic System*[14,15] and it implies restrictions on the elements of the dynamic system description. Formalisms build subclasses of systems. Figure 1 shows the constraints[6,7,11] imposed on our modelling formalisms. Most important is that each of these formalisms is closed under coupling, i.e., one can couple systems together so that the resultant again specifies a system.[7] Thus, coupled systems can in turn be coupled together and hence the construction of modular, hierarchical models is possible.

2.1. Differential Equation Specified Systems Formalism

An atomic differential equation specified system (DESS) is a structure[6,12]

$$\text{DESS} = (X, Y, Q, f, \lambda)$$

where

X	the set of inputs is a real vector space
Y	the set of outputs is a real vector space
Q	the set of states is a real vector space
$f: Q \times X \rightarrow Q$	is the rate of change function
$\lambda : Q \rightarrow Y$	is the output function

The time base of DESS are the real numbers. The input segments for a DESS have to be piecewise continuous segments. The output segment produced by DESS as response are continuous segments over Y and the state segments are continuous segments over Q.

The output value y_t produced by the DESS at any time t where s_t is the state at time t is given by the function $y_t := \lambda(s_t)$. A state trajectory *straj* over $(Q, \mathfrak{R})$ of a DESS for a given input segment $\omega \in (X, \mathfrak{R})$ has to satisfy at any point in time t of the domain of ω the condition imposed by the differential equation $d\,straj(t)/dt := f(straj(t), \omega(t))$.

2.2. Discrete Time System Formalism

An atomic discrete time system (DTSS) is a structure[6,12]

$$\text{DTSS} = (X, Y, Q, \delta, \lambda)$$

where

X	is the set of inputs
Y	is the set of outputs
S	is the set of states.
$\delta : Q \times X \rightarrow Q$	is the state transition function
λ	is the output function

and λ is a function either

$$\lambda : Q \rightarrow Y$$

in the case of *Moore* type components or

$$\lambda : Q \times X \rightarrow Y$$

in the case of *Mealy* type components.

With atomic DTSS we associate the following dynamic behavior: As time base of DTSS we take the integers $\Im$ ($= 1 * \Im$) or a set $ta * \Im$ isomorphic to the integers where ta is a constant time advance. When a DTSS is in state s_t at time t and the input at time t is x_t, then it gives out the value $y_t := \lambda(s_t)$ or $\lambda(s_t, x_t)$ in the case of Moore and Mealy type components respectively and transits to state $s_{t+ta} := \delta(s_t, x_t)$. s_{t+ta} is the state at time $t + ta$.

2.3. Discrete Event System Formalism

An atomic discrete event systems (DEVS) is a structure[6,7]

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

where

X	is the set of external events
Y	is the set of outputs
S	is the set of sequential states
$\delta_{ext} : Q \times X \rightarrow S$	is the external transition function
$\delta_{int} : S \rightarrow S$	is the internal transition function
$\lambda : S \rightarrow Y$	is the output function
$ta : S \rightarrow \Re_0^+ \cup \infty$	is the time advance function

and

$$Q := \{(s, e) \mid s \in S,\ 0 <= e <= ta(s)\} \text{ is the set of } \textit{total states}.$$

Discrete event systems have a continuous time base but inputs and state transitions occur at discrete time instances. So input segments are constrained to so-called *DEVS segments,* i.e., for a finite number of points in time there are inputs and between these times there is no input (or the so-called *non-event*). State and output trajectories are piecewise constant segments. The set of inputs, outputs and states are arbitrary sets. The state transition consists of two parts—the external transition function is executed whenever an input arrives and the times of execution of the internal transition function are scheduled by the time advance function.

If at time t the DEVS has been in state s for $e \leq ta(s)$ time units, then the DEVS is scheduled to undertake its internal transition at time $tn = t + (ta(s) - e)$. When there will arrive no external event until time tn, then the DEVS will stay in state s until that time so that it has been in s exactly $ta(s)$ time units and then will transit from s to $s' := \delta_{int}(s)$ at time tn. Right before undertaking the internal transition, the DEVS will put out the value $\lambda(s)$. After the internal transition, the DEVS is scheduled for the next internal transition at time $tn + ta(s')$.

Input values can arrive at arbitrary time instances at the input ports. If at time t the DEVS has been in state s for time $e \leq ta(s)$ and there arrives an external input, then the DEVS has to execute its external transition function and the DEVS will transit from s to $s'' := \delta_{ext}(s, e, x)$ where it will remain either to the next external event or $ta(s'')$ time units.

Well defined DEVS must have the property of *legitimacy*. This property prevents the DEVS from getting into an infinite sequence of states in which the clock would not advance beyond a certain point.[7] Formally we can define legitimacy in the following way:

In every finite time interval $\langle t_{in}, t_f\rangle$, and for every initial state $q \in Q$ and input segment $\omega:\langle t_{in}, t_f\rangle \to X$ there are only a finite number of state transitions.

2.4. *Memoryless Models—Functions*

A memoryless model or function (Func) is a structure[6,12]

$$\text{Func} = (X, Y, \lambda)$$

where

X	is the set of inputs
Y	is the set of outputs
$\lambda : X \to Y$	is the output function

Memoryless Models can be defined in connection with DTS and DESS. When the memoryless model receives an input x_t at time t it simple responds instantaneously with the output value $y_t := \lambda(x_t)$ at the same time t.

2.5. *Coupled Systems or Networks*

When coupling systems of our above defined system types together, we derive coupled systems or networks. The coupling of the components is exclusively done by connecting input and output ports. The network also has its own input ports and output ports. So we have to distinguish between three types of couplings:

— external input couplings—coupling of input ports of the network to input ports of the components;

— external output coupling—coupling of output ports of some components to output ports of the network;

— internal couplings—coupling of output ports of components to input ports of components.

To specify coupled systems, we use the following formalism:[7,8,12,16]

$$\text{Netw} = (X, Y, D, \{M_d\}, \text{EIC}, \text{EOC}, \text{IC})$$

where

X	is the set of inputs of the network
Y	is the set of outputs of the network
D	is the set of the component names
$\{M_d \mid d \in D\}$	is the set of the component systems
EIC	is the external input coupling
EOC	is the external output coupling
IC	is the internal coupling

As we distinguish three system types for atomic system specification, we also derive three coupled system types—DTS networks, DESS networks, and DEVS networks—when coupling together components of the same type.

2.5.1. DTS network Beside the specification of the components and the couplings, the definition of a DTS network requires the specification of the time base. For a well defined DTS networks, the following constraints must hold:

— the components are DTS of type Mealy or Moore, Functions or DTS networks;

— every input port of a component must be influenced by exactly one port;

— every external network output port must be influenced by exactly one port;

— the time base of the network and all its components have to be equal;

— in a feedback loop there has to be at least one component the output of which can be computed without knowledge of its input.

The last constraint is especially important. It is an extension of the well known rule[6] that in every feedback loop there must be at least one Moore type component. This extension is necessary in hierarchical specified networks.

2.5.2. DESS networks For a well defined DESS network the following constraints must hold:

— the components have to be of type DESS, Function or DESS network;
— every input port of a component must be influenced by exactly one port;
— every external network output port must be influenced by exactly one port;
— in a feedback loop there has to be at least one component the output of which can be computed without knowledge of its input.

2.5.3. DEVS networks In DEVS the time base is implicitly given by the real numbers. DEVS networks require the addition of a tie breaking rule to the network definition. This tie breaking rule *Select* selects one component to undertake its internal transition function when more than one of the components are imminent to do so. So Select is a function[7]

$$\text{Select} : \text{subsets of } D \rightarrow D.$$

For well defined DEVS networks the following constraints must hold:

— The components have to be of type DEVS or DEVS network.
— No direct feedback loops are allowed, i.e., no output port of a component may be connected to an input port of itself.

3. NEW SYSTEM FORMALISMS FOR COMBINED DISCRETE-CONTINUOUS MODELLING

To support modelling of a large variety of combined systems, we introduce the following new system types:

— modular, hierarchical multi-formalism systems which are coupled systems whose components are either of the DEVS, DTSS or DESS type;
— atomic DEVS systems with some continuous inputs;
— atomic DESS & DEVS systems which are DESS systems with state and time events causing discontinuities in the state and output trajectories;
— atomic DEVS & DESS systems are DEVS systems also employing some continuous state variables.

In the following we introduce these formalisms and give prototypic modelling examples. Doing so, we will present the state transition and output functions of our atomic model examples in a pseudo-code description similar to a computer program implementation. In particular, we use several pseudo-code elements which should be self-descriptive and we regard the argument s of the state transition functions as an input and output parameter. Hence setting of a state variable in the transition functions means definition of a new state value.

3.1. Coupled Multi-Formalism Systems

This system specification formalism facilitates coupling of components which can be either of the DEVS, DTSS or DESS type.[17] It is possible to couple the components of different types simply by connecting their output and input ports in the same way as we build the one-type coupled systems. Special simulation concepts for this new coupled systems have been developed based on the DEVS abstract simulator[7] and will be outlined in section 4. The DEVS network formalism has proved to be just the right formalism also to specify multi-formalism models. As we will see in section 4, the event handling mechanism of the DEVS network simulation algorithm is used to schedule the internal transitions of DEVS components as well as the state transition and the integration steps of DTSS and DESS components respectively. The Select function of DEVS networks resolving conflicts when more than one component is scheduled to undertake its internal transition is used in multi-formalism networks for the equivalent purpose.

In the following we illustrate how we have to interpret the coupling of systems of different types:

Coupling of DTSS to DEVS Every state transition of a DTSS results in an output and hence in an input event in the DEVS model at regular time intervals.

Coupling of DEVS to DTSS The output of a DEVS which is the input to a DTSS model is interpreted to be piecewise constant. Whenever the DTSS is scheduled to undertake its transition, it takes the value originated from the last output event.

Coupling of DEVS to DESS The output segments of the DEVS are interpreted to be piecewise constant segments which are a subclass of piecewise continuous segments and hence are valid input segments for DESS models.

Coupling of DESS to DEVS The output of a DESS is a continuous segment, i.e., for every point in time a value is defined. Such a segment cannot be interpreted as an DEVS segment because it would mean an infinite number of events and thus the output of a DESS cannot be used as an input segment of ordinary DEVS. The new system type *continuous-input-DEVS* defined below allows to specify DEVS models with continuous input segments.

Coupling of DTSS to DESS In such a coupling structure the output of a DTSS is interpreted in the same way as we interpreted the output of DEVS in DEVS to DESS couplings, namely as piecewise constant. The output at one transition of the DTSS specifies a constant input value for the DESS until the next transition.

Coupling of DESS to DTSS The input of a DTSS coming from a DESS at one transition of time t is defined by the current value of the continuous output segment at time t.

Modelling Example: Combined Analog and Event-Based Motor Speed Control System

A simple model of a motor the speed of which is first controlled by a conventional analog controller which is then controlled and supervised by an intelligent event-based control system[9] will serve as an example of a combined discrete-continuous multi-formalism coupled model. This model and simulation results will be presented in detail in section 5. Figure 6 in section 5 shows the modular, hierarchical structure of the model. The analog part which consists of the model of the motor and the model of the analog PI-controller is modelled employing a modular, hierarchical DESS network. The other part, the event-based control part, is modelled as a modular, hierarchical DEVS network. The overall network "eventBMotorContr2" is a multi-formalism network.

3.2. Continuous-Input-DEVS Systems

Input segments of DEVS systems are constrained to DEVS segments. This guarantees that only a finite number of input events will occur and hence only a finite number of external transitions have to be executed. In a multi-formalism coupled system, when we couple an output of a DESS system to a DEVS system, the input to this DEVS is a continuous segment which represents an infinite number of external events and hence leads to an invalid system description. To overcome this problem, we introduce a new type of atomic DEVS which allows continuous input segments.

An continuous-input-DEVS is a DEVS system with a continuous input segment and hence an infinite number of external events subject to the following constraint:

For every continuous input segment $\omega{:}\langle tin, tf\rangle \rightarrow X$ only a finite number of values $\omega(t)$ actually result in a change of state through the external transition function. We call these events *actual events* and the event times the *actual external event times*.

We call this property *input legitimacy* in the sense *legitimacy* is defined for internal transitions of DEVS.[6] The property guarantees that from the infinite number of events, we only have to care for the finite number of actual events.

Modelling Example: Model of an Intelligent Thresh-Hold Sensor

The following continuous-input-DEVS is a model of a smart thresh-hold sensor which is able to detect if the incoming input segment is stable around 0. Such a sensor can be used to supervise an analog control process. The continuous input to the sensor has to be the deviation of the actual control value from a nominal value. If the absolute value of the deviation is lower than a specified value epsilon and stays lower this value epsilon for a specified time interval, then the signal is regarded to be stable.

StableSensor = $\langle X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, ta\rangle$

X = {deviation | deviation $\in \Re$}
Y = {sensorOut | sensorOut $\in$ {off, on}}
S = {(sigma, sensor) | sigma $\in \Re^{+}_{0\infty}$, sensor $\in$ {off, on}}
δ_{ext}((sigma, sensor), e, deviation)

```
    when-event sensor = off & |deviation| < epsilon then
  sensor := on            —wait timeInterval time units
  sigma := timeInterval   —to put out the sensor value is ''on''
```

```
    when-event sensor = on & |deviation| >= epsilon then
        sensor := off
        sigma := 0   —put out the sensor value is ''off'' immediately
$\lambda$((sigma, sensor)) :=
    sensor   —output of the model is the state variable ''sensor''
$\delta_{int}$((sigma, sensor))
    sigma := $\infty$
ta((sigma, sensor)) :=
    sigma —the time advance is specified by the state variable ''sigma''
```

Figure 2 shows an example simulation run with a test deviation signal (a damped sinusoidal signal) as input. Whenever the signal gets into the epsilon band which is from −0.2 to 0.2 in our example, the state variable sensor is set to "on" and sigma is set to 5. But as soon as the signal leaves the epsilon band, the sensor value is set to "off" again. At time 13 the deviation enters the epsilon band and now stays within the band for more than 5 time units. Only now the sensor reacts with an output at time 18.

3.3. Atomic DESS & DEVS Systems

Atomic DESS & DEVS systems are a combination of DESS and DEVS systems. They employ a derivative function to specify the rate of change of the continuous state variables. The external transition function is used to model state events, i.e., events which depend on conditions stated on the continuous state variables as well as on the continuous input segments.[18,19] The internal transition function together with the time advance function is used to model time events, i.e., events whose time of occurrence is specified in advance.[18,19] While the time advance function specifies the time to the next event, the internal transition function specifies how to treat this time event. Input segments as well as output are continuous or piecewise continuous and hence we regard the system to be continuous.

Formally an atomic DESS & DEVS system is a structure:

$$\text{Atomic DESS \& DEVS} = \langle X, Y, Q, f, \delta_{ext}, \delta_{int}, ta, \lambda \rangle$$

where

X, Y, Q, f and λ have the interpretation as in DESS systems and
$\delta_{ext}, \delta_{int}$ and ta have the interpretation as in DEVS systems.

With an atomic DESS & DEVS system we associate the following dynamic behavior: When there are no state or time events, then the system behaves in the same way as a DESS system does. Input segments have to be piecewise continuous segments and state and output trajectories are continuous segments. The dynamic behavior of the system is defined by the derivative function f specifying the rate of change of the state trajectory at every point in time.

Additionally the DESS & DEVS system has to look for events and handle the events. Therefore it continuously checks if conditions in the external transition func-

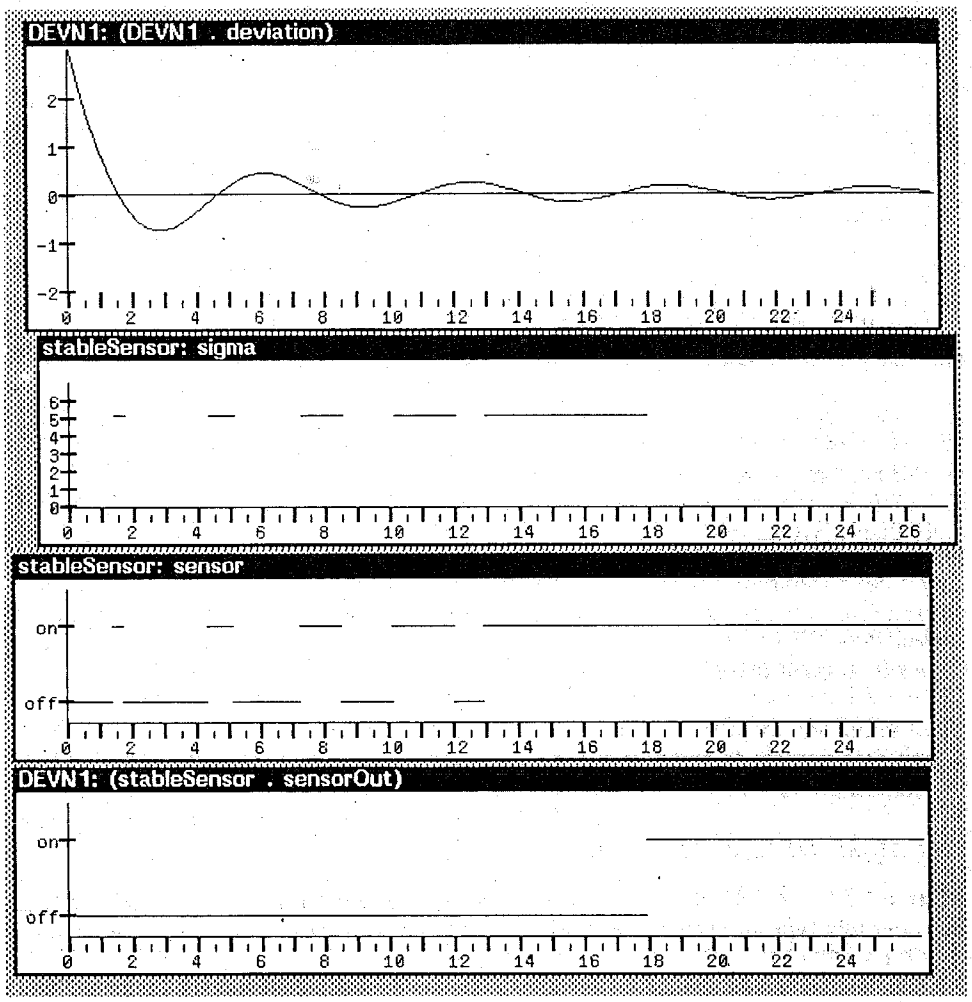


Figure 2 Input segment and state and output trajectories of stable sensor model

tions imposed on the continuous state and input variables and causing state events become true and it checks if the elapsed time e giving the time since the last event becomes equal the value of the time advance $ta(s)$. In the first case, a state event is detected, the external transition function is executed and the elapsed time e is set to 0. In the second case, we have a time event, the internal transition function is executed and again the elapsed time e becomes 0. The state and output trajectories of such a system now are piecewise continuous segments.

As a well defined DEVS and a well defined continuous-input-DEVS have to be legitimate and input-legitimate, we also impose these properties on well defined DESS & DEVS systems. Input-legitimate here means that only a finite number of state events may occur in a finite time interval.

Modelling Example: Bouncing Ball

A table tennis ball dropped from a specified height is a typical example of a situation best modelled by this type of system. The free fall is modeled by the DESS part while the impact of the ball on the ground is modelled by a state event in the external transition function.

Bouncing Ball = $\langle X, Y, S, f, \delta_{\text{ext}}, \delta_{\text{int}}, ta, \lambda \rangle$

$X = 0$
$Y = \{\text{pos} \mid \text{pos} \in \Re\}$
$S = \{(\text{pos}, \text{vel}) \mid \text{pos}, \text{vel} \in \Re\}$

```
f((pos, vel))
   d vel/dt := -g - resistance * vel     —gravitation - resistance
   d pos/dt := vel

λ((pos, vel)) :=
   pos   —output of the model is the position of the ball

δext((pos, vel), e)
   when-event pos <= 0 then
      vel := -vel      —invert the velocity

ta((pos, vel)) := ∞        —time advance = ∞, so no time events
```

3.4. Atomic DEVS & DESS Systems

Atomic DEVS & DESS systems are very similar to atomic DESS & DEVS systems but now this type of systems primarily is a DEVS system which also employs some

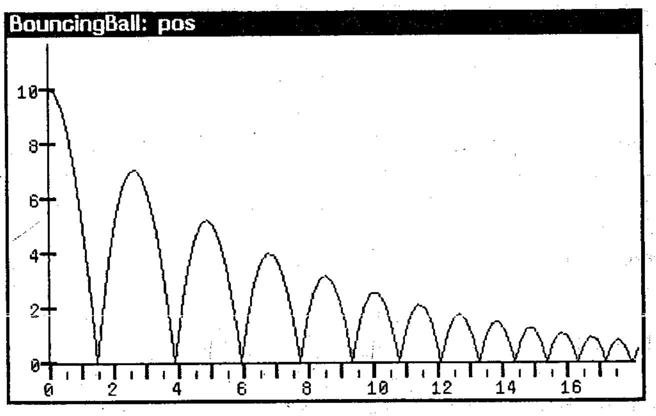


Figure 3 Trajectory of position variable of bouncing ball

continuous state variables. That means, the input and output behavior is like that of a DEVS system or like a continuous-input-DEVS system but some of the interior states behave like that of DESS systems. The external transition function of such a system is used for several purposes. First it can be used in the sense of normal DEVS systems, i.e., it defines the treatment of event inputs. Then it is used in the sense of continuous-input-DEVS, events caused by continuous inputs are specified here. And it is used to model state events caused by continuous state variables.

Formally an atomic DEVS & DESS system is a structure:

$$\text{Atomic DEVS \& DESS} = \langle X, Y, S, \delta_{\text{ext}}, \lambda, \delta_{\text{int}}, ta, f \rangle$$

where

X, Y, S, δ_{ext}, δ_{int}, ta and λ have the interpretation as in DEVS systems and f has the interpretation as in DESS systems.

With an atomic DEVS & DESS system we associate the following dynamic behavior: When there are no external events, state events or actual events caused by continuous inputs, then the system transits from one internal transition to the next. As in DEVS, before every internal transition there is an output. An event input is handled like in DEVS, it activates the external transition function. Between the event times the continuous state variables' trajectories are computed employing the derivative function f. Hence the conditions in the external transition function imposed on these continuous states variables have to be continuously checked if they become true. If so, a state event is detected and the external transition function is executed to handle the state event.

Well-defined DEVS & DESS systems also have to be legitimate and input-legitimate. Input-legitimate here means that only a finite number of state events and in the case that also a continuous input has to be handled that only a finite number of actual events may occur during a finite time interval.

Modelling Example: Model of a Continuous Filling Process

In queueing network simulation models it is sometimes required to model a process more detailed to get better information of the performance indices of the system under study. This may necessitate to model a component at the continuous level and use it in the discrete level. An example of such a component is the model of a continuous filling process which is used in a network modelling a brewery.[20]

The inputs to this model are the arrivals of empty barrels at the filling station. If the filling station is empty, then the filling process starts otherwise the barrel is queued before the station. Depending on the current content of the barrel, the filling rate is increased from an initial filling rate of 1 to a full filling rate of 6. Then the filling rate stays constant until the barrel is full. The current barrel is putted out immediately and a signal is sent out indicating that the next barrel can be served.

The external transition function is used to handle the arrivals of the barrels as well as all state events. There are two state events—one to specify the event when the filling process has to switch from the phase where the filling rate is accelerated to a full speed filling rate of 6 and the second to model the barrel-full event. In the derivative function f, the phase variable of the model is used to distinguish between the different filling rates. Depending on the value of the phase variable, different derivatives are defined for the continuous state variable content. Hence, this model also shows how the phase variable and state events can be employed to implement multi-models as defined by Ören.[21,22]

```
FillingProcess = $\langle X, Y, S, \delta_{ext}, \lambda, \delta_{int}, ta, f\rangle$

$X$ = {in | in ∈ {20, 40, 60}}
$Y$ = {(out, next) | out ∈ {20, 40, 60}, next ∈ {true, NIL}}
$S$ = {(sigma, phase, barrel, content) | sigma ∈ $\Re^{+}_{0\infty}$,
      phase ∈ {waitForBarrel, acceleratedFilling, fullSpeedFilling, done},
      barrel ∈ {20, 40, 60}, content ∈ $\Re^{+}_{0}$}}

$\delta_{ext}$((sigma, phase, barrel, content), $e$, in)
   when-input-event-on-port
      in:  barrel := in
         phase := acceleratedFilling
         sigma := ∞

when-event [(0.2 * content) + 1] >= 6 then
   phase := fullSpeedFilling   —full speed filling now
   sigma := ∞

when-event content >= barrel then
   phase := done
   sigma := 0   —activate the model to put out the full barrel

$\lambda$((sigma, phase, barrel, content)) :=
   (barrel, true)   —send out barrel at port out and true at port next

$\delta_{int}$((sigma, phase, barrel, content))
   barrel := 0
   content := 0
   phase := waitForBarrel
   sigma := ∞   —passivate in idle

$ta$((sigma, phase, barrel, content)) := sigma

$f$((sigma, phase, barrel, content))
   model-in-phase
      waitForBarrel, done: $d$ content/$dt$ := 0
      acceleratedFilling: $d$ content/$dt$ := (0.2 * content) + 1
      fullSpeedFilling: $d$ content/$dt$ := 6
```

Figure 4 shows the piecewise constant trajectory of the state variable *phase* and the piecewise continuous trajectory of the state variable *content* of an example simulation run. Note the dependences of these two variables. A filling process always starts with the phase *waitForBarrel* and a filling rate of 0. In the *acceleratedFilling* phase the content increases with an ever increasing rate (see the curvature at the beginning of the filling processes). After about 9 time units the system transits into

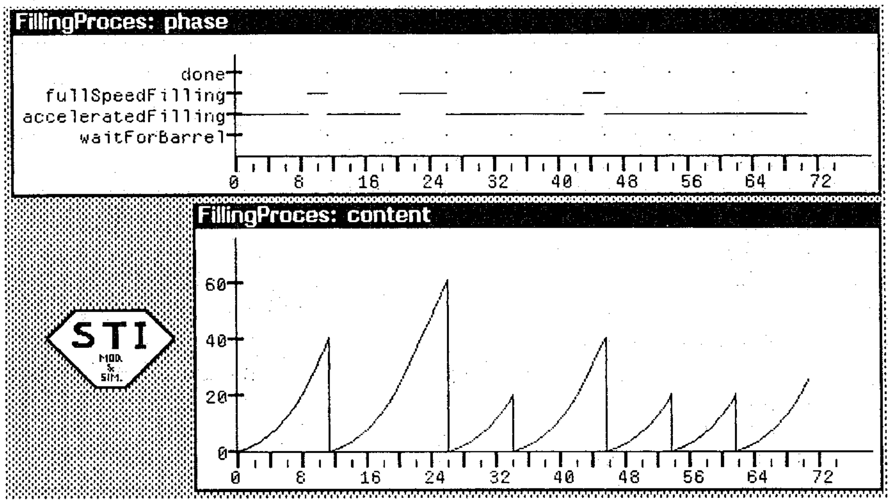


Figure 4 Trajectories of phase and content variables of filling process

the *fullSpeedFilling* phase and we observe a constant increase of the content variable. Finally, in the *done* phase the content makes a discontinuous jump to 0.

4. SIMULATION CONCEPTS FOR MODULAR, HIERARCHICAL MODELS

Zeigler[7] introduced simulation concepts for modular, hierarchical discrete event systems. The abstract simulator for DEVS has a hierarchical structure reflecting the structure of the hierarchical DEVS. It handles all the simulation needs. Thus a DEVS model can be directly transformed into an executable simulation program using the abstract simulator. Based on these ideas we developed similar simulation concepts for DTSS and DESS.[17] We pursued an approach very similar to the DEVS abstract simulator. The abstract simulator for DTSS and DESS also reflect the hierarchical structure of the model. The same methodology and terminology is used. The reason why we do this is mainly to be compatible to the DEVS simulator and to make preparations for the usage of them in the simulation of multi-formalism coupled models.

Each of the different types of abstract simulators consist of two types of object—the simulators and the coordinators. With every atomic model we associate a simulator, with every coupled model we associate a coordinator of the appropriate type. Simulation proceeds by messages passed among the simulators and coordinators.

4.1. DEVS Abstract Simulator

In the abstract simulator for DEVS,[7,16] four different types of messages are distinguished, namely *-, *x*-, *y*-, and done-messages. A *-message indicates that an internal event shall be executed. When a devs-coordinator receives a *-message, the message is transmitted to the subordinate representing the imminent component DEVS. If there are more than one imminent components, the tie breaker order *Select* is used to select one of them. When a *-message is received by a devs-simulator, it computes the output and carries out the internal transition function of the associated DEVS.

The output is sent back to the parent coordinator in a y-message. Finally, a done-message to the coordinator indicates the completion of the state transition.

An x-message indicates the arrival of an external event. When a coordinator receives an x-message, it consults the external input coupling of the DEVS network to generate the appropriate x-message for the subordinate influenced by the external event. When an x-message is received by a simulator, it directly executes the external transition of the atomic DEVS. A done-message reports the completion of the external transition.

When a coordinator receives a y-message which carries the output information of its imminent child, it consults the external output coupling of the DEVS network to see if it should be transmitted to its parent coordinator and the internal couplings to obtain the children and their respective input ports to which the message should be sent. In the latter case the y-message is converted into an x-message indicating the arrival of an external event. It is important to see that devs-simulators and devs-coordinators are able to handle the same type of messages. This enables the simulation of hierarchical specified DEVS.

4.2. DTSS Abstract Simulator

In the abstract simulator for DTSS,[17] special dts-simulators and dts-coordinators exchange messages appropriate for discrete time simulation. As there is only one state transition function, one message to schedule the transitions and compute the outputs (dts-x-message) is sufficient. Another message is used to carry back the output value to the parent coordinator (dts-y-message). When a simulator receives a dts-x-message, it just computes the next state and output of the atomic DTSS and sends the output back to the parent coordinator. When a dts-coordinator receives a dts-x-message, it first stores the input values and then schedules the computations of the next states and the output values of each of its components. The determination of a working sequence of the activations of the components is data driven, i.e., when all inputs for one component are available, the component can go and a dts-x-message is sent to it. In feedback loops, the output of components which can be computed without the knowledge of the input (output of "Moore" type components) have to be computed prior to all other computations. To avoid such a two cycle simulation algorithm, the output of all these components are computed just after the next state computation and are stored for the next simulation step. For the first simulation step these outputs have to be generated in a special initialization phase.[17]

As DTSS models work with fixed time steps, the time scheduling of the DTSS abstract simulator is trivial. The time of the next state transition is equal for all components of a DTSS network and is determined by the current time plus the time step.

4.3. DESS Abstract Simulator

The abstract simulator of modular, hierarchical DESS is very similar in structure and operation to the DTSS abstract simulator. It employs dess-coordinators and dess-simulators and dess-x- and dess-y-messages as well as the same data-driven scheduling scheme. The first difference is that the dess-simulator uses a numerical integration method to compute the next states of the atomic components. The second difference is that, although only a fixed time step numerical integration method is implemented so far, the time step of the DESS abstract simulator is not fixed but additional simulation cycles can be inserted when state or time events or discontinuities in input segments have to be handled.

The DESS abstract simulator first undertakes a normal integration steps to compute the next state value. Then it tests if a state or time event is to be to occur between the current time and the next time. If so, the next integration step will not be scheduled in integration interval time units but an additional simulation cycle will be scheduled at the exact time of the event to handle it. The dess-coordinator now schedules its time of the next simulation cycle by the minimum of the next simulation times of its components. In distinction to the devs-coordinator, the actual time of the next simulation step for all the components is the next simulation time computed in the overall DESS network. Also discontinuities in input segments which are common when the DESS is a component in a multi-formalism network are handled at the exact time of the discontinuity. An additional simulation cycle is inserted whenever an input segment makes a jump from one value to the next.

To be in the position to manage the above simulation scheme a further extension has been made in the DESS abstract simulator. Instead of working with discrete state, input and output values, the trajectories between two consecutive simulation steps are approximated by straight lines. Straight lines now are transmitted as output and input values between components and stored as state values. These lines are used to compute the exact times of state events much better (see below) and they provide much better approximations of the state, input and output values at the time of state, time and external events.

4.4. Abstract Simulator for Multi-Formalism Coupled Models

When components of different system types are coupled, a discrete event network is used to form the multi-formalism network and hence the coordinator for DEVS simulation takes care of the mixed-mode simulation.[17] A *-message triggers the scheduled executions of the transitions of the DTSS and DESS components at their state transition times. X-messages deliver the output of influencers. When a DTSS component receives an input, it just saves this input for the next simulation cycle. When a DESS component in a multi-formalism network receives an input from a DTSS or DEVS component, the input segment makes a jump from the old value to the new one and sticks to this new value. A simulation cycle is inserted at the exact time t of the input. As input, state and output trajectories are approximated by lines, quite accurate state, input and output values can be computed for event time t.

A coupling of a DESS component to a DEVS component results in a continuous input segment—approximated by a straight line—to the DEVS model. The continuous segment finally has to terminate at an atomic continuous-input-DEVS which is able to handle it. When a simulator of a continuous-input-DEVS receives a straight line input at time t, it interprets this line as the input at the current time t and the development of the input value until the next line is received. It then searches for all the actual event times caused by this line segment and executes the external transition function at these times. The property of input-legitimacy—which every continuous-input-DEVS must hold—guarantees that only a finite number of actual events will occur. As the resulting state values are not valid yet, they are saved in a special list. As soon as an internal transition is scheduled or an new external input is received at time t', all the precomputed states resulted from the actual external events until event time $t < t'$ become valid and have to be actualized.

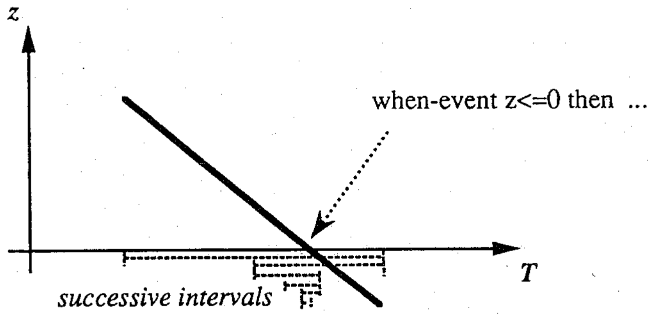


Figure 5 Binary search for event times

4.5. *Computation of Event Times of DESS & DEVS, DEVS & DESS and Continuous-Input-DEVS*

To compute the actual times of the state events of DESS & DEVS and DEVS & DESS as well as the actual external event times of continuous-input-DEVS we employ the following method[18] (Figure 5): First we analyze the external transition function to see if any condition specifying an event changes its value during the beginning and the end of the current lines approximating the input and state segments. If so, an iterative process is started to search for a sufficient accurate time of the event. As a starting interval the whole interval from the beginning to the end of the lines is taken. Then we compute the state values and input values in the middle of the interval. Using the line approximations it is possible to compute quite accurate values for them. Then we check if the event is in the first half or the second half of the whole interval. If it is in the first half, we take the first half as new interval, otherwise we take the second. With the new interval we continue the approximation process. This binary search process stops when the time interval is sufficient small.

5. SIMULATION OF AN EVENT-BASED MOTOR SPEED CONTROL SYSTEM

An example of a motor speed control system[23] shall demonstrate the expressive power of our system theoretic modelling formalisms and simulation concepts for multiformalism coupled systems. The model has been implemented and evaluated using the STIMS modelling and simulation environment—the implementation of our modelling and simulation concepts in Interlisp-D/LOOPS.

A motor which drives a machine has to be brought to different speeds and different work loads are exerted to fulfill a predefined sequence of tasks. An event-based controller is used to define this sequence of tasks in pursuit of a given goal. It issues the required nominal speeds as well as the work loads and supervises the process runs. To drive the motor to the nominal speeds, a conventional analog controller is used.

Figure 6 shows the structure of this control system. The analog part, which is modelled by a DESS coupled system, consists of the component modelling the motor

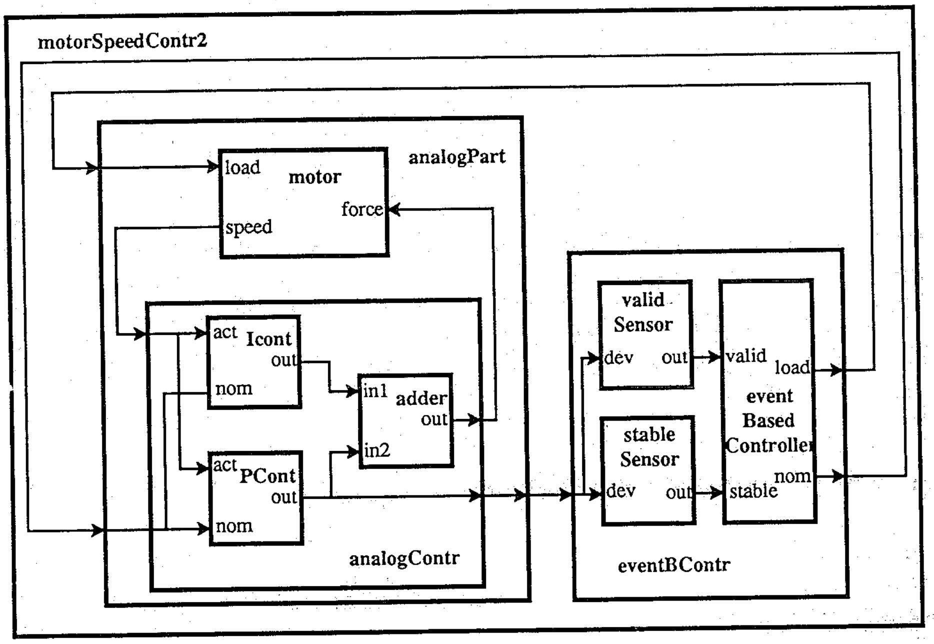


Figure 6 Model of an analog and event-based motor speed control system

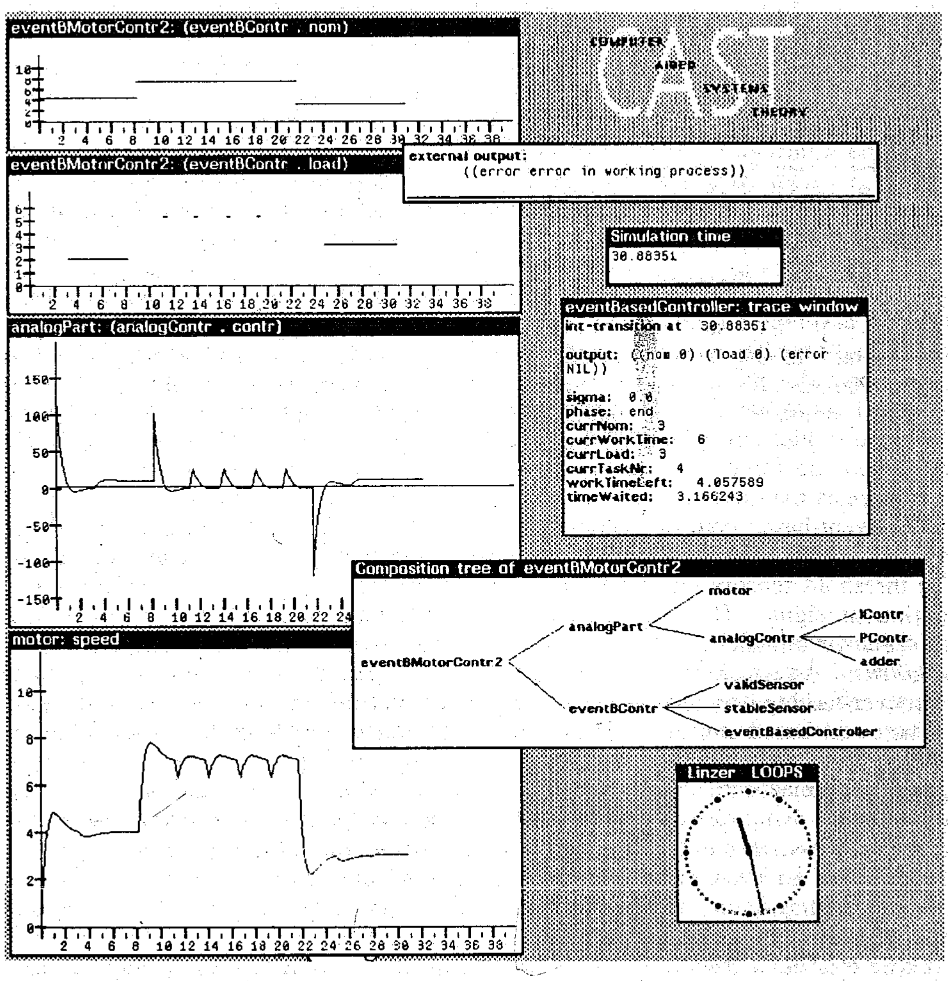
as well as the model of the analog controller. The motor is modelled at the behavioral level by a simple atomic DESS and should be considered as a motor driven by a force input together with a tachometer which provides the current speed. The input "load" provides the current work load which functions like a resistance. The conventional controller is a combination of an integral and proportional controller the outputs of which are added. The adder also amplifies the control signal to provide an appropriate force to drive the motor. A second output from the analog control part supplies the deviation of the current speed from the nominal speed.

The event-based control system is modelled by a DEVS network and consists of three parts. There are two threshold sensors and the event-based controller itself. The threshold sensors receive the deviation from the analog controller in form of a continuous signal. They are modelled by continuous-input-DEVS. For the model *stableSensor* a model as specified in the example of section 3.2 is used. The model *validSensor* is very similar with the only difference that the sensor immediately reacts whenever the input is recognized to be lower than epsilon.

The event-based controller now knows of a sequence of tasks. Each task consists of a nominal speed, a work load and a working time. First it exerts the nominal speed. This makes the analog controller to react and drive the motor to the new speed. In the event-based controller, a time window[9] is used to supervise the analog control process. In the case it takes to long to drive the motor to the new speed, a failure in the analog controller is detected and the process is stopped.

When the threshold sensor stableSensor reacts, the speed is at the right value and is stable, the real working process can begin, and the work load is exerted. The working load now causes a loss of speed which, if the work load is not too high, can be compensated by the analog controller. But if the work load is too high, the controller is not able to compensate the loss fast enough, the threshold sensor validSensor reacts and the load is withdrawn to allow a restabilization of the speed. This procedure is continued until the absolute stabilization time exceeds the working time. After that, the event-based controller prints an error message and continues with the next task.

Figure 7 shows a screen of the simulation of this control system using the STIMS environment. The four trajectories show nominal speeds and work loads issued by the event-based controller, the control output of the analog controller and the speed of the motor. The first task with nominal speed 4 and work load 2 is processed correctly. After some time the speed is stable and the work load is exerted. The loss in speed through the load can be compensated fast enough and the work is finished without any interrupt. In the second task which is characterized by nominal speed 7 and work load 5, the work load is too high resulting in a big and fast loss of speed, a reaction of the valid sensor, and hence in a withdrawal of the load. After some


eventBMotorContr2: (eventBContr . nom)
eventBMotorContr2: (eventBContr . load)
analogPart: (analogContr . contr)
motor: speed
COMPUTER AIDED SYSTEMS THEORY
CAST
Composition tree of eventBMotorContr2
eventBMotorContr2
analogPart
motor
analogContr
IContr
PContr
adder
eventBContr
validSensor
stableSensor
eventBasedController
Linzer LOOPS

```
external output:
      ((error error in working process))
```

```
Simulation time
30.88351
```

```
eventBasedController: trace window
int-transition at   30.88351

output: ((nom 0) (load 0) (error
NIL))

sigma:  0.0
phase:  end
currNom:  3
currWorkTime:   6
currLoad:  3
currTaskNr:   4
workTimeLeft:   4.057589
timeWaited:   3.166243
```

Figure 7 Simulation of the event-based motor speed control system

other trials the event-based controller prints an error message and continues with the next task.

6. SUMMARY AND OUTLOOK

System theoretic concepts have been employed to set up a theoretical basis for combined discrete-continuous system modelling and simulation. Therefore, several new system theoretical formalism have been introduced; by these new formalisms it is possible to model differential equation specified systems with discontinuities, discrete event systems which also employ some continuous state variables and multi-formalism coupled models, i.e., multi-component systems whose components are of different system type. Simulation algorithms are based on the abstract simulator concept for discrete event simulation.

Solutions of differential equation specified systems have been restricted to numerical, stepwise integration and no analytical methods have been employed so far. However, it is intended to extend the modelling and simulation schemes in that direction. The method follows the approach of Zeigler and Cellier to DEVS modelling of continuous systems[9,24] and looks the following: we assume that we like to model a system by a DEVS which also incorporates some continuous state variables for which we know an analytical solution. Furthermore we assume that we also know all the values of the continuous state variables which may lead to state events. Then, we are in the position to do without a stepwise integration and "jump from one event to the next" just like in ordinary discrete event simulation. At every event—external, state or time event—we use the analytical solution to compute the values of the continuous state variables just before the event. Then we execute the external or internal transition function and thereafter we can compute the time of the next state event using the state equations and a numerical root-finding method.[25]

ACKNOWLEDGEMENTS

Part of this work has been conducted while staying at the University of Arizona as a visiting scholar. I would like to thank Professor Bernard P. Zeigler and Professor Jerzy W. Rozenblit for their helpful contributions, advice and guidance. This work has been supported by SIEMENS AG Munich, Department ZFE IS.

REFERENCES

1. NASA, *The Space Station Program*. NASA Publication, 1985.
2. B. P. Zeigler, "High Autonomy Systems: Concepts and Models." In: *Proc. of the conference AI, Simulation and Planning in High Autonomy Systems*, IEEE Computer Society Press, 1990, pp. 2–7.
3. A. Meystel, "Intelligent Control: A Sketch of the Theory." *J. of Intelligent and Robotic Systems*, **2**, Nos. 2 & 3, 1989, pp. 97–107.
4. P. J. Antsaklis, K. M. Passino and S. J. Wang, "Towards Intelligent Autonomous Control Systems." *J. of Intelligent and Robotic Systems*, **1**, No. 4, 1989, pp. 315–342.
5. G. Saridis, "Intelligent Robotic Controls." *IEEE Trans. Auto. Control*, AC-28, No. 5, 1983.
6. B. P. Zeigler, *Theory of Modelling and Simulation*. John Wiley, New York, 1976.
7. B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, London, 1984.
8. B. P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models*. Academic Press, London, 1990.

9. B. P. Zeigler, "DEVS Representation of Dynamical Systems: Event-Based Intelligent Control." *Proceedings of the IEEE,* **77,** No. 1, January 1989, pp. 72–80.
10. D. G. Bobrow and M. Stefik, *The LOOPS Manual.* Xerox PARC, Palo Alto, CA, December 1983.
11. B. P. Zeigler, "System-Theoretic Representation of Simulation Models." *IIE Transactions,* March, 1984, pp. 19–34.
12. T. I. Ören, "GEST—A Modelling and Simulation Language Based on System Theoretic Concepts." In: *Simulation and Model-Based Methodologies: An Integrative View,* edited by T. I. Ören, B. P. Zeigler and M. S. Elzas, NATO ASI Series, Series F: Computer and System Sciences, Vol. 10, Springer Verlag, 1985, pp. 281–335.
13. T. I. Ören, "Simulation Models: Taxonomy." In: *Encyclopedia of Systems and Control,* edited by M. Singh, Pergamon Press, New York, 1987.
14. F. R. Pichler, "Dynamical Systems: A Survey." In: *Encyclopedia of Systems and Control,* edited by M. Singh, Pergamon Press, New York, 1987.
15. F. R. Pichler, *Mathematische Systemtheorie.* Walter de Gruyter, Berlin, FRG, 1975.
16. B. P. Zeigler, "Hierarchical, Modular Discrete Event Simulation in an Object Oriented Environment." *Simulation Journal,* **49,** No. 5, 1987, pp. 219–230.
17. H. Praehofer and B. P. Zeigler, "Modelling and Simulation of Non-Homogeneous Models." In: *Computer Aided Systems Theory—EUROCAST '89,* edited by F. Pichler and R. Moreno-Diaz, Lecture Notes in Computer Science, Springer Verlag, Berlin 1990, pp. 200–211.
18. F. E. Cellier, *Combined Continuous/Discrete System Simulation by Use of Digital Computers. Techniques and Tools.* PhD Thesis, Diss ETH No. 6483, Swiss Federal Institute of Technology, Switzerland, 1979.
19. F. E. Cellier, "Combined Continuous/Discrete Simulation—Application, Techniques, and Tools." *Proceedings of the 1986 Winter Simulation Conference,* edited by J. Wilson, J. Henrikson, S. Roberts, 1986, pp. 24–33.
20. B. Schmidt, *Model Construction with GPSS-FORTRAN Version 3.* Springer Verlag, Berlin, 1987.
21. T. I. Ören, Dynamic Templates and Semantic Rules for Simulation Advisers and Certifiers." In: *Knowledge-Based Simulation: Methodology and Application,* edited by P. A. Fishwick and R. B. Modjeski, Springer-Verlag, Berlin, Heidelberg, New York, 1989.
22. B. P. Zeigler and H. Praehofer, "System Theory Challenges in the Simulation of Variable Structure and Intelligent Systems." In: *Computer Aided Systems Theory—EUROCAST '89,* edited by F. Pichler and R. Moreno-Diaz, Lecture Notes in Computer Science, Springer Verlag, Berlin 1990, pp. 41–51.
23. H. Praehofer, "Simulation of Event-Based Control of Continuous Systems." In: *Proc. of the conference AI, Simulation and Planning in High Autonomy Systems,* IEEE Computer Society Press, 1990, pp. 89–96.
24. Q. Wang and F. E. Cellier, "Time Windows: An Approach to Automated Abstraction of Continuous-Time Models into Discrete Event Models." In: *Proc. of the Conference AI, Simulation and Planning in High Autonomy Systems,* IEEE Computer Society Press, 1990, pp. 204–211.
25. R. L. Burden and J. D. Faires, *"Numerical Analysis."* PWS-Kent Publishing Company, Boston, 1989.

Herbert Praehofer is a faculty member of the Department of Systems Theory and Information Engineering at the Johannes Kepler University of Linz, Austria. He got his M.Sc. degree in Computer Science from the University of Linz in 1986. Currently he is working on his Ph.D. degree. From September 1988 to March 1989 he was a visiting scholar at the University of Arizona, Tucson, working with Professor Zeigler and Professor Rozenblit. His research interests include modelling and simulation, object oriented programming, interactive user interfaces and artificial intelligence techniques in modelling and simulation.