DEVS Theory of Quantized Systems

Bernard P. Zeigler

AI and Simulation Group[1]
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona

A Deliverable in Fulfillment of

Advance Simulation Technology Thrust (ASTT)
DARPA Contract N6133997K-0007

June 1998

[1] **URL:** www-ais.ece.arizona.edu; **Email:** zeigler@ece.arizona.edu

ABSTRACT

This report presents a theory of quantized systems in support of predictive filtering based on a process called "quantization" to reduce state update transmission. A quantized system is a system with input and output quantizers. Quantization, which generates state updates only at quantum level crossings, abstracts a sender model into a DEVS (Discrete Event System Specification) representation. This affords an alternative, efficient approach to embedding continuous models within distributed discrete event simulations. The theory of quantized systems examines the conditions under which a coupling of DEVS-represented systems is a good representation of the original composition. This corresponds to closed loop study of predictive filtering, i.e., where both sender and receiver are exposing abstractions of each other. Previous analyses of Dead Reckoning accuracy/performance tradeoffs have assumed that open loop analysis carries over to the closed loop case. Unfortunately, experience in numerical analysis suggests that the dynamics of feedback interaction may cause errors generated to grow without bound. The theory of quantized systems provides conditions under which homomorphic (error-free) quantization-based predictive filtering is possible. It shows how error can be generated if the conditions are violated and formulates a suitable concept of approximate homomorphism. Applications of quantization to message traffic reduction are discussed. The theory has been confirmed by simulations of test cases. It will be subject to further test in actual distributed simulations using the DEVS/HLA modeling and simulation environment under construction.

EXECUTIVE SUMMARY

Under the influence of the HLA (High Level Architecture) standard, distributed simulation is slated to become the predominant form of model execution. Distributed simulation architectures must meet simulation goals such as speed and fidelity under constraints imposed by limits of communication bandwidth connecting computing nodes and compute power within the nodes. Filtering techniques such as Dead Reckoning, have been shown to significantly reduce the message traffic required to simulate realistic battlefield environments in distributed interactive simulation (DIS). Most of the research in this area has been empirical in nature and there is a need to establish a rigorous theoretical basis for developing and evaluating message reduction schemes. Such a framework should help to assess the trade-off between message filtering, local computation and loss of accuracy, particularly, the critical effect of feedback on error propagation.

Since digital data messaging requires that continuous quantities be coded into discrete packets or cells and sent discontinuously, event-driven simulation is the most natural means to examine such problems. However, time-driven simulators, the traditional way of simulating continuous systems, must also be considered. DEVS (Discrete Event System Specification) is a system-theory formalism capable of representing both kinds of models in discrete event form. The aim of the work reported here is to extend the theoretical framework underlying the DEVS formalism to enable it to support rigorous investigation of predictive filtering methods.

This report discusses a concept, called DEVS Bus, in which non-discrete event formalisms are represented in DEVS in a distributed simulation environment, such as HLA. Particularly, we focus on an approach, called quantization, in which attribute updates are sent only when threshold boundaries are crossed, rather than continuously as they are generated. The greater the separation between thresholds – called the quantum size – the greater the reduction in messages sent, and, also the greater the potential for accuracy loss through out-of-date information transmission. Thus such a representation offers the promise of greatly reduced message transmission with controllable error and computational cost.

We begin with a baseline approach to representing time-driven systems in which time is discretized and standard numerical integration employed to simulate each component. Quantization is then applied to the outputs of components to reduce message traffic between them.

A novel alternative approach is then introduced in which a notion of quantized system is formulated. Quantized system theory facilitates a direct and efficient mapping of time-driven systems into discrete event form. It employs discrete-event techniques to advance time to the next predicted boundary crossing thus saving on both computation and message updating.

The report offers some empirical evidence of the advantages of the quantized system over the baseline approach to the DEVS Bus. We show that significant reductions of message bandwidth demands (number and size of messages) with controllable error and local computation costs are possible.

While the main goal of the report is to present a framework for addressing issues in predictive filtering, we go on to formulate conjectures to explain why the quantized DEVS approach appears to afford an efficient basis for such techniques and where its limitations might lie. We also indicate areas requiring further research along the lines established in this work.

1. INTRODUCTION

Fostered by the promulgation of the HLA (High Level Architecture)[1] distributed simulation, the use of multiple computers for model execution is fast becoming the predominant form of simulation. The major difference between distributed simulation and its non-distributed counterpart is that information and data are encoded in messages that travel from one computer to another over a network. This requires a change in perspective on simulation from one in which only computation matters, to one in which communication matters just as much, and sometimes, more. The brunt of simulation analyses in the past (e.g., in the design of numerical methods) had to do with trade-offs between accuracy and speed of computation. While these retain their importance, our new perspective demands that we look at trade-off between accuracy and communication resources as well. For example, we can ask what is the best architecture to simulate a model distributed among several sites, given limited bandwidth, or given limited bandwidth *and* computer memory (the latter taking both computation and communication constraints into account). Research in this direction has been done in the context of distributed interactive simulation (DIS) where tremendous bandwidth and communication resources are required[2]. While significant advantages of message reduction techniques such as Dead Reckoning predictive filtering have been established, most of these studies have been empirical in nature[3, 4]. In this report, we take a fundamental approach that is aimed establishing a rigorous theoretical basis for developing and evaluating message reduction schemes. Notably, generalizing from classical numerical analysis, our approach enables us to incorporate the critical effect of feedback on error propagation.
Since messaging requires that continuous quantities be coded into discrete packets or cells and sent discontinuously, discrete event simulation is the most natural means to examine such problems. DEVS (Discrete Event System Specification) is a formalism representing models in discrete event form[5-7]. However, the models to be simulated cannot be restricted to discrete event form only since many, especially coming from the physical sciences, are cast in differential equation terms and may come packaged in time-driven simulators, the traditional way of simulating continuous systems.

The aim of the work reported here is to extend the theoretical framework underlying the DEVS formalism to enable it to support the investigation of predictive filtering methods. The structure of the remaining part of report is:
Section 2 presents some of the background needed to follow the mathematical presentation. Only the most immediate supporting material is presented with references made to the literature for further details.
Section 3 discusses a concept call DEVS Bus in which non-discrete event formalisms can be represented in DEVS in a distributed simulation environment, such as HLA. Particularly, we focus on an approach, called quantization, in which attribute updates are sent only when threshold boundaries are crossed, rather than continuously as they are generated. The greater the separation between thresholds – called the quantum size – the greater the reduction in messages sent, and, also the greater the potential for accuracy loss through out-of-date information transmission.
Section 4 discusses our first approach to representation of differential equation systems in which time is discretized and standard numerical integration employed to simulate each component. Quantization is then applied to the outputs of components to reduce message traffic between them.
Section 5 builds on the techniques of Section 4 to study a novel approach in which quantized systems are formulated. and facilitate a direct DEVS representation of differential equation systems.
Section 6 provides some empirical evidence of the advantages of the quantized system approach to the DEVS Bus. We show that significant reductions of message bandwidth demands (number and size of messages) with controllable error and local computation costs are possible.
Section 7 offers conjectures on why the quantized DEVS approach provides an efficient basis for predictive filtering techniques and indicates areas requiring further research along the lines established in this work.

2. BACKGROUND

In this section we review some of the background material needed to present the research results. Beginning with an informal presentation of the DEVS formalism, we go on to discuss the basics of the other major formalisms for simulation. The aim is to present basic concepts and notation but not to go into any depth beyond that needed to follow the later discussion. The reader needing greater detail is directed to the extensive development of the background material available in [5, 6]

2.1. Informal Review of the DEVS Formalism

We begin with an informal exposition of the DEVS formalism, focusing on the concepts rather than the formal underpinning. A DEVS model is a modular system receiving inputs, changing states, and generating outputs over a time base. Input and output streams are time-indexed series of events. Figure 1 illustrates an input event segment containing inputs x_0 x_1 x_2 arriving at times t_0 t_1 t_2 respectively; and a output event segment with outputs y_0 y_1 departing at times t'_0 t'_1 respectively. There are no constraints on the spacing between events; however, only a finite number of events are allowed in a finite time interval.

Figure 1 Input and Output Event Streams

DEVS models have input and output ports through which all interaction with the external world takes place. By coupling together output ports of one system to input ports of another, outputs are transmitted as inputs and acted upon by the receiving system. Thus, there are two types of DEVS models, *atomic* and *coupled.* An atomic model directly specifies the system's response to events on its input ports, state transitions, and generation of events on its output ports. A coupled model is a composition of DEVS models that presents the same external interfaces as do atomic models. For example, in Figure 2, CM is a coupled model with four components. A coupled model specifies three types of coupling:

- *external input* – from the input ports of the coupled model to the input ports of the components (e.g., from start of CM to start of counter)
- *internal* – from the output ports of components to input ports of other components (e.g., from explosion of bomb to strike of target), and
- *external output* – from the output ports of components to output ports of the coupled model (e.g.. from damage of target to damage of CM.

Although arbitrary fan-out and fan-in of coupling is allowed, no self-loops are permitted. DEVS is closed under coupling, which means that a coupled model can itself be a component within a higher level coupled model, leading to hierarchical, modular model construction.

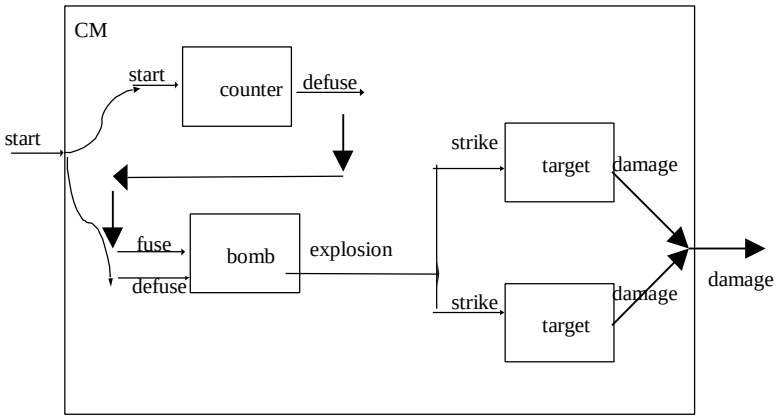

Figure 2. Coupled Model

To illustrate the DEVS modeling process, consider a scenario in which a bomb can threaten some objects in its vicinity called targets. If the bomb is left undisturbed over a long period it will lose its potency. If its fuse is ignited, it will in a very much shorter time, explode and impact the targets. However, if during this period, it is deactivated, the bomb will not go off but return to its dormant state. Many processes have similar *conditionally timed behavior*. For example, a grain can be dormant until picked up by a bee and transported to pollinate other flowers; but the bee may be killed before arriving at its destination thereby aborting the attempt at plant reproduction. A pre-emptive processor can be leave the job currently being done if interrupted by one of higher priority. A reliable network packet protocol waits for a fixed time for all packets in a message to be acknowledged by the receiver; otherwise it retransmits them (often called a timeout). The modular, state-based concept of DEVS enables it to respond such external inputs based on its current state and the time that has elapsed in that state.

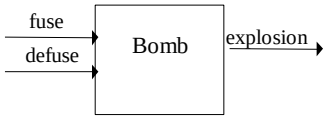

Figure 3 Atomic Model

To model this type of behavior in DEVS we define a coupled model, as in Figure 2, containing components for the bomb, some targets, and a defuser capable of deactivating the bomb. Each component is specified as a modular atomic model with input and output ports. In particular, the bomb has input ports for fusing (activating) and defusing (deactivating) it and an output port for the explosion event (Figure 3). The behavior over a continuous time line is illustrated in Figure 4. External input events are shown as vertical lines on the X time line – e.g., a fuse event occurs at time = 2 in Figure 4b. Phases (which are states labels

such as dormant, dead, fused, etc.) are persistent as shown by constant lines and the transition from one phase to another are shown by jumps in the lines (as from dormant to dead in Figure 4a). A DEVS system autonomously determines the *resting time* (also called *time advance*) in a phase as a function of that phase (actually of the state associated with the phase which may contain other variables). For example, the time to stay in phase dormant is 150 after which time the system transitions to phase dead. The time that has passed in a phase is the elapsed time as depicted on the E access in Figure 4. The change in state occurring when the resting time has elapsed is called an *internal event.* In a typical uniprocessor simulation program, an internal event is scheduled on an event list.

Due to its modularity, DEVS distinguishes between internal and *external* events. While internal events represent state changes that are self induced, *external input events*, on the other hand, are impressed from outside the model and are inputs to which the system must respond. For example, the arrival of a fuse event at time = 2 in Figure 4b causes the phase to change from dormant to live. *As part of the effect of an external input event, the timing of the next internal event may be changed.* In this case, the time advance associated with phase live is 10, which means that at time = 12, the phase is scheduled to change to explode (whereas before the external event, the dormant-to-dead transition was scheduled at 150). *Output external events* are generated by a system at its internal events. For example, the transition from explode to dead generates the explosion output event in Figure 4b.

Figure 4c illustrates the most critical conditional timing feature of the bomb example – this is the arrival of a defuse external event while the system is in phase live and counting down toward the explosion output. The defuse input causes the system to revert to phase dormant thus aborting the scheduled explosion. Note that from the bomb's point of view, the arrival of the defuse input is unpredictable, it could arrive in time (i.e., before the time elapses in phase live) or after this time has elapsed (including never). The DEVS formalism, which modular concept of external events, requires and allows explicit specification of the dynamics of such conditionally timed events.

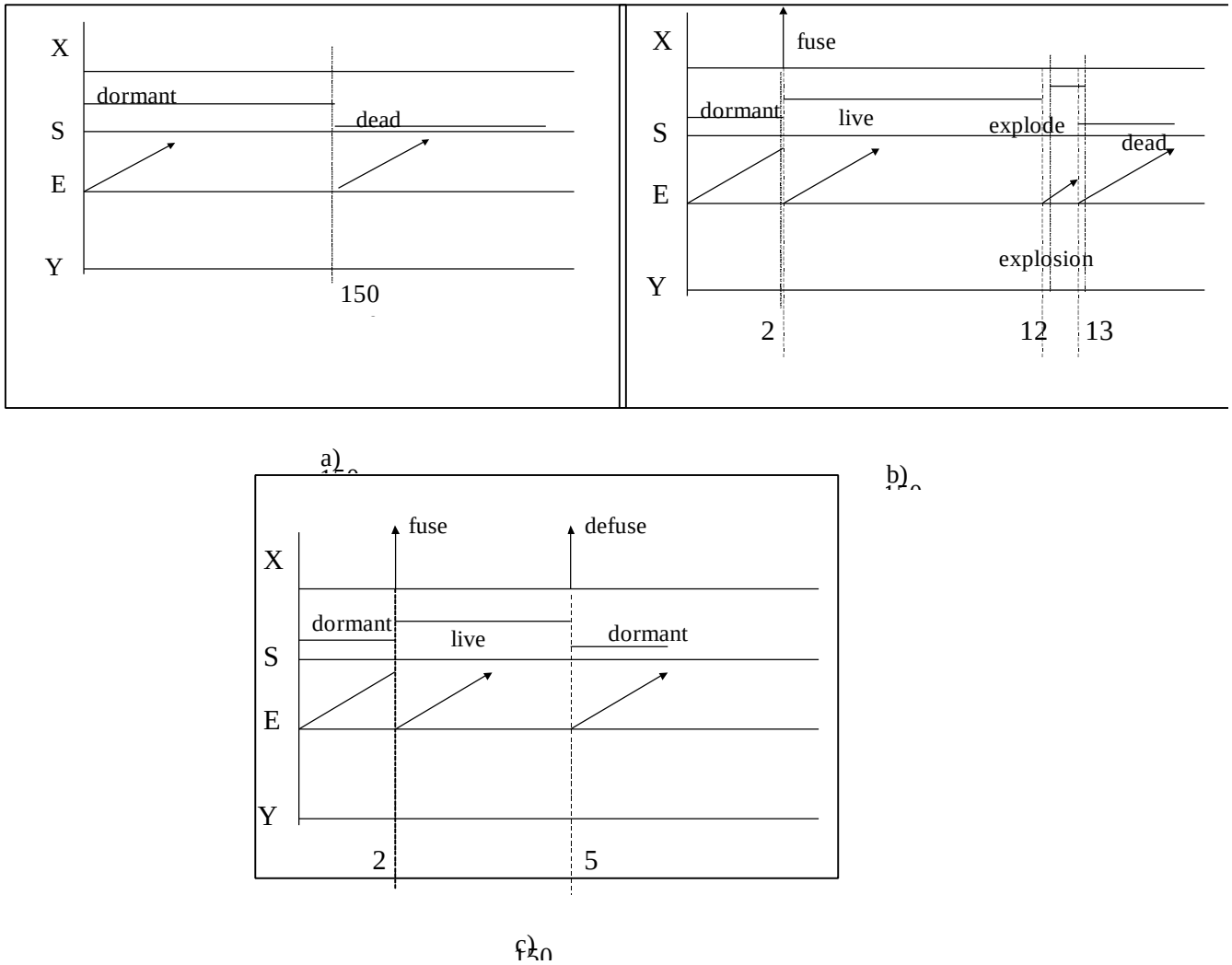

Figure 4 DEVS Temporal Behavior

Figure 5a illustrates a *phase transition diagram* that portrays the dynamics of an atomic model. The two kinds of transitions are shown – *external*, an arrow labeled by an internal port, and *internal* – a dashed arrow labeled with a time advance. For example, the transition from dormant to live is externally induced by the fuse external event. The transition from live to explode is an internal transition as is the transition from explode to dead.

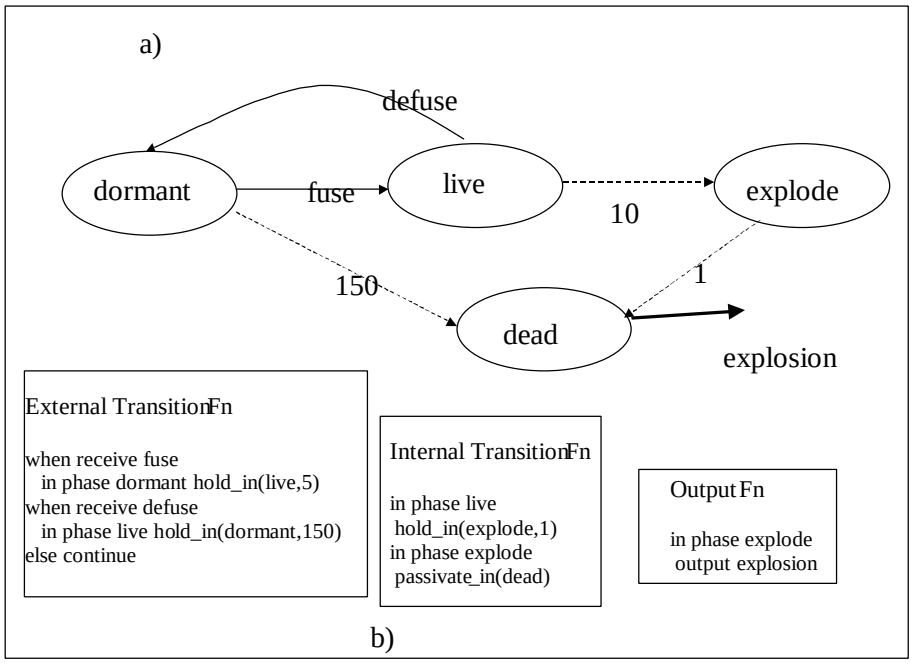

Figure 5 Atomic Model Specification

Such state diagrams are good for giving a graphical abstract representation of the dynamics but they have difficulty conveying their details. The authoritative specification of an atomic model is given in its internal transition, external transition, and output functions as in Figure 5b.

The External Transition Function specifies how the system responds to external events. In general, this response is a change in state conditioned on the nature of the external event (input port of arrival and parameters of the received message), the current state, and the time elapsed since the last event. For example, an arrival on port fuse changes the phase from dormant to live. But the time that the system has been dormant might degrade the power of the explosion. This might be modeled by including a state variable, potency, that would be reset by the external transition function to a decaying exponential function of its original value and the elapsed time in phase dormant.

The Internal Transition Function specifies the change in state that occurs upon an internal event. It is given as a function that maps the state before the event to the state that is to be in effect immediately after it.

The Output Function specifies the output port and value that will be generated just before an internal event. It is given as a function that maps the state before the event to the output space (set of possible outputs). Not all states need to generate outputs. For example, transitioning from live to dead generates the explosion output, while no other transitions generate output events.

Figure 5b does not show another characteristic DEVS feature, *the time advance function*. This assigns to every state the time in which the system will stay in this state before an internal event occurs. Rather then give this function explicitly, we allow a state variable, called *sigma*, to hold its value for the current state, and to have this value set by the external and internal transition functions. The command "hold_in (live, 10)" means set the phase to live and sigma to 10. The command "passivate_in(dead) means to set phase to dead and sigma to infinity. In other words, the system is scheduled to remain in dead forever[2]. It is also possible for a state to have a zero time advance. For example, we might change the time to go from explode to dead (the explosion time) from 1 to 0.

[2] unless an external event changes this), but an event capable doing this is not shown in Figure 5

2.2. Coupled Model Simulation Cycle

DEVS has a well-defined semantics and associated simulation algorithm, which we present informally here[3]. Each component in a coupled model has two time keeping variables, tL (time of last event) and tN (time of next event). Before starting a simulation run, each component is initialized to its designated initial state and its time keeping variables set: tL= 0, tN = the time advance of the initial state.

We present a somewhat simplified version of DEVS simulation cycle for a single level coupled model:

1. Set the current global time, t = the minimum of the components' tN's
2. Send t to each component
3. Each component, c then compares t with its tN, if t= tN, this component is said to be imminent and
    - c generates its output (if any) stamped with time t
    - c executes its internal transition function
    - c sets tL = t and tN = tN + time advance of the new state
4. The collected outputs move, as dictated by the coupling specification, to the input ports of other components
5. Each component examines its input ports and:
    - if it receives an input, it applies its external transition function with this input, using the elapsed time, t - tL
    - sets tL = t and tN = tN + time advance of the new state

(if no input was received it does nothing)

1. if not at the end of the run, return to 1.

Figure 6a illustrates this process. Suppose the bomb is in state dormant, the countermeasure has time advance = 15, and the targets are in passive states (with infinite time advance). The bomb is then the only imminent component since it has the smallest tN = 10. Current time then advances to 10, and the bomb transitions to state explode, which having time advance = 1, tN becomes 11. There are no outputs to be sent.

In the next cycle, time is advanced to 11 and the bomb is still the imminent component. It generates the explosion output which the coupling directs to the targets. Since it passivates in dead, the bomb will no longer be imminent in the next cycle. Note that in this case, the countermeasure is too late to save the targets.

[3] The algorithm, somewhat simplified, is for a recently introduced version of DEVS called Parallel DEVS. Parallel DEVS removes constraints that originated with the sequential operation of early computers and hindered the exploitation of parallelism, a critical element in more modern computing.

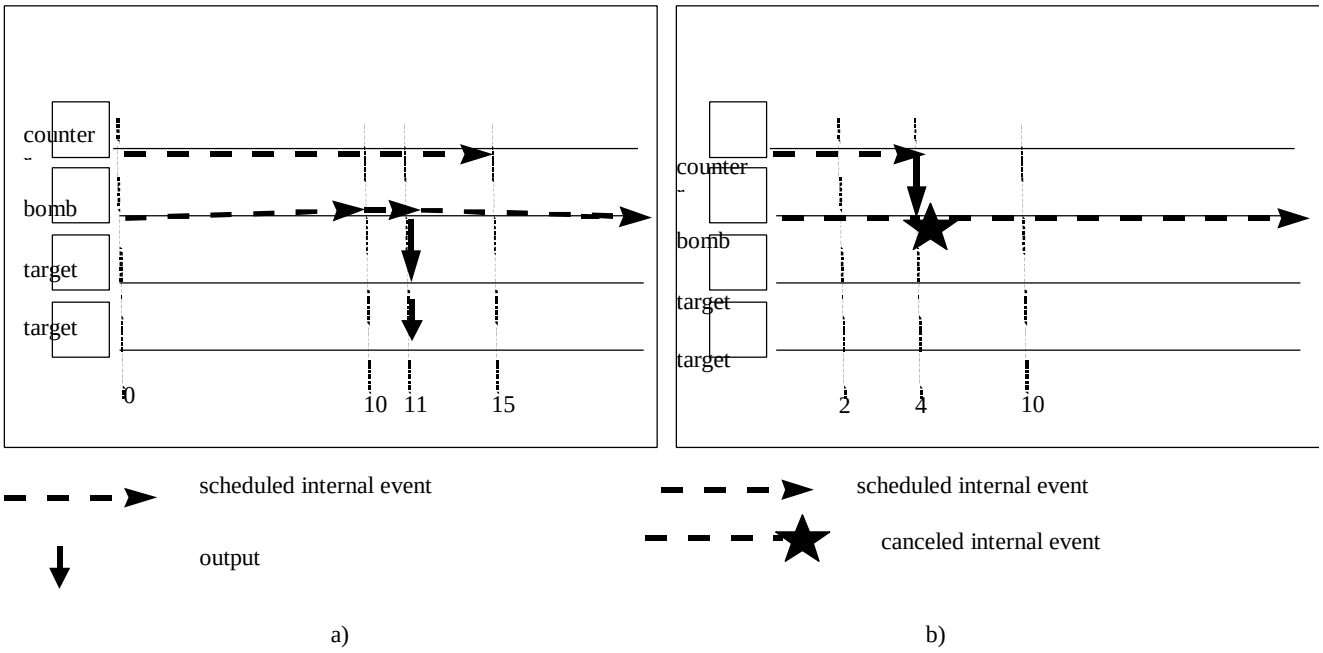

Figure 6 Coupled Model Simulation

Now consider a case in which the countermeasure acts in time as in Figure 6b. Here the countermeasure's initial tN = 4 and being less than the others, it is imminent. Time is advanced to 4 and the countermeasure generates a defuse output which the coupling sends to the bomb's defuse input port. Arrival of this input, causes the bomb's external transition function to execute, which sends it to phase dormant. As a consequence tL becomes 4 and tN becomes 154 (current time + time advance in dormant).

Note that in the last case, the scheduled internal event of the bomb did not materialize since an input interrupted it. This caused the bomb, in effect, to cancel its currently scheduled internal event, reschedule another one for a different time

2.3. Systems Definitions

Before proceeding, we need to provide a brief introduction to the generic systems formalism that underlies all our subsequent discussion. A full presentation of the underlying systems theory formalism is available in [5, 6]

We define a system as a structure:

$S = (T, X, \Omega, Y, Q, \Delta, \Lambda)$

where

T is a time base

X is a set – the input values set

Y is a set – the output values set.

Ω is a set – the set of allowable input segments

Q is a set, the *set of states*

$\Delta: Q \times W \rightarrow Q$ is the *global state transition function*

$\Lambda: Q \rightarrow Y$ is the *output function.*

subject to the following constraints:

- *Closure property*. Ω is closed under concatenation as well as under left segmentation
- *Composition (or semigroup) property*. For every pair of input segments ω, $\omega' \in \Omega$
  $\Delta(q, \omega) = \Delta((\Delta(q, \omega), \omega')$.

The interpretation of the functions Δ and Λ is as illustrated in Figure 7. Suppose that the system is in state $q \in Q$ at time $t_1 < t$ and we inject an input segment $\omega: <t_1, t_2> \rightarrow X$. The system responds to this segment and finishes in state $\Delta(q, \omega)$ at time t_2. Thus $\Delta(q, \omega)$ specifies the *final* state reached starting from q with input ω. It says nothing about the intermediate states traversed to arrive there. We observe the output of the system at time t_2 which is $\Lambda(\Delta(q, \omega), \omega(t_2))$. Again $\Lambda(\Delta(q, \omega), \omega(t_2))$ tells us the final output only and nothing about the interior outputs generated within the observation interval $<t_1, t_2>$.

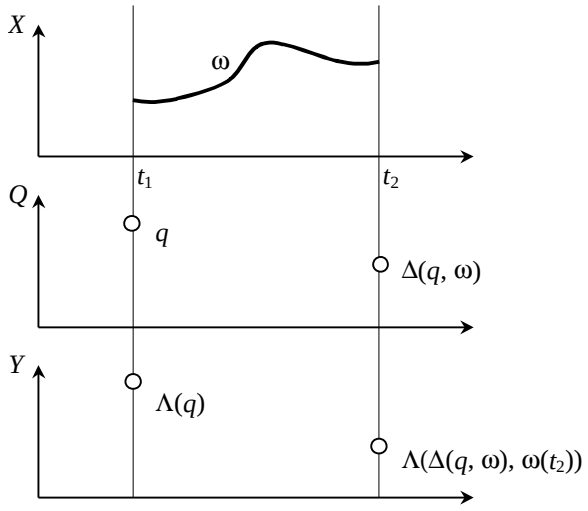

Figure 7 The basic system functions

The important composition property of Δ guaranties that the state set Q is in the position to summarize all relevant history of the system. To interpret the composition property refer to Figure 8. Suppose again that the system is in state $q \in Q$ at time t_1 and we inject an input segment $\omega: <t_1, t_2> \rightarrow X$. The system finishes in state $\Delta(q, \omega)$ at time t_2. Now lets select any time $t \in <t_1, t_2>$ to divide the segment ω into two contiguous segments $\omega_{t>}$ and $\omega_{<t}$. We inject $\omega_{t>}: <t_1, t> \rightarrow X$ which transfers q into $\Delta(q, \omega_{t>})$ at time t. We immediately apply $\omega_{<t}: <t, t_2> \rightarrow X$ and reach $\Delta(\Delta(q, \omega_{t>}), \omega_{<t})$ at time t_2. From the composition property it follows that $\Delta(\Delta(q, \omega_{t>}), \omega_{<t}) = \Delta(q, \omega)$. This means that the system can be interrupted at any point in time and its state

recorded. Then restarting from that state with the continuation of the input segment will lead to the same final state as if no interruption had occurred.

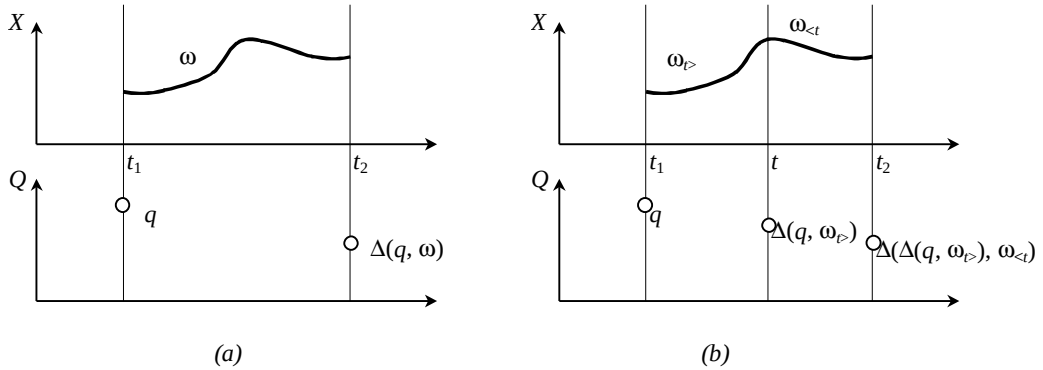

Figure 8 The composition property

2.4. Formal Definition of DEVS

Having informally illustrated DEVS modeling concepts, we now present the DEVS formalism in its formal guise. This form is needed for the mathematical work that will be presented later in the report. DEVS exhibits, in particular form, the concepts of systems theory and modeling. DEVS is important not only for discrete event models, but also because it affords a computational basis for implementing behaviors that are expressed in the other basic systems formalisms – discrete time and differential equations.

In this section we present the basic DEVS formalism[4] – the form that is used in atomic model construction. and then the DEVS formalism for coupled models.

A basic DEVS is a structure:

$DEVS = (X_M^{+}, Y_M^{+}, S, \delta_{ext}, \delta_{nt}, \delta_{ext}, \delta_{con}, \lambda, ta)$

where

$X_M = \{(p,v) | p \in IPorts, v \in X_p\}$ is the set of input ports and values;

$Y_M = \{(p,v) | p \in OPorts, v \in Y_p\}$ is the set of output ports and values;

S is the set of *sequential* states;

$\delta_{ext} : Q \times X_M^{+} \to S$ is the *external state transition function*;

$\delta_{nt} : S \to S$ is the *internal state transition function*;

$\delta_{con} : Q \times X_M^{+} \to S$ is the *confluent transition function*;

$\lambda : S \to Y^{+}$ is the output function;

$ta : S \to R_0^{+} \cup \infty$ is the *time advance function*;

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

[4] The presentation is of the Parallel DEVS formalism

Note that instead of having a single input, basic DEVS models have a bag of inputs. A *bag* is a set with possible multiple occurrences of its elements. Note the existence of a transition function, called *confluent*. It decides the next state in cases of collision between external and internal events.

The interpretation of these elements is illustrated Figure 9. At any time the system is in some state, *s*. If no external event occurs the system will stay in state *s* for time *ta*(*s*), a positive real number. Values of 0 and ∞ are also allowed. In the first case, the stay in state *s* is so short that no external events can intervene – we say that *s* is a *transitory* state. In the second case, the system will stay in *s* forever unless an external event intervenes. We say that *s* is *a passive* state in this case. When the resting time expires, i.e., when the elapsed time, $e = ta(s)$, the system outputs the value, $\lambda(s)$ and changes to state $\delta_{int}(s)$. Note output is only possible just before internal transitions.

If an external event $x \in X$ occurs before this expiration time, i.e., when the system is in total state (s,e) with $e \le ta(s)$, the system changes to state $\delta_{ext}(s,e,x)$. Thus the internal transition function dictates the system new state when no events have occurred since the last transition. While the external transition function dictates the system's new state when an external event occurs – this state is a determined by the input, *x* , the current state, *s*, and how long the system has been in this state, *e*, when the external event occurred. The confluent transition function, δ_{con} is applied in the case that $e = ta(s)$, that is when external and internal events are confluent. In all cases, the system is then is some new state *s*′ with some new time advance, *ta*(*s*′).

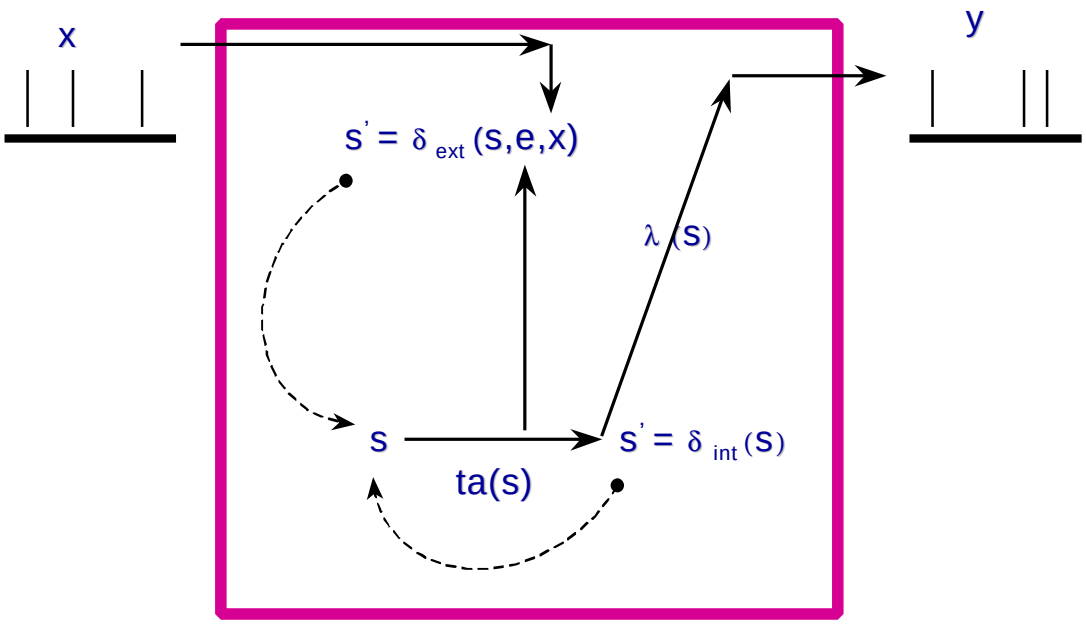

Figure 9 DEVS in action

We do not discuss here how a basic DEVS actually specifies a system-theory object. However, we mention that the necessary and condition for a DEVS to specify a system is called *legitimacy* – this requires that for any indefinitely extended sequence of internal transitions the accumulated time advance is monotonically increasing. For example, the presence of a cycle of transitory states would violate this condition – a simulation would "hang up" upon entering this cycle.

The DEVS coupled model formalism includes the means to build models from components. The specification includes the external interface (input and output ports and values), the components (which must be DEVS models), and the coupling relations. A DEVS coupled model (also called network) is a structure:

$N = (X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC)$

where

$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$ is the set of input ports and values;

$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$ is the set of output ports and values;

D is the set of the component names;

Component Requirements:

Components are DEVS models:

For each $d \in D$

$M_d = (X_d, Y_d, S, \delta_{ext}, \delta_{int}, l, ta)$ *is a DEVS*

with $X_d = \{(p,v) \mid p \in IPorts_d, v \in X_p\}$

$Y_d = \{(p,v) \mid p \in OPorts_d, v \in Y_p\}$

Coupling Requirements:

external input coupling connect external inputs to component inputs:

$EIC \subseteq \{((N, ip_N), (d, ip_d)) \mid ip_N \in IPorts, d \in D, ip_d \in IPorts_d\}$

external output coupling connect component outputs to external outputs:

$EOC \subseteq \{((d, op_d), (N, op_N)) \mid op_N \in OPorts, d \in D, op_d \in OPorts_d\}$

internal coupling connect component outputs to component inputs:

$IC \subseteq \{((a, op_a), (b, ip_b)) \mid a, b \in D, op_a \in OPorts_a, ip_b \in IPorts_b\}$

However, *no direct feedback loops are allowed*, i.e., no output port of a component may be connected to an input port of the same component i.e.,

$((d, opd), (e, ipd)) \in IC$ implies $d != e$.

with the following **interface constraints**:

$\forall ((N, ip_N), (d, ip_d)) \in EIC : range_{ip_N}(X) \subseteq range_{ip_d}(X_d)$;

$\forall ((d, op_d), (N, op_N)) \in EOC : range_{op_d}(Y_d) \subseteq range_{op_N}(Y)$;

$\forall ((d, op_a), (b, ip_b)) \in IC : range_{op_a}(Y_a) \subseteq range_{ip_b}(X_b)$.

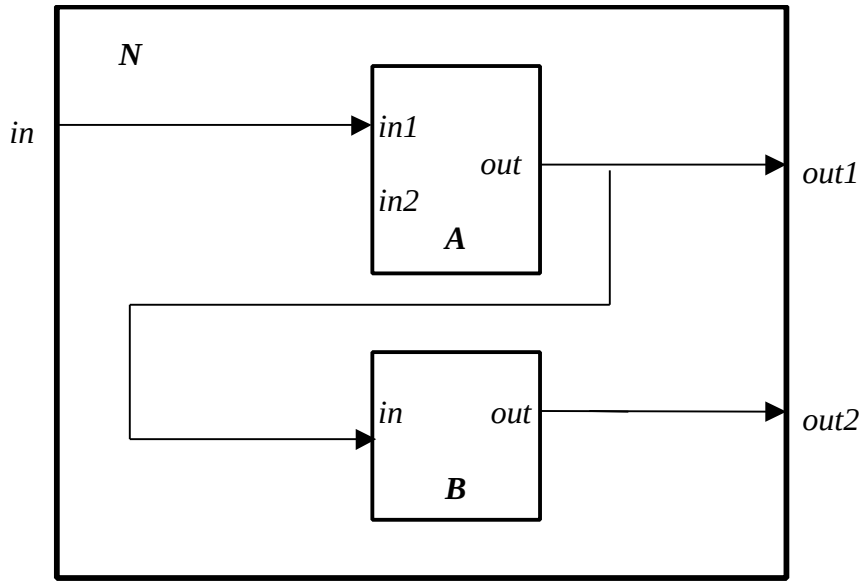

Figure 10 Coupled Model Example

In the example in Figure 10 for the coupled model (network) N,

- the input port set is {in}, output set is {out1,out2} with unspecified value sets
- the component name set D = {A,B},
- the coupling is,
  - external input coupling, EIC = {((N,in), (A, in1))},
  - external output coupling, EOC = {((A, out)}, (N,out1)), ((B, out), (N,out2))}
  - internal coupling, IC = {((A, out)}, (B, in))}.

Closure under coupling of DEVS is an important feature justifying hierarchical construction. *Closure under coupling* of a formalism requires that for any coupled model all of whose components are expressed in the formalism, there is an equivalent system, called the resultant, that is also expressible in the formalism. Thus, closure under coupling of DEVS is demonstrated by showing how to express the resultant of any coupled DEVS as a basic DEVS. This algorithmic construction provides the formal basis for the simulation cycle informally presented earlier – this provides the standard for verifying the correctness of implementations of DEVS simulation engines.

2.5. Other Formalisms - Discrete Time and Differential Equation

Although event-driven simulation is the basis for our work in predictive filtering, we must be able to accommodate the common forms of *time-driven* simulation – *discrete time*, in which time is advanced in fixed steps, such as in finite state machines, and *differential equation*, in which conceptually, time is continuous, but in practice, must be discretized as well. As with the DEVS formalism, both variants can be captured as system specifications with basic and coupled (network) forms.

A basic *Discrete Time System Specification* is a structure

DTSS = (X, Y, Q, d, l)

where

X is the set of inputs;

Y is the set of outputs;

Q is the set of states;

d: Q ´ X ® Q is the *state transition function*;

l : Q ® Y (Moore-type)

or

l : Q ´ X ® Y (Mealy-type) is the output function.

As with DEVS, the semantics of a DTSS is given by the system that it specifies. Informally, time advances in integer steps; at each step an input is consumed, causing a state transition, dictated by *d*, and an output produced which is either a function of the state alone (Moore-type) or the state and input (Mealy-type).

A basic *Differential Equation System Specification* is a structure

DESS = (X, Y, Q, f, l)

where

X is the set of inputs;

Y is the set of outputs;

Q is the set of states;

f : Q ´ X ® Q is the *rate of change (derivative) function*;

l : Q ® Y

Informally, the system specified by a DESS operates on a real number time base; the sets *X*, *Y* and *Q* are real valued vector spaces and the input, state and output trajectories are bounded piecewise continuous time functions.

Each of the formalisms, DTSS and DESS, has a definition of a coupled model and can be shown to be closed under coupling with suitable preconditions.

Differential equation systems have a standard representation as coupled models whose components are integrators and memoryless functions as illustrated in Figure 11. The *integrator* has one input variable *x*, one output variable *y*, and state variable *q*. It is described by the differential equation

$$\mathrm{d}\, q(t) / \mathrm{d}t = x(t)$$

which indicates that the derivative function, *f (q,x) = x*; the output *y* is equal to the current state

$$y(t) = q(t)$$

which indicates that the output function is the identity, *l(q) = q*.

In general, let $q_1, q_2, ..., q_n$ be the state variables and $x_1, x_2, ..., x_m$ be the input variables. Then a DESS corresponds to a set of first-order differential equations

$$
\begin{aligned}
\dot{q}_1(t) &= f_1(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t)) \\
\dot{q}_2(t) &= f_2(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t)) \\
&\ldots \\
\dot{q}_n(t) &= f_n(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t))
\end{aligned}
$$

Note that the derivatives of the state variables q_i are computed respectively, by functions f_i which have the state and input variables as arguments. This can be interpreted in the coupled network form of Figure 11. The state and input vector are input to the derivative functions f_i. These memoryless functions provide as outputs, the current derivative values dq_i/dt of the state variables q_i and these are forwarded to integrators. The outputs of the integrators are the state variables q_I. Note the feedback look in which the outputs of the integrators are coupled back to the inputs of the derivative functions.

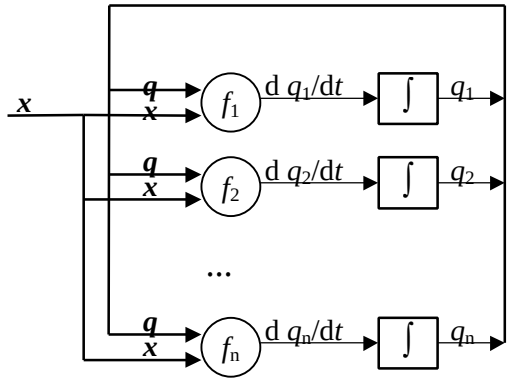

Figure 11 DESS – Structure of Differential Equation Specified System

2.6. System Morphism and Approximation

Much of the work needed to establish the ability of DEVS to embed other formalisms involves proving mathematical relations between systems that establish different kinds of equivalence. The basic concept is that of *system morphism* and we think of it relating a pair of systems, one "big", the other, "little" (Figure 12). A *homomorphism* from big system $S = (T, X, \Omega, Y, Q, \Delta, \Lambda)$ to small system $S' = (T, X, \Omega, Y, Q', \Delta', \Lambda')$ is a mapping, $h: \overline{Q} \rightarrow^{\text{onto}} Q'$, where $\overline{Q} \subseteq Q$,

where for all $q \in \overline{Q}$, $\omega \in \Omega$,

- $h(\Delta(q,\omega)) = \Delta'(h(q), \omega)$ *transition function preservation*

- $\Lambda(q)) = \Lambda'(h(q))$ *output function preservation*

To understand the preservation of the state transitions, we start the big system in one of the states q in the restricted set $\overline{Q}$. The map h yields a corresponding state in the little system, $q' = h(q)$. We then inject a segment ω into both systems – sending them to states $\Delta'(q', \omega)$ in S' and $\Delta(q, \omega)$, respectively. Preservation of state transitions requires that the latter states also correspond under h, that is, $h(\Delta(q,g(\omega'))) = \Delta'(q', \omega')$. For preservation of outputs, consider that when big system S is in state q and little system S' is in corresponding state $q' = h(q)$, the outputs observed in these states are $y = \Lambda(q)$ and $y' = \Lambda'(q')$,

respectively. Output preservations requires that y‘ = y. If $\overline{Q}$ = Q and h is a one-to-one, we call an *isomorphism*, and S and S‘ are said to be *isomorphic*.

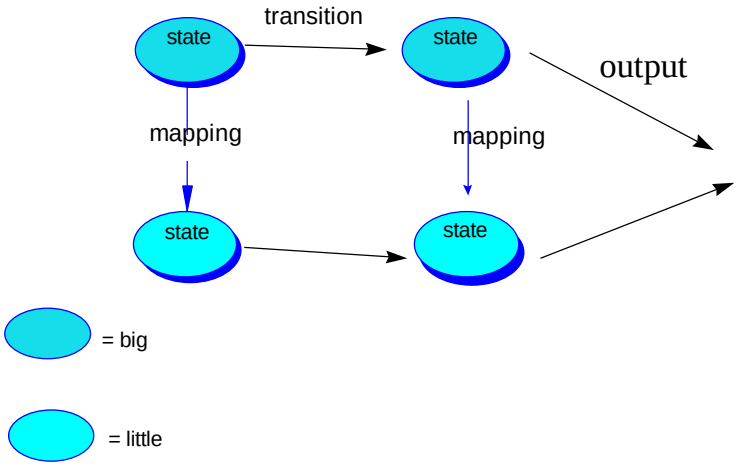

Figure 12 Commutative diagram showing the I/O system morphism.

When S‘ is a homomorphic image of S, both systems turn out to have the same I/O function behavior, but the difference is that the state space of S‘ may be much “smaller“ than that of S – all the state space Q of S is lumped by h onto that Q‘ of S‘.
When S and S‘ are isomorphic, there is no essential difference in their structural descriptions at the I/O system level, although they may differ very significantly at higher levels of specification involving more structural detail.

As illustrated in Figure 13, let there be a mapping from the states of a big model to those of a little model. Let the big and little models be in states that correspond under this mapping and consider corresponding transitions in these states for both models. Were the state mapping to be an exact homomorphism then the states after the transition would correspond under the mapping. However, in an *approximate morphism*, the states may differ. More specifically, small model may be in a state that differs from the mapped big model state. As shown, this is the error introduced in one transition. After two transitions, we compare the mapped big model state to the small model state, expecting them to be in error. Similarly, the mapped state trajectory of the big model and the little model state trajectory may be compared at every transition.

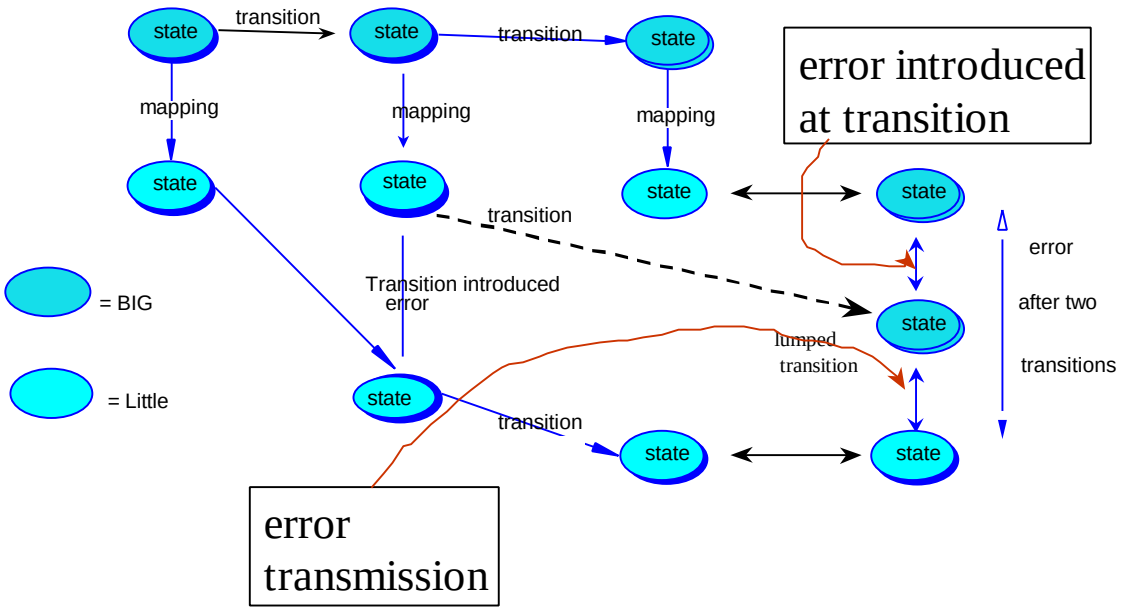

Figure 13 Error Accumulation Over Time

We will assume the existence of normed[5] vector spaces when needed[7]. Relative to such a norm, let e_n be the error introduced at transition n and let e_n the total error at transition n where $e_0 = 0$ (since we start in corresponding states). Let a_n be an upper bound on the growth of the error at transition n, i.e., $e_{n+1} \leq a_n e_n$. Let a and e upper bound the a_n and e_n, respectively, then

$$e_n \leq \frac{e(a^n - 1)}{a - 1}$$

Moreover, if $a = 1 + b$ where b is small and n is large then

$$e_n \leq n\,e$$

as n goes to infinity. Note that the effect of b disappears. This last form of error bounding will be the key to establishing DEVS approximations of continuous systems in the sequel.

2.7. Global vs Local (Component-wise) Representation

Before proceeding, we need to clarify our concept of representation of one formalism by another. Each of the formalisms DEVS, DTSS and DESS, define a subset of systems. Can the subset defined by DEVS "represent" the other two? The answer depends on the strength of the equivalence, or morphism, that we are interested in. Looking at Figure 28, we say that formalism *F can represent a formalism F' with morphisms M*, if for every system in the subset specified by *F'* there is a system in the subset specified by *F* that simulates it in the sense that there is a morphism of type M that holds between them.

[5] A norm is a metric on a vector space satisfying the triangle inequality, $||x+y|| \leq ||x||+|y||$ as the critical property

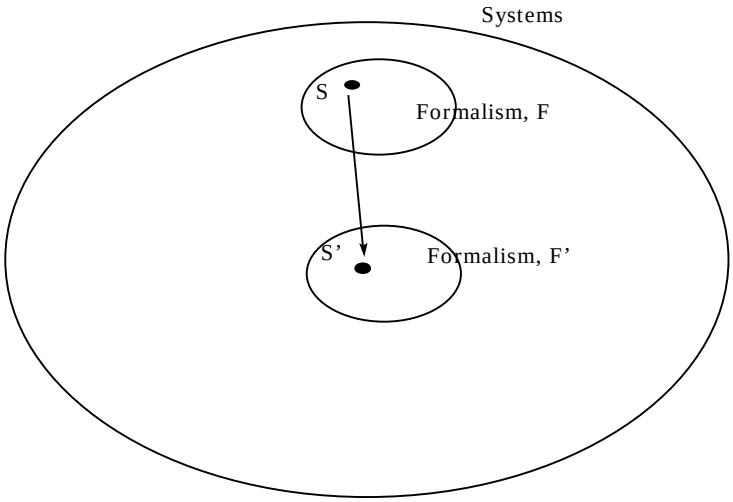

Figure 14 Formalism Representation

Thus we say that the DEVS formalism can represent a formalism at the coupled system level if for every coupled model specifiable in the formalism, there is a DEVS coupled model which simulates it with a coupled system morphism. We'll call this *strong, local* or *component-wise* simulation, since we don't have to preprocess the model to get a DEVS simulation. Instead, we convert the components of the model individually to a DEVS representation and then run the resulting DEVS coupled model to generate the original behavior. Contrast this with *weak* or *global* representation in which we only know how to create a morphism given the model expressed in the basic formalism but not as a coupled system. Then, any coupled model has to be first converted into its resultant form before being translated to a DEVS equivalent. This is rather impractical for complex coupled models.

3. THE DEVS BUS CONCEPT

A concept of *DEVS Bus* provides the framework we need to investigate the use of event-based simulation as a generic means of supporting predictive filtering in distributed simulation. It has been shown that traditional discrete event world views can be embedded into the DEVS formalism[8]. In such a "wrapping", models expressed in various event-based formalisms can be expressed in the basic DEVS formalism and interoperate as components within a DEVS coupled model. Figure 15 depicts an *extended DEVS Bus* in which this concept is extended to encompass the other major formalisms, DTSS (Discrete Time Systems Specification) and DESS (Differential Equation System Specification). The idea is to provide means to represent time-driven and continuous state/time models in discrete event form thus providing an all-DEVS, distributed simulation environment for any modeling domain[9]. For example, we could have rule-based DEVS models representing intelligent agents communicating through a distributed network represented by cellular automata and interacting over terrain represented by spatially continuous (partial) differential equation models. The message passing layer below the DEVS Bus can be any suitable network protocol to which DEVS has been adapted. In fact, DEVS-C++[10] has been mapped to the C++ version of the RTI (Run Time Infrastructure) and offers a testbed implementation of the DEVS Bus concepts.

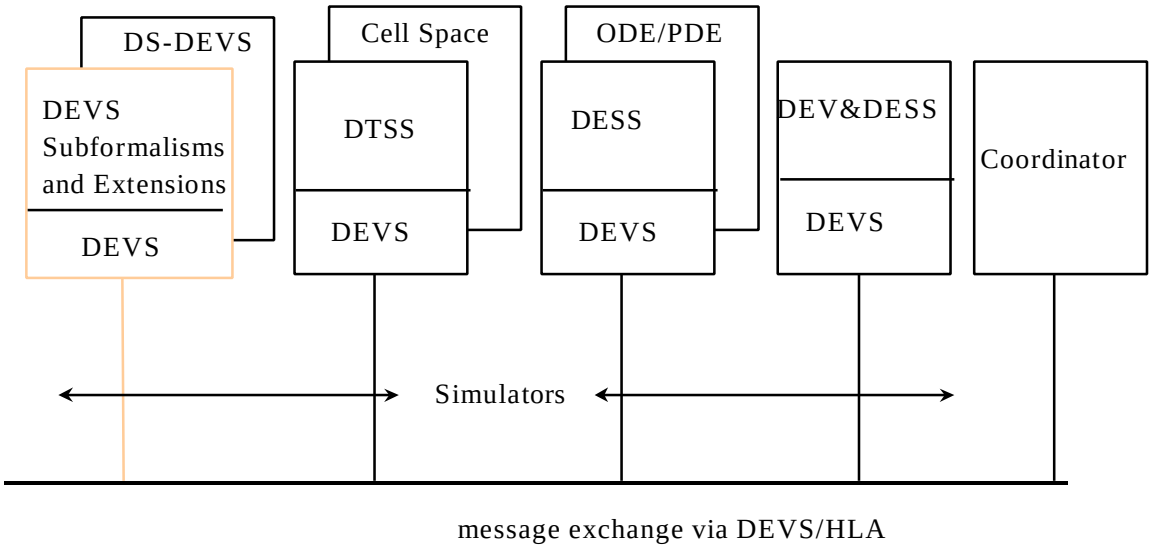

Figure 15 DEVS Bus

To make the extended DEVS Bus a reality requires that we have workable representations of the DTSS and DESS formalisms in DEVS. It is known that DEVS can exactly simulate discrete time models at both the component and network (coupled model) levels[6]. This allows us to develop the first of two approaches to DEVS simulation of differential equations systems. This first approach, the DEVS equivalent of traditional numerical integration, corresponds to the top pathway in Figure 16. The major step here is to discretize time to fixed time steps and employ standard numerical methods to obtain DTSS representations of the DESS components. Since this can be done in a component-wise manner, we have a component-wise simulation by DTSS of DESS coupled models. Now we can directly construct a DEVS coupled model to simulate the DTSS representation, and hence, the original DESS, coupled in the required component-wise manner. Of course, as indicated in the figure, error may be introduced in the DTSS approximation of the DESS model and this will carry-over to the DEVS simulation. Notice that since the DEVS simulation of the DTSS stage is error-free, the DEVS simulation will not suffer more error than the DTSS simulation.

However, in going second hand through a DTSS simulation we are not taking advantage of reduced computation and message passing that a direct DEVS representation might provide. Thus, for our second

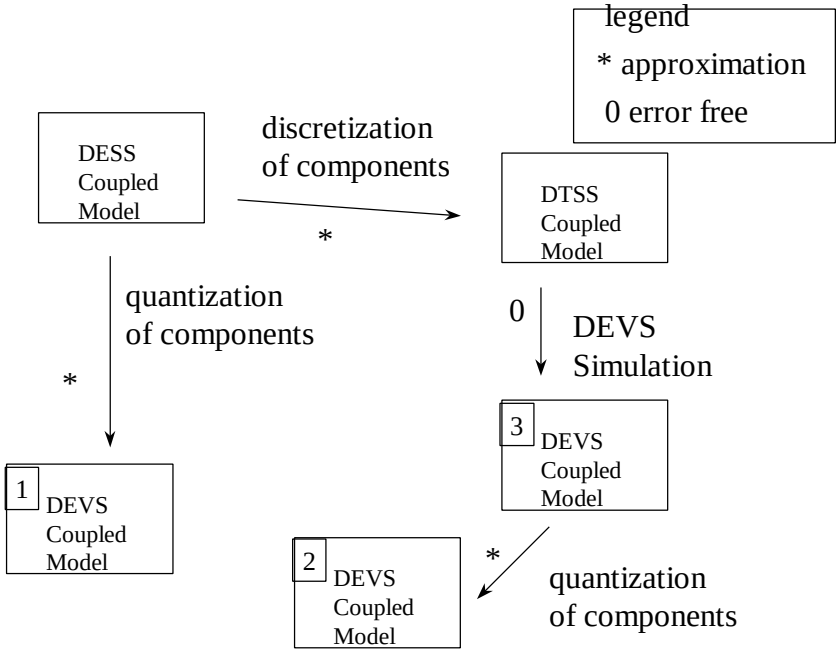

Figure 16 Approaches to DEVS Simulation of DESS Coupled Models

approach to DEVS representation of DESS (the left hand pathway in Figure 16) we recall that DEVS offers an alternative way of handling continuous time. Instead of advancing time in discrete steps, we can advance time based on discrete events, where an event is a significant change in an input, state or output variable. The key to handling the continuous variables of a DESS is to determine when a significant event occurs in a component. Such an event can then be reported to other components. We will introduce a significant event detector, called a *quantizer*, which monitors its input and uses a logical condition to decide when a significant change, such as crossing a threshold has occurred. The concept of quantization will formalize the operation of significant event detectors. A quantum is measure of how big a change must be to be considered significant.

To reduce local computation and message passing we would like to have the quantum size be large. However, the decision of what is a significant change cannot just be made locally since it also involves other components. We are really asking how big a change must be before we need to tell others about it. The answer to this kind of question is through error analysis of coupled models whose components are quantized. Put another way, we need to look at closed loop behavior, i.e., how any error propagates through a network where feedback loops exist. Such analysis is well known for numerical integration of differential equation systems. The theory of quantized systems extends this analysis to the realm of discrete event simulation.

4. DEVS REPRESENTATION USING CONVENTIONAL APPROACH

We consider approximate simulation of DESS by DTSS first at the global, then at the local, level. Using numerical integration such as the Euler integration method, a coupled model of integrator components can be component-wise simulated by a DTSS counterpart. The simulation is approximate for any time step $h > 0$. We will provide conditions under which the maximum error in a finite interval goes to zero as h goes to zero. We say that the DESS is exactly discretizable under these circumstances.

Our approach to DTSS representation will be first to formulate the required properties for systems specified at the I/O system level and then to specialize them to differential equation systems.

Theorem 1: Sufficient Conditions for DTTS Simulation at the I/O System Level

Let S by an I/O system have the following properties:

1. Lipschitz condition on state (upper bounding error growth): For small enough τ

$$\|\delta(q, x_{\tau>}) - \delta(q', x_{\tau>})\| \leq (1 + a\,\tau)\,\|q - q'\| \text{ for all } q,q' \text{ in } Q$$

2. Lipschitz condition on input (upper bounding the error introduced at each transition):

$$\|\delta(q, \omega_{\tau>}) - \delta(q, \rho_{\tau>})\| \leq k\,\|\omega_{\tau>} - \rho_{\tau>}\|$$

3. Bound on the input growth, for small enough τ,

$$max_{t \in <0,\,\tau>}\ \|\omega_{\tau>} - \omega(0)_{\tau>}\| \leq M_{max}\ t$$

4. Norm preserving output map:

$$\|\lambda(q) - \lambda(q')\| \leq \kappa\|q - q'\|$$

this is clearly true if the state appears as the output.

Then there is a DTSS S_{DTSS} and an approximate morphism at the I/O system level from S_{DTSS} to S where the error goes to zero as the time step, h goes to zero.

Proof: Define S_{DTSS} so that it has the same input, state, and output sets as S. Since the state sets are both Q so we use the identity mapping as the approximate morphism. Consider the input segment ω and its discretized version, $\nu = w(0)\,w(h)\ldots w((n-1)h)$. Define $\delta_{DTSS}(q, x) = \delta(q, x_{h>})$, where $x_{h>}$ is the constant input segment having value x for period h. The output functions of S_{DTSS} and S are the same.

The proof consists of two parts:

1) We must bound the state error introduced at each time step. Using this and the error growth bound in condition 3., we apply the error growth approach in Section 1.6 to bound the error in the state trajectory.

2) We must bound the error in the output trajectories generated from the same starting state by both systems. This we can do immediately given the norm preserving condition (4) on the output map and the bound on the error in the state trajectory from 1).

The number of time steps in the interval <0,T> is

$N = T/h$

The error introduced at the end of such time step is

$$\begin{aligned}\varepsilon &= \|\delta(q, \omega_{h>}) - \delta_{DTSS}(q, \omega(0))\| \\ &= \|\delta(q, \omega_{h>}) - \delta(q, \omega(0)_{h>})\| \\ &\leq k\| \omega_{h>} - \omega(0)_{h>}\| \text{ (using condition 2.)} \\ &\leq kh\, max_{te<0,h>} \| \omega_{h>}(t) - \omega(0)\| \text{ (definition of trajectory norm)} \\ &\leq k M_{max} h^2 \text{ (using condition 3.)}\end{aligned}$$

Since the magnification of the error in each time step is given by condition 1. as $(1 + a\,h)$, and we can make h as small as desired, we use error bound in Section 1.6 for $a = 1 + b$. Thus the accumulated error is

$$e_T \leq N\varepsilon = k M_{max} T h$$

and the error goes to zero as h goes to zero.

4.1. DTSS Simulation at of a DESS Integrator

Let S be a DESS integrator. The transition function of the integrator is:

$$d_S(q, w_{t>})) = q + \int_0^t w(t)\,dt \ .$$

where $\omega_{t>}$ is an admissible input segment. The output of the integrator is its state: $l(q) = q$. The transition function of the discrete integrator employing Euler integration is an extension of the simple transition function:

$$D(q, w_{t>}) = q + x*t$$

where $\omega_{t>}$ is represented by a piecewise constant segment $x_{t>.}$.

Under suitable constraints on the derivative of its input segments, the integrator satisfies all the conditions of the exact discretizability in the Theorem 1 just given.

We say that an input segment, ω is *BCS (Bounded, Continuous and Smooth*), if

1) ω is a bounded, continuous function of its domain and
2) ω is smooth (its derivative, $\frac{d}{dt}w(t)$ is finite at every point, t in its domain) and the derivative function, ω' is bounded ($\omega'(t) = \frac{d}{dt}w(t)$)

Proposition: **DTSS Simulation of DESS integrator**

The DTSS integrator with a BCS input segment set is an exactly discretizable system.

Proof: Using the integrator transition function it is easy to show that the integrator satisfies the two Lipschitz conditions. Indeed, condition 1) is satisfied because $\|\delta(q, x_{\tau>}) - \delta(q', x_{\tau>})\| = \|q - q'\|$, i.e., $a = 0$. For condition 2) we have:

$\|\delta(q, \omega_{\tau>}) - \delta(q, \rho_{\tau>})\|$

$= \| \int_0^t w\text{-}(t) - r(t) dt \|$

$\leq \int_0^t \| w\text{-}(t) - r(t) \| dt$

$= \|\omega - \rho\|$

I.e., k = 1.

Now for condition 3 which bounds the input growth for small enough τ. This follows from the bound on the derivative of the admissible input segments. We can identify $M_{max} = max \{ \| \frac{d}{dt} w(t) \| | t \in dom(\omega)\} = \|\omega'\|$, the largest derivative in the interval $<0, \tau>$. This is because the mean value theorem of calculus which states that

$\omega_{\tau>}(t) - \omega_{\tau>}(0) = t \times \frac{d}{dt} w(t')$ for some t' in the interval $(0, t)$.

Thus,

$max_{t \in <0, \tau>} \| \omega_{\tau>} - \omega(0)_{\tau>} \| \leq t \; max \{ \| \frac{d}{dt} w(t) \| | t \leq t\} \leq M_{max} \; t$

Since the output function is the identity it preserves the state norm. Thus the error bound for the integrator is given by $e_T \leq \|\omega'\| T h$. In other words, the larger the rate of change of input, the smaller the time step must be to reduce the error below a given tolerance.

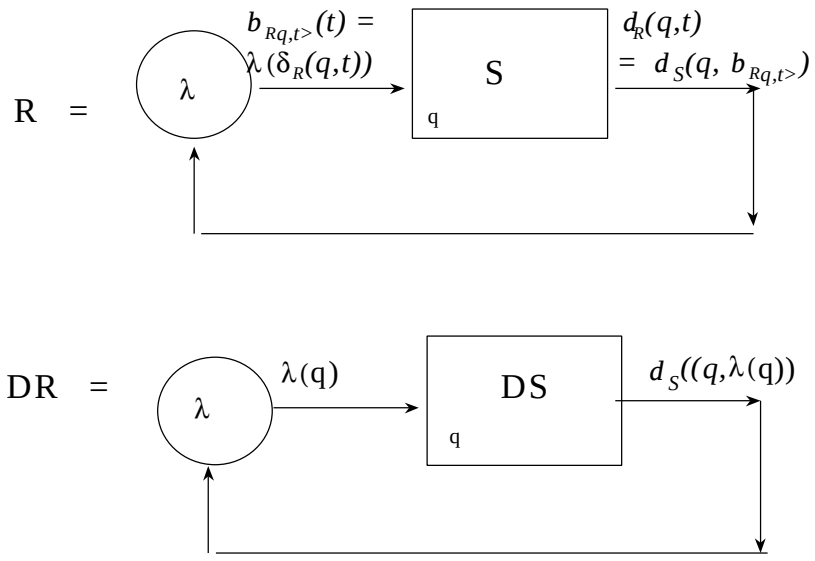

Figure 17 DTSS Closed Loop Simulation of a System

4.2. Simulation of Coupled Systems By DTSS

Theorem 2: Component-wise Simulation of Coupled Systems by DTSS with arbitrarily small error

A well-defined coupled model can be component-wise simulated with an arbitrarily small error by a DTSS coupled model.

Rather than work with an arbitrary coupled model we will consider closing the loop around a single component (Figure 17). This will demonstrate the essence of what is involved in a full scale proof. By feeding back the output of the system, S to its input, the resultant is an input-free system, R. Let DS be the DTSS representation of S and let DR be the coupled version of R, which, as shown, preserves the coupling from output to input. To show that we can make DR simulate R as closely as desired by reducing *h*, we would like to use Theorem 1for the open loop case. It turns out that the most of the properties required in Theorem 1 still apply. The one exception is the input property, condition 3). This property has to be reconsidered since, due to feedback, we can't specify the input's properties independently of the system itself.

Theorem 3: Feedback Simulation of a System by a DTSS with arbitrarily small error

Let S be a system with the following properties:

1. (Replacing Input Property 3)
   Small excursion behavior: S stays close to its initial state for short duration inputs. For small t, for all w possible as feedback input (see below) there exists $m > 0$ such that

   $$|| d_S(q,w_{t>})) - q|| \leq m\ t$$

2. Lipschitz condition on state (upper bounding error growth): For small enough τ

$$\|\delta_S(q, x_{\tau>}) - \delta_S(q', x_{\tau>})\| \leq (1 + a\,\tau)\,\|q - q'\| \text{ for all } q,q' \text{ in } Q.$$

3. Lipschitz condition on input (upper bounding the error introduced by each segment):

$$\|\delta_S(q, \omega_{\tau>}) - \delta_S(q, \rho_{\tau>})\| \leq k\,\|\omega_{\tau>} - \rho_{\tau>}\|$$

4. Norm preserving output map:

$$\|\lambda(q) - \lambda(q')\| \leq \kappa\|q - q'\|$$

Then a discretized version of S with feedback can simulate S with feedback with arbitrarily small error.

The proof is given in Appendix 1.

4.3. Discretized Simulation of Coupled DESS with Arbitrarily Small Error

We now specialize the theorem to DESS by considering the system *S* to be an integrator. Replacing the output function *l* by the derivative function, *f* results in Figure 18. It is easy to see that this represents the standard form of a first order differential equation system without input. Consider, the condition:

$$\|f(q) - f(q')\| \leq k\|q - q'\|$$

This was the norm preservation requirement of the output function, but now can be identified as the standard Lipschitz condition on the derivative function required for unique solutions[7].

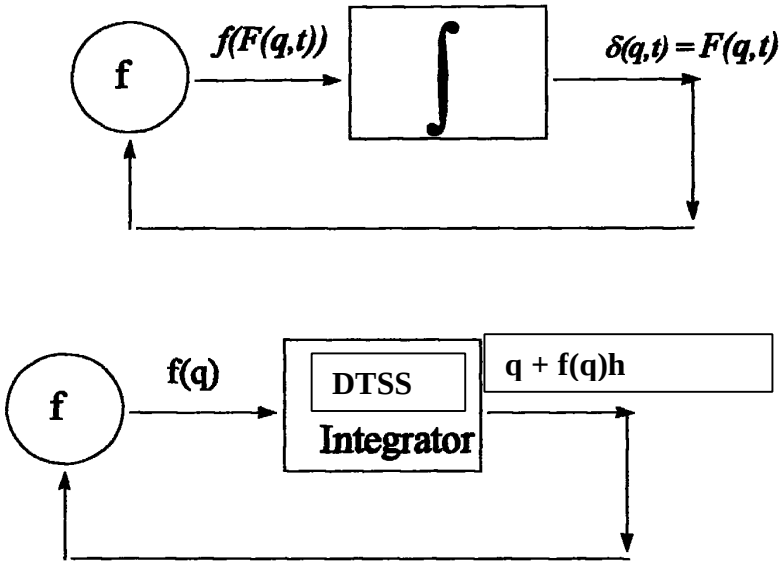

Figure 18 DTSS Closed Loop Simulation of DESS Integrator

Proposition: Closed loop simulation of a first order DESS by a Discrete Integrator with arbitrarily small error

A discretized version of the integrator can simulate a first order DESS with arbitrarily small error provided that the derivative function is bounded from above.

Proof: Recall that for the integrator we have $d_S(q,w_{t>}) = q + \int_0^t w(t)dt$. In Theorem 2 , the Lipschitz conditions on state and input were already verified. We consider the following:

Small excursion behavior: For any ,$w = f(q)_{t>}$,

$$\begin{aligned} &|| \; d_S(q,w_{t>}) - q \; || \\ &= || \int_0^t w(t)dt \; || \leq t \; max\{||w(t)|| \; | t \leq t\} \\ &\leq t \; max\{ \; || f(q)| \; || \; \} \end{aligned}$$

Thus $m = max\{ \; || f(q)| \; || \; \} = f_{max}$.

As already said, the norm preservation requirement is really the Lipschitz condition on the derivative in disguise and comes along with the fact that the system forms a DESS, with $k = L$. Recall $k = 1$. Thus, $\boldsymbol{M} = f_{max} L$ and the error bounded by

$$e_T \leq h T \; L f_{max} .$$

4.4. DEVS Representation of DESS via DTSS Simulation

Recall the two approaches to DEVS representation of DESS. The first is to employ standard numerical methods which result in a DTSS simulation of the DESS. Since a DEVS can component-wise simulate a DTSS coupled model with zero error, we have that a DEVS can component wise simulate a DESS. The second is to map the DESS model directly into a DEVS. Taking the first approach, illustrated in Figure 19, leads to the following

Theorem 4: Component-wise Simulation of DESS by DEVS with arbitrarily small error

Let there be a well-defined DESS coupled model that can be component-wise simulated with an arbitrarily small error by a DTSS coupled model. Now a legitimate DEVS coupled model can simulate the DTSS coupled model with zero error. Thus there is a legitimate DEVS coupled model that can similarly simulate the DESS with an arbitrarily small error.

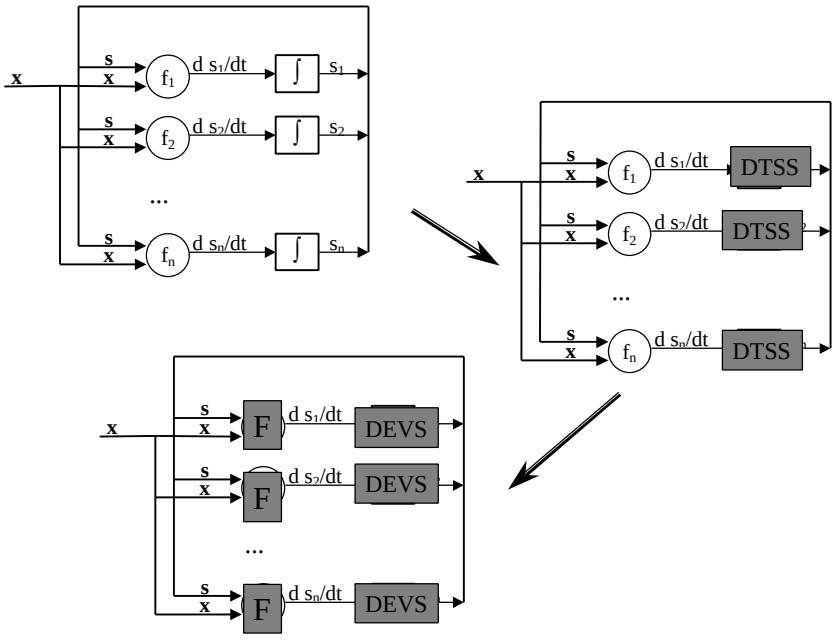

Figure 19 DEVS Simulation of DESS Coupled Models via DTSS simulators

5. QUANTIZATION: AN ALTERNATIVE APPROACH DEVS REPRESENTATION

The second approach to DEVS representation of DESS will be based on a concept of quantization. It requires a change in viewpoint. To simulate systems with a digital computer requires that we approximate a continuous trajectory with a finite number of values in a finite time interval. One way, we have just seen to discretize the time base to obtain a DTSS approximation. Rather than discretize the time base we may partition the trajectory into a finite number of segments each of which has a finite computation associated with it. One way to do this is to quantize the value space as illustrated in Figure 20a). Quantization is a very general concept applicable to any value set, *X*. Figure 20 b) shows a partition π on *X* with a finite number of partition blocks (equivalence classes). Let [*x*] denoting the block containing point, *x* and let $\hat{x}$ be the representative of the block containing *x*. We'll return to consider the choice of representative.

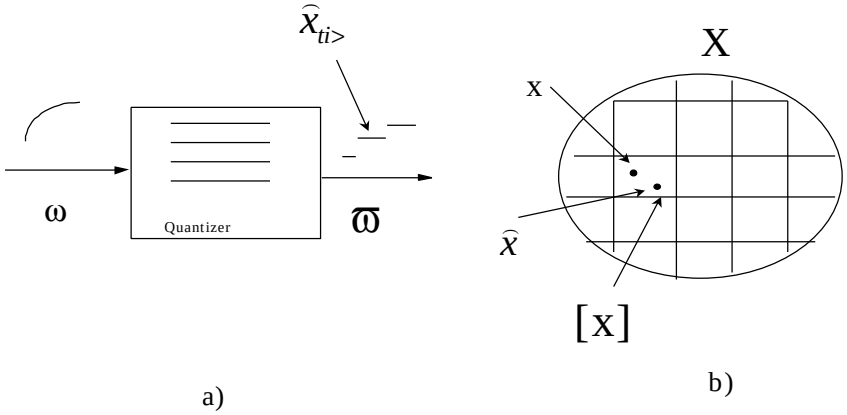

Figure 20 Quantization Principles

Let ω be a trajectory on interval <0,τ>mapping into X. Then a *quantization* of ω, based on π, is a piecewise constant segment over the same interval, ϖ defined by:

$\varpi = \hat{x}_{t1>}\ \hat{x}_{t2>} \ldots \hat{x}_{tn>}$ where each constant segment $\hat{x}_{ti>}$ point-wise represents a segment of ω whose range lies entirely in block [$\hat{x}_{ti}$], i.e., the representative of ω(t) is $\hat{x}_{ti}$ for all points *t* in the domain of $\hat{x}_{ti>}$. To pick out the longest such segments, let *ti* be the largest time *t*, such that $\omega(t)$ is in [$\hat{x}_{ti}$]. Thus a quantization is a maximum length segmentation[6] and the domain of each constant segment is well defined.

For a special quantization, we let the value set be a finite interval of the reals and let the partition π be given by an integer mesh, {…,-D,0,D,2D,…,nD….} where a block is an interval [nD,n+1D) and D is called the *quantum.* We call such a quantization, *quantum-based*. Given a number *x*, the partition block [*x*] is computed as the integer floor of *x*/D. The latter value can also be designated as the representative of *x*,

[6] This is a canonical means of breaking a segment into parts[5]

corresponding to rounding down. Other choices of representative are possible, for example, the half point, floor(x/D) + D/2. Quantization of an n-dimensional real valued space can be done in many ways. One way is through the independent quantization of each dimension.

A *quantizer* is a system defined at the I/O function level, with its only function, Q_π defined by $Q_\pi(\omega) = \varpi$. Such systems have a number of properties to which we will return.

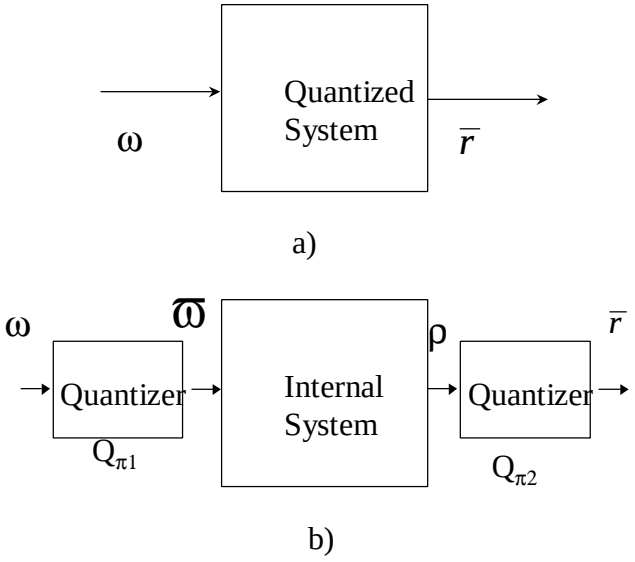

Figure 21 Quantized Systems

5.1. Quantized Systems

To study representation of DESS by DEVS we introduce quantized systems as in Figure 21a). A *quantized system* can be decomposed into, hence has the same behavior as, a system with input and output quantizers, as in Figure 21b). We call the system sandwiched between the quantizers, the *internal system*. The reason for introducing quantized systems is that, as we shall show, they are exactly simulateable by a DEVS at the global level, opening the way to study their local (strong, component-wise) simulateability.

We denote a quantized system by $< Q_{\pi 1}, S, Q_{\pi 2} >$, the sequence of the input quantizer, internal system, and output quantizer, respectively. The output segments of quantized system are clearly piecewise constant. In the following, we'll restrict attention to continuous inputs segments with quantum-based quantizers. Such a quantized system is realizable as a DEV&DESS where the crossing of the partition block boundaries are implemented as state events.

5.2. Exactly Quantizable Systems

The quantum, D_{in} of the input quantizer determines the *input sensitivity* of the quantized system. Similarly, the quantum, D_{out} of the output quantizer determines the *output resolution*. Next we formulate conditions under which a quantized system approaches its internal system as both D_{in} and D_{out} go to zero. We call such systems *exactly quantizable.*

First, we note that no error propagation problem arises when we let D_{out} get smaller and smaller. The difference between the output trajectory of the internal system and its quantized counterpart emerging from the output quantizer is always at most D_{out}. Thus, we can safely make D_{out} as small as needed to make the quantized output match the output of the internal system as small as we like. So from here on, we won't worry about D_{out}.

However, to make the output of the internal system match that of the original system, we need to consider the size of the input quantum, D (we will drop the subscript, form here on). Here, we must take into account the accumulation of the error introduced by the quantization. This occurs because the internal system has memory and can wind up in different states due to the different inputs. Moreover, as we have seen, once states have diverged, they may be forced into even greater divergence due to the amplification dynamics of the system. Thus, we must come up with conditions on the system that enable us to control this error propagation to as small as level as desired by reducing the input quantum.

First we formulate constraints on the system's input segment set that allow each of its segments to be given an mls segmentation in a uniform manner as a function of the quantum size D.

An input segment w is *uniformly segmentable* if there is a small enough quantum size, D_0, and positive numbers $Max > Min > 0$ such for every $D_m = D_0/2^m$ $(m = 1,2,3\ldots)$

1) there is a finite set of boundary crossings for a quantization based on D_m i.e., the set $\{ t: w(t) = nD_m$, for some integer $n\}$ is finite,

2) let $t_1 \ldots t_n$ be the set of inter-crossing times ($t_i = t_i - t_{i-1}$). Then for $i = 1 \ldots n$,

$$D_m/Max = t_{min} \leq t_i \leq t_{max} = D_m/Min.$$

This means that as we refine the quantization (by halving the quantum size at each stage), the upper and lower bounds on the lengths of the representing segments behave in a very well defined manner: both are proportional to the quantum size (D_m). In the following theorem, we will see that the smallest representing segment length, t_{min} upper bounds the number of segments, while the largest representing segment length, t_{max} upper bounds the error introduced by each segment.

An input segment set, W is *uniformly segmentable* if every one of its segments is uniformly segmentable.

Theorem 5: Sufficient Conditions for Exactly Quantizable Systems

Let $< Q_{\pi 1}, S, Q_{\pi 2}>$ be quantized system and let its internal system S, have the same properties as in Theorem 1, except that the input property 3) is replaced by:

3. The input segment set of S is uniformly segmentable.

Then there is an approximate morphism at the I/O system level from $< Q_{\pi 1}, S, Q_{\pi 2}>$ to S where the error goes to zero as the input quantum, D goes to zero.

Proof: The state sets of the quantized system and its internal system are both Q so we use the identity mapping as the approximate morphism. As illustrated in Figure 21, consider the input segment ω and its quantized version, $\varpi = \hat{x}_{t1>} \; \hat{x}_{t2} \ldots \hat{x}_{tn>}$ both sent to S. The proof consists of two parts:

1) We must bound the state error at the end of the corresponding generator segments. Using this and the error growth bound in condition 3., we apply the error growth approach in Section 1.6 to bound the error in the state trajectory.

2) We must bound the error in the output trajectories generated from the same starting state by both systems. This we can do immediately given the norm preserving condition (4) on the output map and the bound on the error in the state trajectory from 1).

By condition 1 (uniform segmentation), we have bounds *Min* and *Max* and a sequence of quantum sizes, $D_m = D_0/2^m$ $(m = 1,2,3\ldots)$, such that for each D_m,

$$D_m/Max = t_{min} \leq t \leq t_{max} = D_m/Min$$

Thus, if w has length T, the number of segments in its domain is:

$$N \leq T/ t_{min} = T\,Max / D_m$$

The error introduced at the end of such a segment is

$$\begin{aligned} e = \| d(q, w_{t>}) - d(q, x_{t>}) \| &\leq k \| w_t - x_{t>} \| \quad \textit{(using condition 2.)} \\ &\leq k t \max_t \| w_{t>}(t) - x_{t>}(t) \| \\ &\leq k t D_m \end{aligned}$$

The error is thus upper bounded by

$$e_{max} = k D_m t_{max} = k D_m^2/Min.$$

Since the magnification of the error is given by condition 1. as $(1 + a\tau)$, we use error bound in Section 1.6 for $a = 1 + b$, and the accumulated error is

$$e_T \leq N e_{max} = k D_m T\,Max /Min.$$

Thus, by choosing an m large enough so that the quantum size, D_m is as small as we need, we can make the error as small as we like.

Given an exactly quantizable system, S and an error tolerance, suppose there is a quantum value for the input quantizer which keeps the error within tolerance. Then there is also a largest quantum value that keeps the error within tolerance. We call this quantum value the input *sensitivity* of system S relative to the error tolerance and we say that S is *sensitizable* relative to the error tolerance. It may also happen that a system is *exactly* represented by its quantized version (i.e., with an error tolerance of zero). In this case, we omit the qualification, and refer to the *sensitivity* of the system as the largest quantum size that for which it is exactly represented.

5.3. Quantized Integrator: Approximation of DESS

Let S be a DESS integrator. Then $< Q_{\pi 1}, S, Q_{\pi 2}>$ will be called a *quantized integrator*. The transition function of the quantized integrator is an extension of the simple transition function:

$$D(q, w_{t>}) = q + x*t$$

where $\omega_{t>}$ is represented by a piecewise constant segment $x_{t>}$ where x is the block representative of $\omega_{t>}(0)$. The output of the integrator is its state: $l(q) = q$.

Under suitable constraints on the derivative of its input segments, the integrator satisfies all the conditions of the exact quantifiability Theorem just given.

Theorem 6: **Quantized Simulation of DESS integrator**

The DESS integrator with a uniformly segmentable input segment set is an exactly quantizable system.

Proof: By assumption, the input segment set is uniformly segmentable and remaining three conditions of Theorem 5 have already been shown to be satisfied.

Appendix 2 Characterizes uniformly segmentable input sets for the continuous bounded set of inputs that are admissible for DESS.

5.4. Coupled Systems with Quantized Components

A coupling of quantized components is a coupled system specification (Chapter 5) in which the components are quantized systems and for which quantizers are present at the input and output interfaces (Figure 22). As usual, our first concern is whether such specifications result in well-defined systems. Such networks operate very much like DEVS networks. Indeed, we shall see that they can be directly represented by DEVS equivalents.

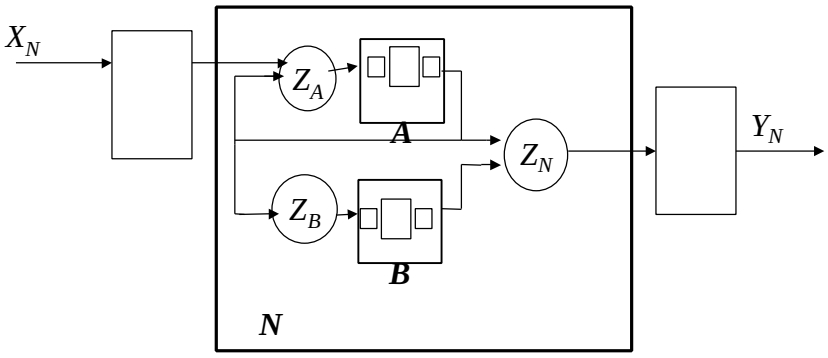

Figure 22 Coupled Model of Quantized Systems

Theorem 7: Quantized Moore Systems are Closed under Coupling

Proof (sketch). We establish that coupling of quantized Moore systems has a well-defined resultant. For a global state of such a coupled model, there is a well defined next event time. This is the smallest of all times to next boundary crossing of the output quantizers of the components. Not considering external input, until the next event time, all outputs of the output quantizers remain constant, and therefore all inputs to input quantizers remain constant. (The interior systems of the components may however, be undergoing state transitions in response to these constant inputs.) Consider an *internal event* which occurs when the

time to next event has elapsed without an intervening *external event* (defined below). At such an internal event one or more outputs change, and the changed outputs are sent to the output quantizer as well as to the influenced components via the component interface maps. If boundaries of the input quantizers are crossed, these changes are transmitted as new input levels to the transition functions of the influenced interior systems. Since the latter are Moore systems, their individual next event times are greater than zero, and the global next event time is also greater than zero. Now consider the response of the network to an external event, i.e., an input causing a change in the output of the input quantizer. Here there the external input causes a change in the output of the input quantizer before the next internal transition. When such an event occurs, the new level is transmitted through the input interface maps to the input quantizers of the receiver components. The same subsequent behavior can be observed as in the internal event case. Both internal and external events can occur at the same time, i.e., a component output quantizer crosses a boundary at the same time that the input quantizer does. This *confluent behavior* is resolved by the component interface functions.

Closure under coupling follows since now the resultant is a system that is sandwiched between the input and output quantizers.

Next consider the relationship of a coupling of quantized systems to a coupling of their interior systems. Suppose each of the interior systems is quantizable with zero error. We assume that in each case, the input quantization is that of the maximum quantum size given by the sensitivity. However, we can choose the output quantizations. Under what conditions on the latter quantum sizes will the coupling of quantized system exactly represent the coupling of interior systems?

Figure 23 illustrates the two sources of error: a) the output quantization may be too coarse for an input quantization it influences, and b) misaligned quantizations. In case a), a change in output of the original sending component would have been noticed earlier by receiving component than by its counterpart in the quantized network. In other words, the sending component uses too liberal a definition of significance for the needs of the receiver. In case b), even though the output quantum is smaller than the input one, the corresponding boundaries do not line up exactly with each other.

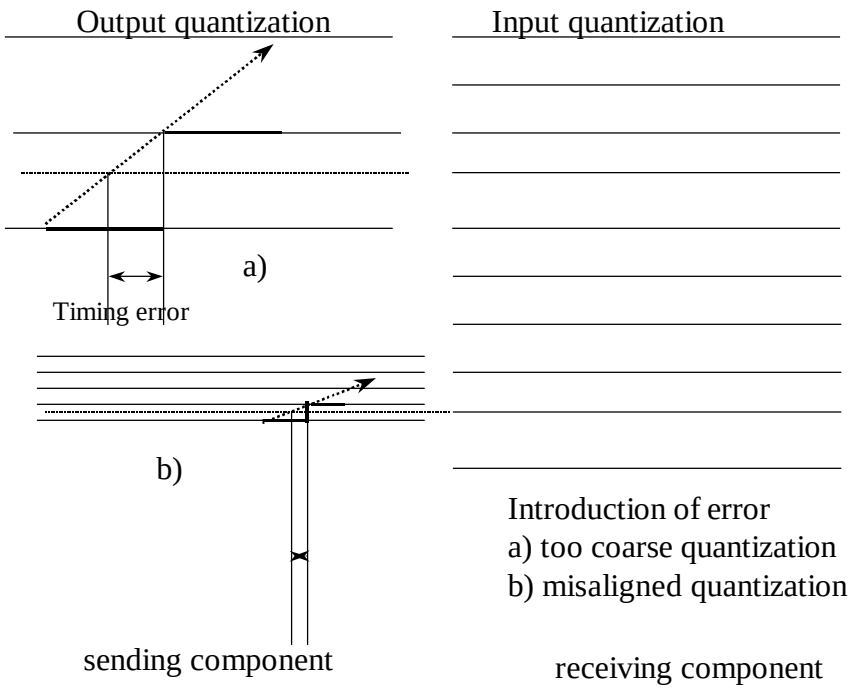

Figure 23 Conditions For Error-Free Coupling

Partition refinement[7] is the formal requirement that obviates the two types of errors. With it we can state:

Theorem 8: Component-wise simulation of Systems with Finite Sensitivities

Given a coupled model of quantizable systems with given sensitivities. A coupled model of quantized systems can component-wise simulate the given coupled model under the conditions that the input quanta are the respective sensitivities and the output quantizations refine the input quantizations they influence.

Figure 24 illustrates an application of this theorem. The state of a sender (A) must be updated for two receivers (B and C) with different sensitivities. The theorem says that we must use output quantizations that refine the input quantizations that they influence. However, it does not require us to use the smallest quantum size 1 (that of C) for both receivers. We can use a quantum of 10 at the output of the sender for receiver B without loss of fidelity, while employing the finer quantization for output to receiver C. Anticipating DEVS simulation of the next section, in *this sensitivity-based updating*, the total number of messages sent by DEVS simulators of the quantized systems can be significantly reduced. As indicated by the reverse arrow, feedback (e.g. from B to A) allowed in such couplings and its benign effect has been accounted for in formulation of the theorem. Of course, as illustrated in the figure, information about others' quantum sizes must be retained in the component simulators.

[7] One partition refines another if every equivalence class in the first is contained in an equivalence class of the second.

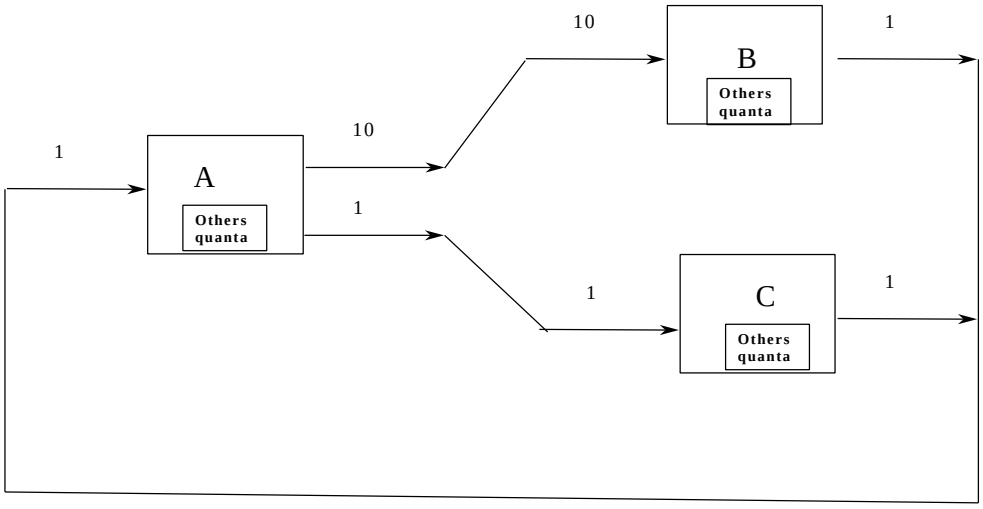

Figure 24 Selective Updating based on Sensitivity

5.5. Quantized Simulation of Coupled Systems with Arbitrarily Small Error

Let each component in a coupled model be an exactly quantizable system. Consider the coupled model with every component replaced by a quantized version. Under what conditions can the quantized coupled model simulate the coupled model with an arbitrarily small error?

As in the discrete time case, for simplicity we consider only a single system S shown in Figure 25. By feeding back the output of the system, S to its input, the resultant is an input-free system, R.

Let QS be the quantized version of S. We consider only an output quantizer with quantum size D. We let QR be the coupled quantized version of R, which, as shown, preserves the coupling from output to input. To show that we can make QR simulate R as closely as desired by reducing D, we would like to use Theorem 5 for the open loop case. It turns out that the most of the properties required in Theorem 5 still apply. The one exception is the uniform segmentation property of the input. This property has to be reconsidered since, due to feedback, we can't specify the input's properties independently of the system itself.

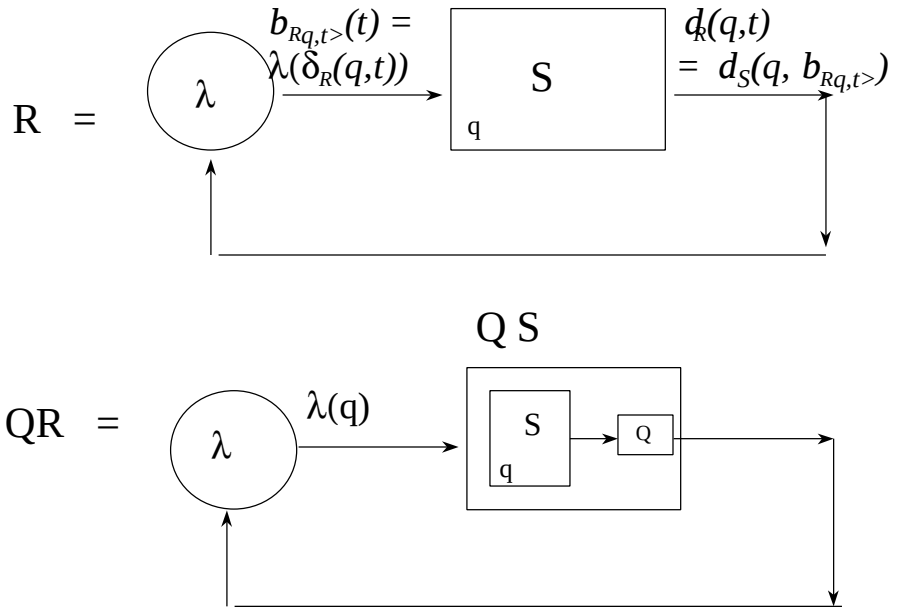

Figure 25 Quantized Simulation of Closed Loop System

Theorem 9: Feedback Simulation of a System by a Quantized System with arbitrarily small error

Let S be a system with the following properties:

4. (Replacing Input Segmentation Property)

a) Small excursion behavior: S stays close to its initial state for short duration inputs. For small t, for all w possible as feedback input (see below) there exists $m > 0$ such that

$$|| \ d_S(q,w_{t>})) - q|| \leq m \ t$$

b) Boundary Crossing Times: For small t and D, for all x in the range of l, there exists $M > 0$ such that if $|| \ d_S(q, x_{t>})) - q|| = D$ then $t \leq D/M$

5. Lipschitz condition on state (upper bounding error growth): For small enough τ

$$\|\delta_S(q, x_{\tau>}) - \delta_S(q', x_{\tau>})\| \leq (1 + a\,\tau)\,\|q - q'\| \text{ for all } q,q' \text{ in } Q.$$

3. Lipschitz condition on input (upper bounding the error introduced by each segment):

$$\|\delta_S(q, \omega_{\tau>}) - \delta_S(q, \rho_{\tau>})\| \leq k \,\|\, \omega_{\tau>} - \rho_{\tau>} \,\|$$

4. Norm preserving output map:

$$\|\lambda(q) - \lambda(q')\| \leq \kappa\|q - q'\|$$

Then a quantized version of S with feedback can simulate S with feedback with arbitrarily small error.

The proof is given in Appendix 3.

5.6. Quantized Simulation of Coupled DESS with Arbitrarily Small Error

We now specialize the theorem to DESS by considering the system S to be an integrator. As before replacing the output function *l* by the derivative function, *f* in the feedback loop represents the standard form of a first order differential equation system. Figure 26 illustrates the quantized version of the coupled model.

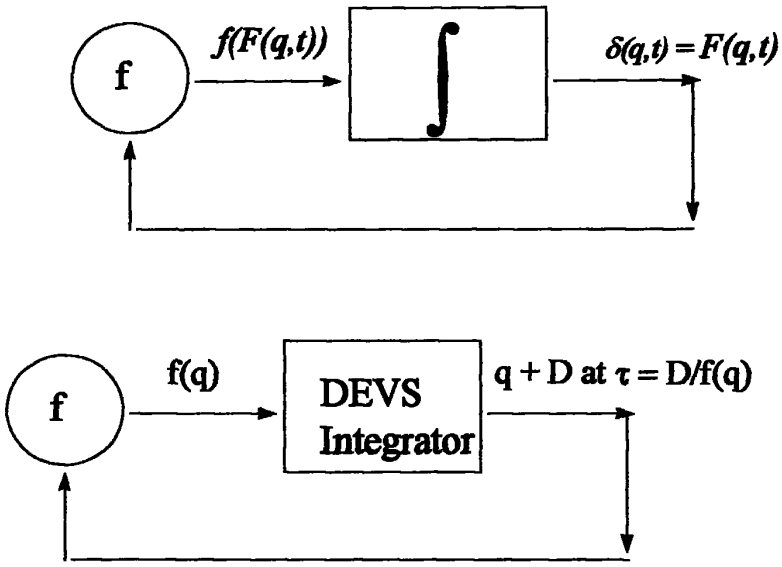

Figure 26 Closed Loop Quantized Simulation of DESS

Proposition: Closed loop simulation of a first order DESS by a Quantized Integrator with arbitrarily small error

A quantized version of the integrator can simulate a first order DESS with arbitrarily small error provided that the derivative is bounded above and below away from zero[8].

Proof: The proof follows that of Theorem 9 for discretized simulation of the integrator. In addition we have to establish the Boundary Crossing Times condition. We have for a boundary event time, *t:*

$D = || d_S(q, f(q)_{t>})) - q || = || \int_0^t f(q)dt || = t \, || f(q)||$ (*f(q)* is constant).

So $t = D / || f(q)||$ and $t \leq D/min \{ || f(q)|| \} > 0$ (by assumption)).

Thus in Theorem 9, we have $m = max \{ || f(q)| || \} = f_{max}$ and $M = min \{ || f(q)|| \} = f_{min}$. Thus, the error bound is (recall $k = 1$ and $k = L$)

$$e_T \leq D \, LT \, f^2_{max} / f_{min}{}^2 .$$

[8] In future work, we will seek to strengthen this theorem so not to required bounding away from zero.

5.7. DEVS Representation of Quantized Systems

Recall that the reason for introducing the quantized system concept is that it bridges the gap between DESS models and their representation in DEVS. Since as we now will show, DEVS can simulate quantized systems at both the I/O system and coupled system levels without error, the ability of quantized systems to represent DESS models directly carries over to DEVS.

Theorem 10: **Exact Simulation of Quantized Systems by DEVS at I/O System Level**

Every quantized system is simulateable (with zero-error) by a DEVS at the I/O system (global) level.

Proof: Let $S = (T, X, \Omega, Y, Q, \Delta, \Lambda)$ be the internal system with input and output quantizers $Q_{\pi 1}$ and $Q_{\pi 2}$, respectively.

Define a DEVS by $M = (X, Y, S, d_{ext}, d_{nt}, l, ta)$. Let $S = Q \times X$. The second component enables the DEVS to store its last input. Then,

- The *time advance* is the time to the next change in output produced by the output quantizer:

$$ta(q,x) = min \{ t \mid Q_{p2}(L(D(q, x_{<t>}))) \neq Q_{p2}(L(q))\}.$$

  i.e., the earliest elapsed time at which a change in output occurs.

- The output of the DEVS at the next internal event is the quantized output of S at that time:

$$l(q,x) = Q_{p2}(L(D(q, x_{<ta(q,x)>}))).$$

- Unless there is an external input, the DEVS will update its state to the state of the system at the next output:

$$d_{nt}(q,x) = D(q, x_{<ta(q,x)>}).$$

- If there is an external event after an elapsed time, e, in $<0,ta(a,x)>$, the DEVS will immediately update its state to

$$d_{ext}(q,x,e,x') = (D(q, x_{<e>}), x')$$

  i.e., it uses the current input to update the state of S and stores the new input from the input quantizer.

Appendix 3 shows how to construct the morphism to demonstrate showing exact simulation of the internal system by the DEVS system.

Combining DEVS exact simulation of quantized systems and the definitions of quantizable and sensitizable systems, we have:

Theorem 11: Every exactly quantizable system is simulateable by a DEVS at the (global) I/O system level with arbitrarily small error. Every sensitizable system relative to a given error tolerance is approximately simulateable by a DEVS with error no larger than the error tolerance.

5.8. DEVS Representation of Quantized Integrator

The DEVS realization of the quantized integrator has the simple definition:

$$M = (X, Y, S, d_{ext}, d_{int}, l, ta).$$

where $X = Y = R$ and $S = R \times R \times I$ and

- $d_{ext}((q,x,n),e,x') = (q+x*e, x', n)$
- $d_{int}(q, x, n) = (n + D*sign(x), x, n+ sign(x))$
- $d_{con}((q,x,n), x') = (n + D*sign(x), x', n+ sign(x))$
- $l(q,x) = n + D*sign(x)$
- $ta(q,x,n) = |((n+1)D - q)/x|$ *if* $x > 0$ and $(n+1)D - q > 0$
  $= |(q - nD)/x|$ *if* $x < 0$ and if $|q-nD| > 0$
  $= |D/x|$ *if* $x \neq 0$ and none of the above
  $= \infty$ otherwise *(i.e., x = 0)*

Here we keep track of the boundary below (or at) the current state q, i.e, the integer floor*(q/D).* Recall that even if we start on a boundary, the state may eventually be inside a block (hence not a multiple of *D*) as a result of an external transition. If, as in Figure 27 a), we are on a boundary, the time advance computation merely divides *D* by the current input *x* (which is the derivative, or slope, after all). If we reach the upper boundary (n+1) or lower boundary (n –1) we output and update the state accordingly. Note that so long as the input remains the same, the time to cross successive boundaries will be the same. Figure 27 b) shows that when a new input is received, we update the state using the old input and the elapsed time. From this new state, q, the new time to reach either the upper or lower boundary is computed.

The Quantized DEVS integrator greatly reduces the number of transitions and output messages needed to simulate an integrator. It also reduces the message size from double to integer. Actually since the only possible transitions are to the upper or lower boundaries, only one bit to represent the binary valued set +1,-1 need be sent.

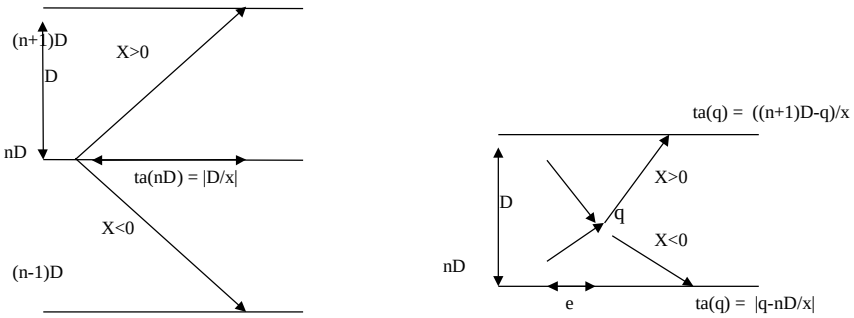

Figure 27 DEVS simulator of a Quantized Integrator

Thus there is a DEVS simulation of the quantized integrator with BCS inputs that can serve as an arbitrarily close representation of the integrator at the I/O system level. We still need to examine coupled networks of integrators to examine whether this error-free simulateability holds in the "closed loop" case.

5.9. DEVS Simulation of Coupled Quantized Systems

We will show that DEVS can strongly simulate coupled systems with quantized systems as components. Since, as we have just seen, quantized systems can approximate DESS models to an arbitrary degree of accuracy, we will conclude that DEVS can also strongly simulate DESS with arbitrarily small error.

Theorem 12: **Exact Simulation of Quantized Systems by DEVS at Coupled System Level**

Every coupling of quantized systems can be component-wise simulated (with zero error) by a DEVS coupled model.

Proof (sketch): Given a coupling of quantized systems, we have seen that each of the components can be simulated without error by a DEVS. Thus, we construct a coupled DEVS model in which each quantized component is mapped to its DEVS simulator. The coupling among DEVS simulators reflects that among the original components. The interface maps in the quantized network are retained without change for the DEVS coupled model. We employ the same morphism approach originally defined at the I/O system level now refined to apply at the coupled system level.

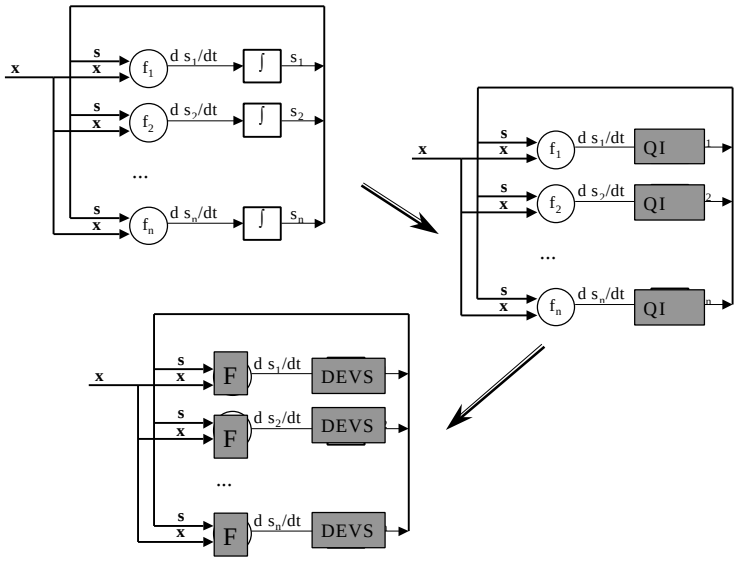

Figure 28 DEVS Simulation of DESS Coupled Models via Quantization

To complete the proof of the simulation we show that the global state transitions of the DEVS coupled model and those of the quantized system coupled model occur in synchronized fashion with corresponding results.

5.10. Quantization-based DEVS Simulation of DESS

Combining the ability of DEVS to exactly simulate quantized systems and the ability of quantized systems to approximate DESS models, yields the ability of DEVS to approximately coupled DESS models. This leads to the approach of Figure 28. Given a network of integrators and memoryless functions, in the first step, we construct a quantized network for simulating it by replacing the integrators by their quantized equivalents. In the second step, we replace each of the quantized integrators by their DEVS equivalents and replace the interface maps by DEVS-equivalent memoryless function representations. The resulting network isomorphically simulates the quantized coupled model without error, and consequently, the original DESS coupled model with the error introduced by quantization. This gives:

Theorem 13: Quantization-based DEVS Simulation of DESS

Any DESS can be simulated by a DEVS network at the coupled system level with an arbitrarily small error.

6. SIMULATION STUDY OF QUANTIZATION

Recall that the motivation for developing the quantization approach to system representation was to support the use of the DEVS Bus for both event-based and time-driven simulation. Using quantization, we are primarily looking for a way to reduce message traffic between components, however reduction in computation and error trade-off are also important considerations. In this section we discuss an early simulation study that provides some promising results and helps to focus on the issues more intensely.

Figure 29 reproduces the two approaches to quantization of DESS that we introduced. In the first (Figure 29 a), we employ conventional numerical methods to do the integration of the differential equations of each component in a coupled model (hence, a DTSS representation). We then attach a quantizer to the output, thus sending messages to other components only when large enough changes occur. We call this the *Quantized DTSS* approach. In the second approach (Figure 29 b), instead of using numerical analysis to do the integration, we directly quantize each integrator and represent it with a DEVS simulator. We call this the *Quantized DEVS* approach.

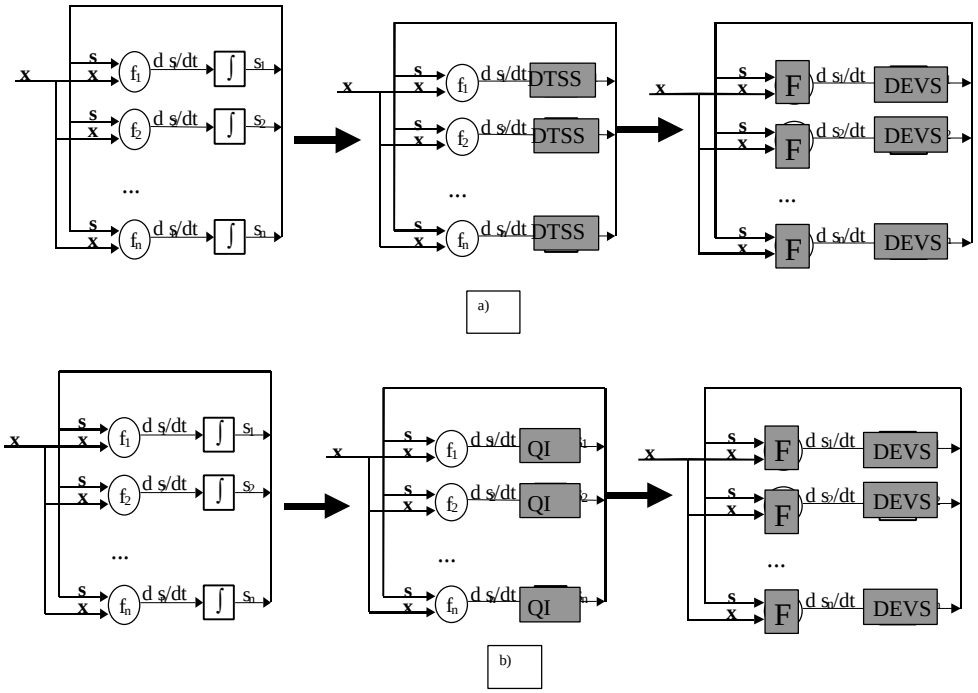

Figure 29 Quantized DTSS and Quantized DEVS Simulation Approaches

6.1. Some Promising Simulation Results

To compare the Quantized DTSS and Quantized DEVS approaches we chose an example with well-understood behavior that would enable us to compare both message reduction and error properties. Figure 30 shows the example used. It is DESS model of a Newtonian body orbiting around a fixed max (e.g. earth around the sun).

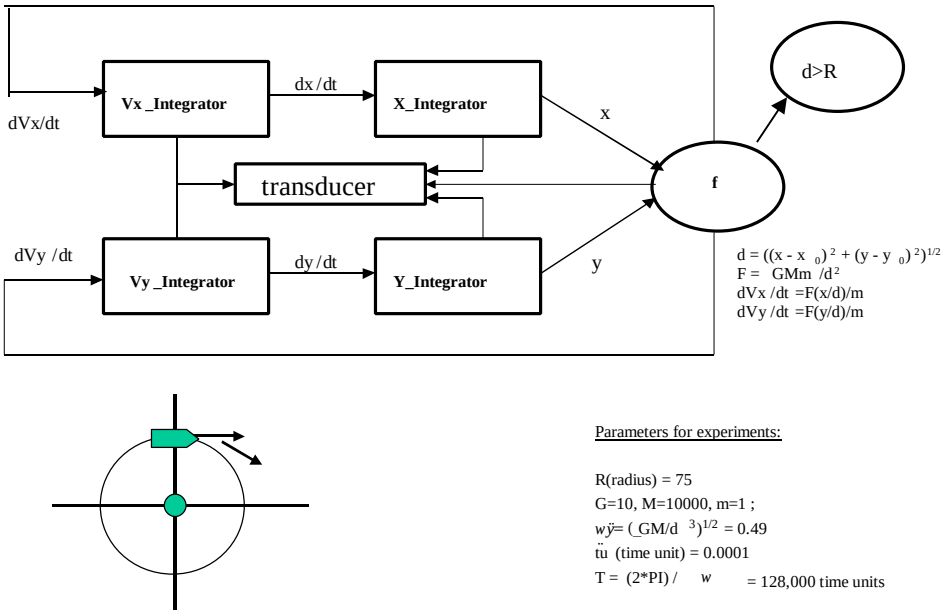

Figure 30 Test Example

Table 1 shows the results of experiments with the parameters shown in the figure. Since these tests were done on a sequential machine[9] the execution time includes internal transitions, external transitions, output computations and quantization checks. Although for the same quantum size, both approaches reduce the number of messages sent about equally, Quantized DEVS also significantly reduces the internal transitions and the computation time. (Internal transitions are required only at event times rather than at each time step as in DTSS.) Also a major advantage of DEVS quantization is that since events are predicted at exact boundary crossings, only information about which boundary has been crossed need actually be sent – which can be a lot fewer bits than required for the state at the boundary. One approach is to send the integer multiple of the quantum requiring the receiver to multiply this integer by the quantum size to recover the original real state value. More dramatic savings can be obtained by sending only the *change* in boundary level. Due to the incremental nature of DEVS quantization, this change can be only +1 or –1, requiring only a *one bit* transmission. Of course the receiver must keep track of the current boundary and know the proper quantization size to recover the actual state, as illustrated in Figure 24. Also important is that the Quantized DEVS approach is much more accurate than the Quantized DTSS for the same quantum size. In other words, for the same accuracy, the DEVS method requires many fewer bits sent and execution times. For example, compare the DTSS (D = .01) with the DEVS (D = .5) – the DTSS requires sending approx. 3000 times the number of bits (each message contains 64 bits to represent a real number with double precision) and approx. 40 times the execution time to achieve the same (low) accuracy.

[9] We used the DEVSJAVA environment using double precision for real number representation.

Performance Measure / Approach		Number of messages Sent	Number of Bits Transmitted	Number of internal transitions	Execution time (seconds)	Error (avg. deviation of radius from constant)
DTSS (h = .1 tu)		4,000,000 +2,00000	384,000,000	4,000,000	118,774	0.0099
DTSS (h = 1 tu)		400,000+200,000	38,400,000	400,000	12,776	0.015
Quantized DTSS (h = 1 tu)	(D= 0.5)	1,215 + 1,664	184,256	400,000	9,250	3.73
	(D= 0.1	6,256 + 8,330	933,504	400,000	9,243	1.47
	(D=0.01)	64,954+74,552	8,928,384	400,000	9,456	0.441
Quantized DEVS (with ±1)	(D= 0.5)	1,272 + 1,710	2,982	1,272	219	0.441
	(D= 0.1)	6,360 + 8,550	14,910	6,360	1,207	0.073
	(D=0.01)	63,602+ 85,510	149,112	63,602	13,130	0.007

Table 1 Comparison of Quantization Approaches (messages shown as the sum of those exiting the integrators and the memoryless function)

We also compared Quantized DEVS with "pure" DTSS, by which we mean using the DTSS without quantization. Effectively, this is Quantized DTSS with a quantum size matching the precision of the computer word size. The remarkable results is that the DEVS achieves a better accuracy than the pure DTSS (h = 1) with approx. the same execution time but with a reduction of approx. 250 times the number of bits transmitted. When we run the DTSS with h = .1, the results are even more dramatic. Now both have about the same accuracy but the DEVS is 10 times faster and has a 2500 times reduction in bits sent

6.2. Comparing Quantized DEVS with Pure DTSS Simulation of DESS

How can we explain the remarkable gains in performance by Quantized DEVS displayed in this example? Is this example just a fluke or is there something fundamentally different about the DEVS approach? The latter explanation is suggested by the fact that these results agree with earlier experiments showing a thousand-fold speedup in computation on massively parallel platforms using DEVS quantization [11]. This was a simulation study of surface water run-off using a high resolution spatial cellular model of a large scale watershed. The cells were modeled with ordinary differential equations and quantization was compared with a standard discrete time numerical method. To address this question we make the following

Conjecture: For any exactly quantizable DESS coupled model, a DEVS component-wise simulation is never less, and often much more efficient, than a counterpart variable or fixed step simulation with the same final accuracy.

We now present a heuristic argument why this conjecture should be true:

Consider a DEVS simulation of a DESS network of integrators as in Figure 28. By assumption for any given final error there is a quantum size D that enables the simulation to stay within that error. Let us choose the largest D that satisfies the tolerance requirement – so that crossing a boundary later than required by this D will result in an error that is above the tolerance limit.

Let's look at a DEVS simulation cycle as illustrated in Figure 31 a). Each DEVS integrator, i schedules itself for the next boundary crossing at $t_i = D/|x_i|$, where is the current input derivative to integrator, i. The time advance to the next event is then $s = \min\{t_i\}$ and the imminent integrators are those for which $t_i = s$. In Figure 31 a), the imminents are {1,4,6}. The coupling then determines which integrators receive outputs from these imminents. Imminents {1,6} send to {2,4,7} in the figure (shown as heavy vertical arrows). The imminents and the influencees {1,2,4,6,7} then execute their transition functions – {1,6} the internal, and {2,7}, the external function and {4} the confluent function (which in the case of an integrator is the same as the external transition function. This results in new time advances for imminents and their influences (shown as dotted arrows for the latter). None of the remaining integrators (e.g., {3,5,8}) execute computations and their next event times are unchanged. We are now ready for the next DEVS cycle.

To measure the complexity of such a cycle we see that we need to obtain measurements from the imminents and their influencees. But we don't necessarily have to identify them at each cycle to do this. This is because the complexity of a run is determined by accumulating the measured elements in the individual cycles. We instrument each component to count its internal transitions, external transitions and confluent transitions. This will catch all the executions without requiring us to identify the imminents and their influencees. We also have all outputs sent to a transducer thus capturing all the messages sent from the imminents to influencees, again without necessarily identifying who did what when.

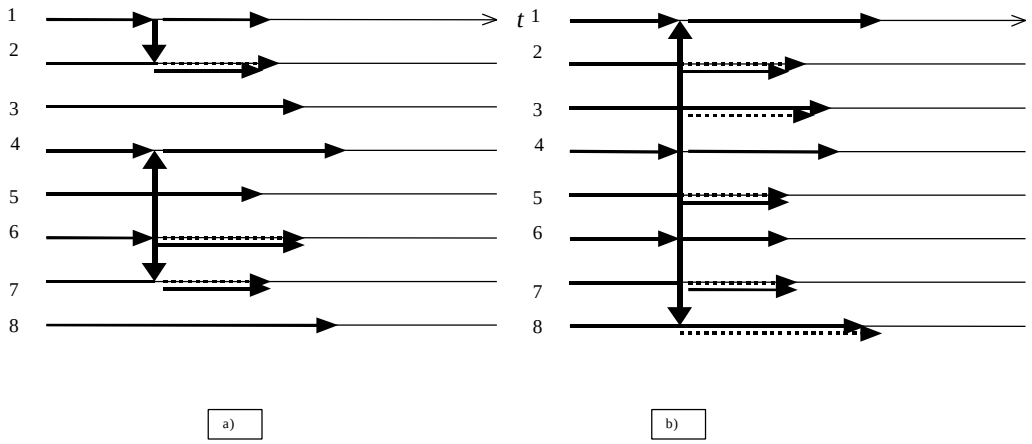

Figure 31 Comparing DEVS and DTSS simulation of DESS

Now consider Figure 31 b) where we illustrate a synchronized variable step simulation strategy. The global time advance is again $s = \min\{t_i\}$. Instead of activating only the imminents however, we activate all the integrators at this time. They exchange messages as prescribed by the coupling and execute their integration step (external transition function). Thus communication and computation occur at each event. We can call this a variable step integration method since at each cycle, s may be different. Note that some integrators such as {3.5,8} are now updated that were not it in the original DEVS simulation. Since we have chosen the quantum size for error tolerance, any increased accuracy that might result from such updating does not matter to us. (Had we wanted this improved accuracy we should have said so in the beginning – we can always go back and do so!)

In our final case, at each cycle we set $h = D/Max$ where *Max* is the largest (magnitude of) derivative of all the inputs to the integrators. Here the time advance is fixed for the whole run, so we have a DTSS simulation with time step *h*. A bigger time step than *D/ Max* runs the risk of increasing error since we would be too late for the boundary crossing of the integrator(s) that receives this largest derivative input (and D was chosen as liberally as possible within the error tolerance). Thus, *D/ Max* is the largest acceptable (constant) time step. Note that the updating in any cycle in which the maximum derivative is not present, is superfluous, since the earliest required updating occurs later.

Corollary of conjecture: To produce the same final error as a quantized DEVS simulation with quantum size, D, a DTSS simulation must have step size,

$$h \leq D/Max,$$

where *Max* is the largest (magnitude of) derivative.

In order of increasing complexity we have the quantized DEVS, the variable step, and the fixed step methods. All reduce the error below tolerance, but unless the maximum derivative is present at each cycle as input to each integrator, the DEVS will demonstrate many fewer transitions and message traffic. If we measure the *simultaneity* of the network by the fraction of *imminent* integrators at each cycle, we can see that DEVS simulation will be more efficient than both the variable and fixed step methods for small to moderately active states of coupled models. It is only for fully simultaneous states that persist for all of the run that a variable step method will equal the DEVS in performance. In addition, the maximum derivative has to be input to at least one integrator at each time cycle for the fixed time step method to match the DEVS performance.

Conjecture: Highly Simultaneous DESS Networks Are Rare

A number of arguments suggest that highly simultaneous networks are rare – in fact the chances of finding one get vanishingly small as the number of components increase to infinity. To start let's consider a stochastic characterization of the selection of derivatives. If we assume each integrator chooses its derivative independently, then he chance that all integrators receive the same input derivative, let alone the same maximum value, decreases exponentially with the number of components. Of course, the derivative process may be highly correlated, since the underlying model is deterministic. Nevertheless, the exponential dependence on number of integrators might well hold in any sequence of model with any sort of interesting behavior.

For example, in a second order linear oscillator, the integrators see sine wave inputs from each other that are 90 degrees out of phase. Indeed, when one integrator is getting the maximum value (amplitude times frequency) the other is getting the minimum, 0. Only every eighth cycle do the integrators see the same input magnitude and as we shall see the DEVS has approx. 30% fewer internal transitions. The 4^{th} order orbiting oscillator above shows a much greater reduction for DEVS integration in internal transitions over the DTSS simulation with the same accuracy.

6.3. Insight From 2nd Order Linear Oscillator

The linear oscillator shown in Figure 32 a) provides a revealing example of the efficiency advantages of DEVS. It is well known that the exact behavior is characterized by sinusoidal oscillations with the integrators 90 degrees out of phase with each other.

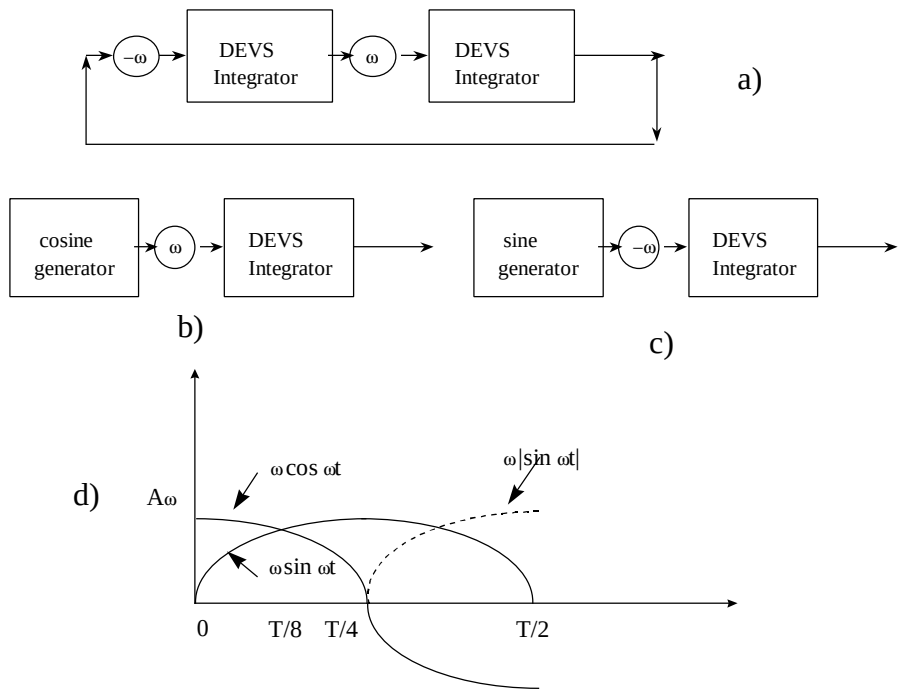

Figure 32 Second Order Linear Oscillator

Thus, while one is generating a sine wave the other is generating a cosine wave. Indeed, from the input perspective of each one, we can replace the other with the corresponding sinusoidal generator in open loop fashion as in Figure 32 b). The derivative input trajectories are shown in Figure 32 c). Taking into account that the step size is inversely related to the magnitude of the derivative, the same pattern of relative step sizes is repeated every eighth of a cycle. In every such period, one integrator always sees higher derivatives than the other, and they alternate among these fast and slow behaviors. Think of it this way: which ever is currently the slow integrator needs the fast integrator to speed it up; conversely, the fast integrator needs the other to slow it down.

Now in the DEVS simulation, in every eighth of cycle, the time advances of the fast integrator are larger than those of the slow integrator. It is only at the end of this period where they are equal. As shown in Table 2, we can estimate the average time step by following the steps of one integrator through a quarter cycle containing its slow and fast phases. We predict that DEVS cuts down on the number internal transitions and output messages by about 30% over the DTSS with the same error. In contrast, the variable step method is predicted to produce only a 10% reduction.

Method	**Means to Estimate**	**Internal transitions (output messages) relative to DTSS as D gets small**
fixed time step DTSS	D (set max. derivative =1)	1
variable time step	Average of minimum time step of both integrators over one eighth cycle	0.9
Quantized DEVS	Average of time step of one integrator over one quarter cycle	$2/\pi \cong 0.64$

Table 2 Predicted Comparison of DEVS and Variable Time Step Relative Baseline DTSS

Simulation of a test case showed that the predictions of relative performance of quantized DEVS versus pure DTSS were remarkably close. We compared the Quantized DEVS with quantum sizes, D shown in Table 3 against the DTSS with step sizes, h = D/max where the maximum derivative is the amplitude multiplied by the frequency with values shown in the table. Note that the ratio of messages sent (and internal transitions) is the same as predicted to 2 decimal places. The errors incurred in quantized DEVS are however, slightly higher than the corresponding DTSS values, although of the same order of magnitude. The data bits passed (using the DEVS binary output technique) are 100 times less for DEVS than for DTSS. The execution time for DEVS, although initially slightly larger, improves to about 40% that of DTSS as the quantum size decreases.

	Avg. Error	**Number of messages passed**	**Data Bit passed**	**Number of total internal transitions**	**Execution time (second)**
DTSS (h=66.7)	0.79	1,882 (0.64)	120,448	1,882	125
(h=13.3)	0.16	9,418 (0.64)	602,752	9,418	550
(h=1.33)	0.016	94,198 (0.64)	6,028,672	94,198	10,140
Quantized DEVS (with ±1) (D=0.5)	0.96	1,213	1,213	1,213	153
(D=0.1)	0.2	6,019	6,019	6,019	657
(D=0.01)	0.02	59,999	59,999	59,997	6,583

Table 3 Simulation Comparison of DEVS Quantization Approach with Pure DTSS : w = 0.0001, A = 75, run time = 62,800 (One cycle)

7. CONCLUSIONS AND FURTHER RESEARCH

The aim of the work was to extend the theoretical framework underlying the DEVS formalism to enable it to support the investigation of predictive filtering methods. We have succeeded in developing much of the theoretical foundation for the quantized system approach to discrete event representation of discrete time and continuous systems. We have also offered empirical evidence of the advantages of the proposed quantization approaches. We have shown those significant reductions of message bandwidth demands (number and size of messages) with controllable error and local computation costs are possible.

We now continue to refine our conjecture on why the quantized DEVS approach provides an efficient basis for predictive filtering techniques and indicate areas requiring further research along the lines established in this work.

7.1. Caveat: Effect of Integration Method

The results presented provide strong evidence for the claim that quantized DEVS can reduce the message traffic burden in distributed simulation visa vise quantized DTSS for the same accuracy. Moreover they support the stronger conjecture that quantized DEVS does better than even pure DTSS in both message passing and total execution time, even for small systems such as the 2^{nd} order oscillator. However, a caveat has to be stated here - *the effect of integration method has not been considered*. The results of the orbiting body in Table 1 were obtaining using a trapezoidal method for both DEVS and DTSS integration. The results presented in Table 3 were obtained using the simple Euler method for both. There are many varieties of integration methods for differential equation systems with arrays of parameter settings and accuracies. Thus we have to cautious with respect to the stronger conjecture concerning DEVS as a better integration approach than conventional integration methods in conventional uniprocessor simulation. It may well be that conventional methods are preferred for small differential equation systems on isolated sequential processors or even multiprocessors with shared memory. However, as the size of the system increases and also the number of processors with message passing required for interprocessor communication, it appears that DEVS quantization has distinct message reduction advantages for the same accuracy.

7.2. Conjectures on Quantized DEVS vs DTSS

The developments so far suggest that we can refine our conjecture into a restricted and strong forms:

Conjecture (restricted): The message passing efficiency of a quantized DEVS simulation of differential equation systems is never less than that of counterpart quantized DTSS to achieve the same accuracy.

Conjecture (strong): The efficiency (both message passing and execution time) of a quantized DEVS simulation of differential equation systems is never less than that of counterpart conventional integration methods to achieve the same accuracy.

The efficiency of quantized DEVS relative to the alternative approaches increases with the

- *order of the differential equation system*, i.e., with increasing numbers of integrators the frequency of occurrence of simultaneity is diminished.
- *temporal heterogeneity of the network*, i.e., the dispersion in the derivative values (interestingly, a class of such systems are called stiff systems in the literature and are hard to solve with conventional methods – are they easier to solve with DEVS?)
- *connectivity of the network*, i.e., with the fraction of other integrators influenced by an imminent integrator. In the extreme of a fully disconnected network (a not very interesting case!), the only cost (internal transitions) of conventional approaches is that due to less than full simultaneity. As the

connectivity of the network, increases there is a greater penalty for the communication required for unneeded state updating.

Further research is needed to establish the correctness of the conjectures. Especially, as indicated, the effect of integration method on the comparison must be investigated for the strong conjecture.

7.3. Further Research

Other theory developments emerging from this work that should be investigated are:

- applying quantization-based selective updating to slow-fast system decomposition [12-14]. Use of different rates of updating based on dynamics has been found to be important for real time simulation. Use of differential quantum sizes should show similar advantages.
- formulating the dead reckoning scheme employed in distributed interactive simulation within the DEVS quantization framework developed here. This will make it possible to compare the current approach to predictive filtering [2] with an approach exploiting perceptual sensitivity, captured by appropriate quantum sizes, to reduce state updates.
- incorporating delay (latency) explicitly in the framework. This will allow improved representation of the latency effects observed in real time DIS.
- improving estimation of quantization errors. The errors introduced by discrete time sampling of continuous signals are well known [15]. Quantization errors are less easily characterized since unlike sampling, quantization is a non-linear operation with respect to the frequency domain.

8. REFERENCES

1. Defense, D.o., *High Level Architecture Interface Specification, Version 1.0*, . 1996, Defense Modeling and Simulation Organization,available via http://msis.dmso.mil.
2. Bassiouni, M.A., *et al., Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation.* ACM Trans. on Model. and Comp. Sim. (TOMACS), 1997. **7**(3): p. 293-331.
3. Goel, S.G. and K.D. Morris. *Dead reckoning for aircraft in DIS*. in *1992 AIAA Flight Simulation Technologies Conference*. 1992. South Carolina.
4. Lin, K.C. and D.E. Schab. *The Performance Assessment of Dead Reckoning Algorithms in DIS*. in *SCSC*. 1994.
5. Zeigler, B.P., *Theory of Modelling and Simulation.* 1976, New York: John Wiley (Under Revision for 2nd Edition 1998).
6. Zeigler, B.P., T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation.* 2 ed. 1998, New York, NY: Academic Press.
7. Perko, L.M., *Differential Equations and Dynamical Systems.* 1990: Springer Verlag.
8. Zeigler, B.P., *et al., DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems*, in *Hybrid II, Lecture Notes in CS*, P. Antsaklis and A. Nerode, Editors. 1996, Springer-Verlag: Berlin. p. 529-551.
9. Zeigler, B.P., D. Kim, and H. Praehofer. *DEVS Formalism as a Framework for Advanced Distributed Simulation.* in *First International Workshop on Distributed Interactive Simulation and Real Time Applications(in conjunction with MASCOTS'97 -- International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems).* 1997. Eilat, Israel: IEEE Press, San Diego, CA.
10. Zeigler, B.P., *et al., The DEVS Environment for High-Performance Modeling and Simulation.* IEEE C S & E, 1997. **4**(3): p. 61-71.
11. Zeigler, B.P. and D. Kim. *Orders of Magnitude Speedup with DEVS Representation and High-Performance Computation*. in *Enabling Technology for Simulation Science, SPIE AeoroSense 97*. 1997. Orlando, FL.
12. Howe, R.M., *Dynamic Accuracy of the State Transition Method in Simulating Linear Systems.* Trans. of SCS, 1988. **5**(1): p. 27-41.
13. Howe, R.M., *The Use of Mixed Integration Algorithms in State Space.* Trans. of SCS, 1990. **7**(1): p. 45-66.
14. Laffitte, J. and R.M. Howe, *Interfacing Fast and Slow Subsystems in the Real-Time Simulation of Dynamic Systems.* Trans. of SCS, 1997. **14**(3): p. 115-126.
15. Soliman, S.S. and M.D. Srinath, *Continuous and Discrete Signals and Systems.* 1997: Prentice Hall.

9. Appendices:

9.1. Appendix 1: Closed Loop DTSS Simulation

Let $d(q,t)$ be the resultant closed loop transition function and let b_q be the output trajectory when starting in state q. Then $b_q(t) = l(d(q,t))$. The feedback of output to input then is expressed by: $d(q,t) = d_S(q, b_{q,t>})$.

Exercise: Using the above relations, show that the possible feedback input segments are segments, b_q that satisfy the fixed point equation:
$b_q(t) = l(d_S(q, b_{q\ t>}))$

In closed loop, we use the DTSS component transition function, for state q with input x, where $x = l(q)$ due to the feedback. Thus the transition function of the DTSS simulation becomes: $d_{DTSS}(q,h) = d_S(q, l(q)_{h>})$

We will show that the identity mapping is an approximate isomorphism between R and QR

The state error at each time step is expressed as:

$|| d(q, h) - d_{DTSS}(q, h)||$
$= || d_S(q, b_{q, h>}) - d_S(q, l(q)_{h>})||$
$\leq k || b_{q, h>} - l(q)_{h>} ||$ (Lipschitz condition on input)
$= k\ h\ max_{te<0,h>} || b_{q, h>}(t) - l(q)_{h>}(t) ||$ (definition of trajectory norm
$\leq k\ h\ max_{te<0,h>} || l(d_S (q, t)) - l(q)||$ (definition of b_q)
$\leq k\ k\ h\ max_h || d_S (q, t)) - q||$ (Norm preservation of l)
$\leq k\ k\ h\ m\ h$ (condition 3)

Thus the upper bound on the introduced error at each time step is:

$$e_{max} = kM\ h^2 \quad (\text{where } M = m\ k)$$

The amplification of the state error is obtained as the difference in states at the next time step:

$|| d_{DTSS}(q, h)- d_{DTSS}(q', h)||$
$= || d_S(q, l(q)_{h>})) - d_S(q', l(q')_{h>}) ||$
$= || d_S(q, l(q)_{h>})- d_S(q, l(q')_{h>})$
$+ d_S(q, l(q')_{h>}) - d_S(q', l(q')_{h>})||$
(adding and subtracting $d_S(q, l(q')_{h>})$)
$\leq || d_S(q, l(q)_{h>})- d_S(q, l(q')_{h>})|| + || d_S(q, l(q')_{h>}) - d_S(q', l(q')_{h>})||$
$\leq k\ h || l (q) - l(q') || + (1+a\ h) ||q - q' ||$
(Lipschitz condition on input and open loop amplification)
$\leq k\ h\ k||q - q' || + (1+a\ h) ||q - q' ||$(output norm preservation)
$\leq ((1+ h\ k\ k + ah) ||q - q' ||$
$\leq (1+K\ h) ||q - q' ||$ (where $K = hk\ k + a$)

Since we can make $K\ h$ as small as desired, we use the error propagation approach of Section 1.6 (for the case where $a = 1 + b$). The error after N steps, for large enough N, is

$$e_N \leq N\ e_{max}$$

For a fixed interval T, the number of steps is $T/\ h$ and the error bound becomes:

$$e_T \leq kMT\ h$$

which goes to zero as h goes to zero. This has the same form as the open loop case. However, here the input growth parameter M is derived as a product of the output and change parameters, $M = mk$.

9.2. Appendix 2: Uniformly Segmentable Input Sets

We now characterize uniformly segmentable input sets. We say that a *BCS* input segment, ω is *finitely oscillatory* if its derivative function, ω' has at most a finite set of points at which it vanishes ($\frac{d}{dt}w(t) = 0$)

The set of points mentioned just now are maxima, minima, or inflection points. The restriction to a finite set of such points does not permit an oscillation with infinite frequency as input – which is reasonable, there are too many events to simulate with a DEVS! The first case to examine is where this set is empty. In this case, we have the

Lemma: A BCS segment ω, for which the derivative function is bounded from below, is uniformly segmentable.

Proof: Bounding from below implies there no points of vanishing derivative (no maxima, minima or inflection points) and we have $Max > Min > 0$ such that:

$$Min < \| \frac{d}{dt} w(t) \| < Max$$

for all t in the domain of ω.

Consider any quantization. It is easy to show that it has a finite set of boundary crossings. Consider any two successive boundary crossings. By time invariance, we can take these instants to be 0 and $\tau>0$. Consider the segment it represents, $\omega_{\tau>}$.

Now since $\omega_{\tau>}(0)$ and $\omega_{\tau>}(\tau)$ are on boundaries, there are two cases: 1) $\omega_{\tau>}(0) = \omega_{\tau>}(\tau)$ and 2) $|\omega_{\tau>}(0) - \omega_{\tau>}(\tau)| = D$. The mean value theorem of calculus which states that

$\omega_{\tau>}(\tau) - \omega_{\tau>}(0) = \tau * \frac{d}{dt} w(t)$ for some t in the interval $(0, \tau)$.

But for case 1), we have $\frac{d}{dt} w(t) = 0$, contradicting the assumption that the derivative does not vanish.

Thus we have only case 2) and for it the length of the segment, τ, can be determined from

$$\tau * \frac{d}{dt} w(t) = D.$$

Using the upper and lower bounds on the derivative of ω, we have that:

$$D/Max = \tau_{min} \leq t \leq \tau_{max} = D/Min$$

where Min and Max as given above.

Remark: The *Min* and *Max* terms in the error expression of Theorem 1 are now interpretable as the lower and upper bounds on the magnitude of the derivative of the input segment. The ratio *Max/Min* in the error expression (recall $k = 1$):

$$e_T \leq D\,T\,Max/Min.$$

is a measure of the variability of the input. The greater this variability, the smaller the quantum size needs to be keep the error below a desired tolerance.

Now we are ready to extend the lemma to a BCS set with a non-empty set of points of vanishing derivative.

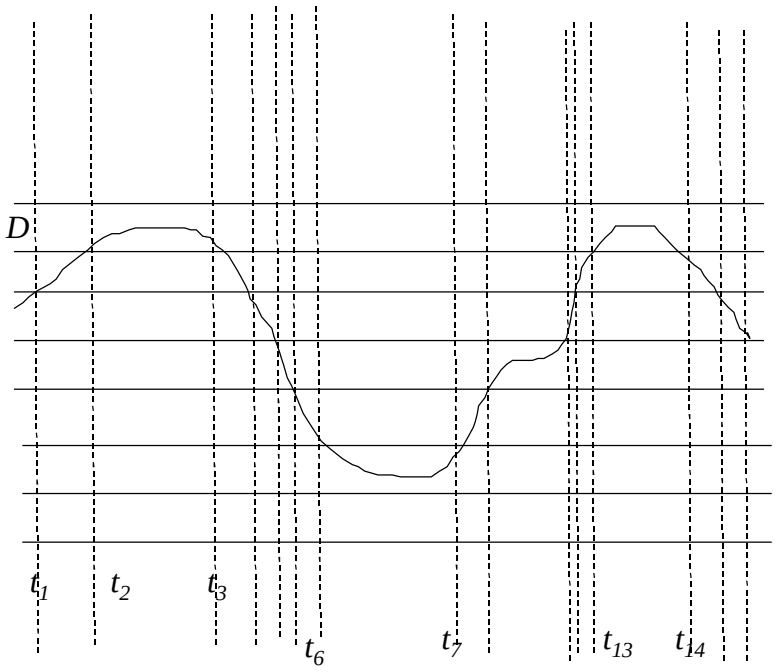

Figure 33 A Bounded Continuous Smooth input segment with 3 points of vanishing derivatives

Theorem 14: The integrator with a BCS input segment set is an exactly quantizable system.

Proof(sketch): As illustrated in Figure 33, given any segment, choose a small enough quantum D which isolates each point of vanishing derivative in the interior of its own generator segment By continuing to reduce D, we can retain the isolation of these points as well as reduce the error introduced in the finite set of isolating segments as small as we like. Now use this quantization as the starting D_0 in Theorem 3. The set of intervals that remain after removing the isolating segments all satisfy the conditions of Theorem 6. Thus we can reduce the error as small as we like in these segments as well.

9.3. Appendix 3: Exact Simulation by DEVS

Figure 34 shows how we can construct the morphism to demonstrate showing exact simulation of the internal system by the DEVS system. Recall that input quantizer produced a piecewise constant segment. We introduce a mapping, *g* that translates piecewise constant segments into corresponding DEVS segments. Its definition is straightforward:

$$g(\hat{x}_{t1>}\ \hat{x}_{t2}\ \ldots\ \hat{x}_{tn>}) = \hat{x}_{t1>}\ \hat{x}_{t2}\ \ldots\ \hat{x}_{tn>}$$

where $\hat{x}_t$ on the left is a constant segment and on the right is a DEVS generator segment starting with the constant as its event.

The morphism requires that 1) when the system is started in state *q* and given an input segment *w*, its ending state corresponds to that of the DEVS system started in state (*q, w(0),0*) with the input *g* ($Q_{\pi 1}(\omega)$), and 2) that the output segments produced by each system agree modulo the *g* mapping, i.e., they are the same except that one is piecewise constant and the other is discrete event in nature.

The proof proceeds by using the iterative system approach[5]. We need only examine the generator segments and show that the homomorphism requirements hold for them.

Define the map $h(q,x,e) = D(q, x_{e>})$ which places the DEVS system states in correspondence with those of the internal system. For any generator segment, we show that 1) this mapping is preserved under corresponding input segments applied to the two systems, and 2) starting in corresponding states, the systems produce the same output generator segments, modulo the *g* map.

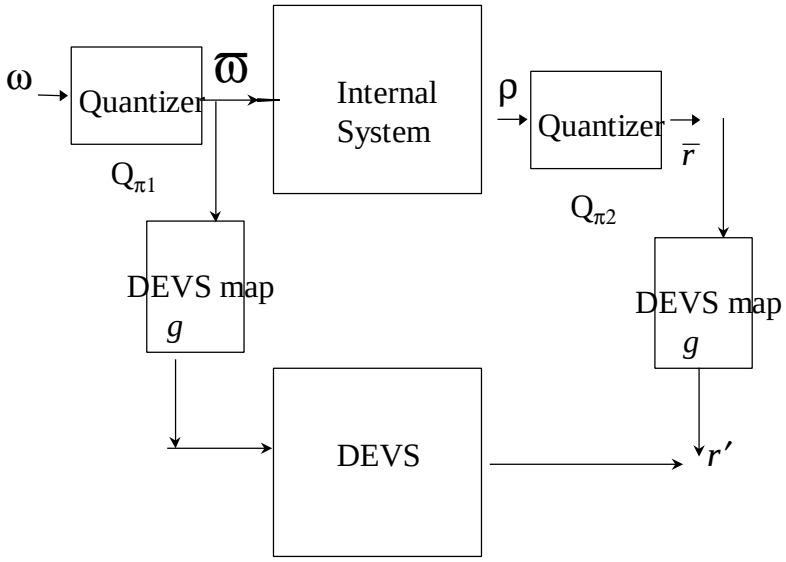

Figure 34 DEVS Simulation of Quantized System

9.4. Appendix 4: Closed Loop Quantized Simulation

We will show that the identity mapping is an approximate isomorphism between R and QR

Using property 1, the spacing between boundary crossings of the resultant quantized system can be bounded as follows:

Let q be on a boundary. Then we have the subsequent trajectory is described by:

$$d_{QR}(q,t) = d_S(q,\ l(q)_{t>})$$

A next boundary crossing occurs in time t where

$$d_{QR}(q,t) = q \pm D$$

i.e., $d_S(q,\ l(q)_{t>}) = q \pm D.$

or $D = \| d_S(q,\ l(q)_{t>}) - q \|.$

From property 1 a) (small excursion behavior), we have $D \le m\ t$, i.e., $t_{\min} = D/m$

Also, directly from property 1 b), we have $\tau_{\max} = D/M.$

The state error at each boundary crossing (output event, hence also input) is expressed as:

$\| d_R(q,\ t) - d_{QR}(q,t) \|$
$= \| d_S(q,\ b_{R,q\ t>}) - d_S(q,\ l(q)_{t>}) \|$
$\le k\ t\ \| b_{R,q,t>}(t) - l(q)_{t>}(t)) \|$ (Lipschitz condition on input)
$\le k\ t\ \| l(\ d_S(q,\ b_{R,q\ t>})) - l(q)) \|$ (definition of trajectory endpoints)
$\le k\,k\ t\ \| d_S(q,\ b_{R,q\ t>})) - q) \|$ (norm preservation of l)
$\le k\,k\ t\ \ mt$ (small excursion behavior)

Thus the upper bound on the introduced error over an interval T is:

$$e_{max} = \boldsymbol{M}\ \ t^2_{\ max} \quad \text{(where } \boldsymbol{M} = mk\,k\text{)}$$

The amplification of the state error at the quantized system is determined by:

$\| d_{QR}(q,t) - d_{QR}(q',\ t) \|$
$= \| d_S(q,\ l(q)_{t>}) - d_S(q',\ l(q')_{t>}) \|$
$= \| d_S(q,\ l(q)_{t>}) - d_S(q,\ l(q')_{t>})$
$\quad + d_S(q,\ l(q')_{t>}) - d_S(q',\ l(q')_{t>}) \|$
(adding and subtracting $d_S(q,\ l(q')_{t>})$)
$\le k\ t \| l(q) - l(q') \| + (1+a\tau)\ \|q - q'\|$
(Lipschitz conditions on input and state)
$\le k\ tk \|q - q'\| + (1+a\tau)\ \|q - q'\|$(output norm preservation)
$\le (1+t\,k\,k+a\ t)\ \|q - q'\|$
$\le (1+K\tau)\ \ \|q - q'\|$ (where $K = k\,k+a$)

Using the error propagation approach of Section 1.6 (for the case where $a = 1 + b$), the error after n steps, for large enough n is

$$e_n \leq n \mathbf{M} \ t^2_{max}$$

For a fixed interval T, the number of steps is less than T/τ_{min} and the error bound becomes:

$$e_T \leq (T/\tau_{min}) \mathbf{M} \ t^2_{max}$$

$$= \mathbf{M} \ T \ \tau_{max}^2 \ / \tau_{min}$$

Since $\tau_{max} = D/M$ and $\tau_{min} = D/m$, we have

$$e_T \leq D \ \underline{k} \ k \ T \ m^2 / M^2$$

Thus the error can be made as small as desired by making D as small necessary.