Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

APPLYING DEVS MODELING FOR DISCRETE EVENT MULTIPLE MODEL CONTROL OF A TIME VARYING PLANT

Alexander Scott Campbell
Gabriel Wainer

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, K1S 5B6, Canada

ABSTRACT

In recent years, we have developed a Modeling and Simulation-Driven Engineering methodology for engineering embedded Real-Time systems. This approach relies on the use of the DEVS formalism for developing components of real-time embedded systems using incremental development. Here, we show how to apply these techniques for an application in hybrid control. The model defines a discrete-event controller for a time varying plant based on multiple model control. Our discrete event approach permitted us to define such application, seamlessly integrating discrete event and continuous components. The approach allows secure, reliable testing, analysis of different levels of abstraction in the system, and model reuse. The common problem of "controller wind-up" or "parameter estimation bursting" can be avoided when performing this proposed form of discrete event adaptive control.

1 INTRODUCTION

Embedded real-time software construction has usually posed interesting challenges due to the complexity of the tasks executed. Most methods are either hard to scale up for large systems, or require a difficult testing effort with no guarantee for bug free software products. Although formal methods have advanced, their adoption by the engineering community is still under development, moreover because they are difficult to apply when the complexity of the system under development scales up. Instead, the use of M&S is a well-known approach by systems engineers, which makes system development tasks manageable. This is a useful approach, moreover considering that testing under actual operating conditions may be impractical and in some cases impossible.

In (Wainer et al. 2005; Glinsky and Wainer 2004; Glinsky and Wainer 2004b) we introduced Modeling and Simulation-Driven Engineering (MSDE), whose main objective is to explore the integration of M&S in all aspects of real-time embedded system engineering. MSDE proposes a discrete-event simulation architecture to be used as the final target architecture for products. Our approach for MSDE is based on the DEVS (Discrete EVents Systems specification) formalism (Zeigler et al. 2000). DEVS provides a formal foundation to M&S that combines the advantages of a simulation-based approach with the rigor of a formal methodology.

Most applications can be thought as a combination of discrete event and Continuous Variable Dynamic Systems, which are represented by continuous variables on a continuous time basis. Analysis of these complex systems has usually been tackled using different mathematical formalisms, including Differential Algebraic Equations (DAEs), Ordinary Differential Equations (ODEs), or Partial Differential Equations (PDEs). Most existing simulation tools implement numerical methods based on the discretization of time to find approximate solutions to these equations, which are based on discretization of time. In the last few years, different approaches developed tried to simulate continuous systems under the discrete event paradigm. This presents some advantages over discrete time simulation, including reduction of the number of calculations for a given accuracy (Zeigler 2005) and seamless integration of complex systems composed by both continuous time and discrete event paradigms. The idea of this method, called *Quantized Systems* theory (Q-DEVS), is to provide quantization of the state variables obtaining a discrete event approximation of the continuous system (Zeigler 1998). The state variables of the system are converted into a piecewise constant function via a quantization function. The *Quantized State System (QSS)* method (Kofman 2003) is an extension to Q-DEVS in which the trajectory of each state variable is converted into a piecewise constant function via a quantization function equipped with hysteresis.

Conventional adaptive control using a single identification model is efficient when the initial parameter estimation error is small, and plant parameters are slowly varying over time. The use of multiple models becomes appropri-

ate, when either of these conditions are not satisfied, such as in the case of a subsystem failure or a change in the operating environment. Typically, a finite number of models are evaluated by an index-of-performance, where, at any instant, the most suitable model's parameterized controller is applied to the plant. An arrangement using multiple fixed models is shown in Figure 1. This approach proves beneficial for maintaining control of a plant when there are parameter jumps (Narendra et al. 2003).

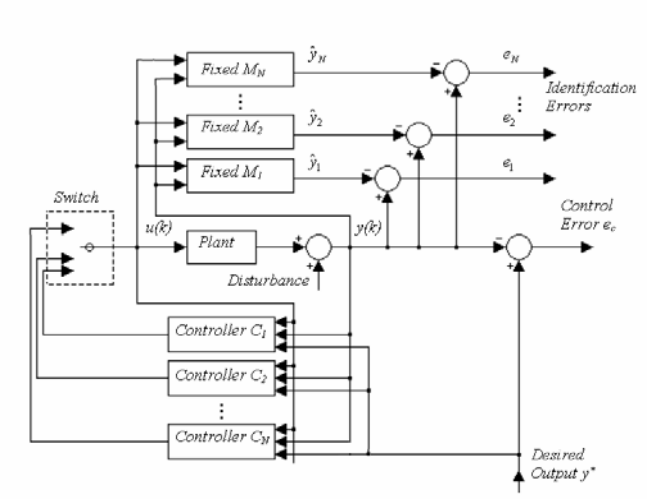

Figure 1: Typical Arrangement of Multiple Fixed Models for Control of an Unknown Plant (Narendra et al. 2003)

Multiple model control demands a union of high-level decision making with mathematically complex algorithms. Here, we present an implementation of such algorithms using a discrete-event mathematical approach that can be built in embedded control systems. This requires studying the application of discrete event modeling to hybrid/continuous systems (Kofman 2003b; Kofman 2003c).

The goal of the controller we present is to have the plant's output match the reference signal y_r, with zero control error. The reference signal is originally a real signal, which must be discretized using quantized DEVS with hysteresis. When performing mathematical calculations of the plant and controller states, interpolation of state, in-between event arrivals, was necessary. Here, we show how to apply these techniques for an application in hybrid control. The model defines a discrete-event controller for a time varying plant based on multiple model control. Our discrete event approach permitted us to define such application, seamlessly integrating discrete event and continuous components. The approach allows secure, reliable testing, analysis of different levels of abstraction in the system, and model reuse. Discrete event control using a single adaptive controller is all performed. With this, the common problem of "controller wind-up" or "parameter estimation bursting" can be avoided using the proposed form of discrete event adaptive control

The experience was carried out using CD++ (Wainer 2002), a software implementation of DEVS with extensions to support real-time model execution. We will explain how to use our approach to integrate the different plant models seamlessly. As showed in (Wainer et al. 2005), the models can be replaced incrementally with hardware surrogates at later stages of the process. The transition can be done in incremental steps, incorporating models in the target environment after thorough testing in the simulated platform. The use of DEVS improves reliability (in terms of logical correctness and timing), enables model reuse, and permits reducing development and testing times for the overall process.

2 DEVS AND CD++

DEVS (Zeigler et al. 2000) is a formal M&S framework based on systems theory. DEVS has well-defined concepts for coupling of components and hierarchical, modular model composition. DEVS defines a complex model as a composite of basic components (called **atomic**), which can be hierarchically integrated into **coupled** models. A DEVS atomic model is described as:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

Every state **S** is associated with a lifetime **ta**, which is defined by the time advance function. When an event receives an input event ***X***, the external transition function $\boldsymbol{\delta_{ext}}$ is triggered. This function uses the input event, the current state and the time elapsed since the last event in order to determine what the next model's state is. If no events occur before the time specified by the time advance function for that state, the model activates the output function $\boldsymbol{\lambda}$ (providing outputs ***Y***), and changes to a new state determined by the internal transition function $\boldsymbol{\delta_{int}}$.

A DEVS coupled model is defined as:

$$CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$$

Coupled models are defined as a set of basic components $\mathbf{M_i}$ ($i \in \mathbf{D}$) interconnected through their interfaces (***X, Y***). The translation function $\mathbf{Z_{ij}}$ converts the outputs of a model into inputs for others using I/O ports. To do so, an index of influencees is created for each model ($\mathbf{I_i}$). This index is used to connect outputs in model M_i are connected with inputs in the model M_j ($j \in I_i$). The formalism is closed under coupling, therefore, coupled and atomic models are semantically equivalent, which enables model reuse.

The execution of a DEVS model is defined by an abstract mechanism that is independent from the model itself. DEVS also permits defining independent experimental frames for the model, that is, a set of conditions under which the system is observed or experimented with. The

CD++ toolkit (Wainer 2002) implements DEVS theory. Atomic models can be defined using C++. Coupled models are defined using a built-in language that follows DEVS formal specifications.

3 THE PLANT AND CONTROL SYSTEM

We performed discrete event simulation of both an adaptive controller and a multiple model controller. In this section we develop the conceptual discrete event models for the more complex controller, the multiple model control. The plant is a 2nd order discrete time plant, defined using the difference equation

$$y_p(k) = p_1(s)y_p(k-1) + p_2(s)y_p(k-2) + p_3(s)u_c(k-1)$$

where the piecewise constant parameter vector is defined

$$\theta(s)^{\mathrm{T}} = [p_1(s) \quad p_2(s) \quad p_3(s)].$$

The *plant State* input, $s=\{1, 2, 3\}$, determines what set of parameters the time-varying plant should operate on ($\theta(1)^{\mathrm{T}}$, $\theta(2)^{\mathrm{T}}$, and $\theta(3)^{\mathrm{T}}$). This plant also requires, as inputs, its most recent outputs. For discrete event simulating, these most recent plant outputs must be found using interpolation (explained later). A new plant output is made when the trigger is enabled.

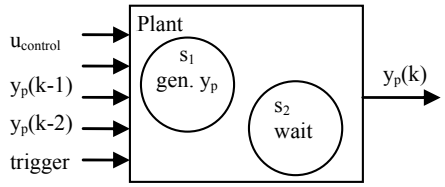

Figure 2: Conceptual Model of the Plant

The three Plant Identification Models must next be defined. Plant identifying model *i* creates an output as

$$y_i(k) = q_{i,1}y_p(k-1) + q_{i,2}y_p(k-2) + q_{i,3}u_c(k-1),$$

where the model's parameter vector is defined

$$\theta_i^{\mathrm{T}} = [q_{i,1} \quad p_{i,2} \quad p_{i,3})].$$

A second output is the modeling error, defined as

$$e_i(k) = (y_i(k) - y_p(k))^2.$$

This error is used by the controller to determine which plant identifying model's parameters are best for controlling the system (i.e., θ_1^{T}, θ_2^{T}, or θ_3^{T}).

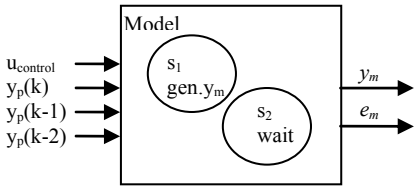

Figure 3: Conceptual Model of a Plant-Identifying Model

A conceptual model of the Unit Delay function must also be defined. This model performs the simple task of delaying a signal's propagation for one time unit. It is used to force the plant-identifying models to update *after* a plant output is generated. A model of the State Interpolation method must also be defined. This model performs the task receiving consecutive events, and interpolating points in between them. Given an event signal *x(k)* and past event information, the two points *x(k-1)* and *x(k-2)* are created as described in Figure 4.

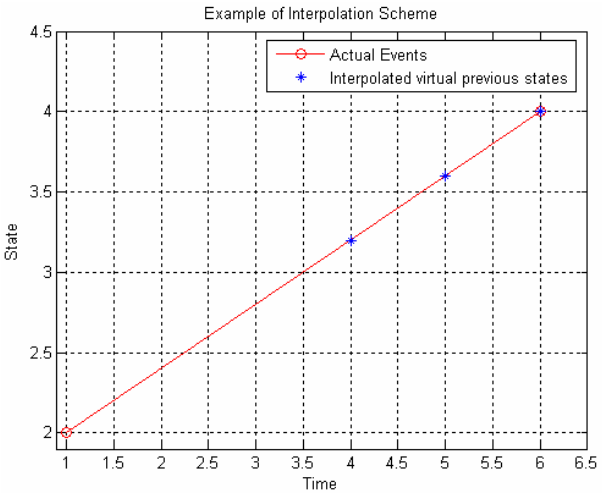

Figure 4: Interpolation Strategy

Finally, we need to build a conceptual model of the Controller. The goal of the controller is to have the plant's output match the reference signal y_r, with zero control error. This controller must analyze the available modeling errors (from the separately implemented plant identifying models) to decide which is the smallest and most suitable. The parameters associated with the best-fit model are used to generate a control signal for the system. These required parameters are already known within the controller and do not need to be sent to it. The certainty equivalence principle (Campbell 2005) is used with the chosen plant's parameters to calculate the control signal as follows

$$u_c(k) = \frac{y_r(k+1) - \theta_i^{\mathrm{T}}\tilde{\varphi}(k)}{q_i}, \text{ where}$$

$$\tilde{\varphi}(k) = [-y_p(k) \quad -y_p(k-1) \quad 0]$$

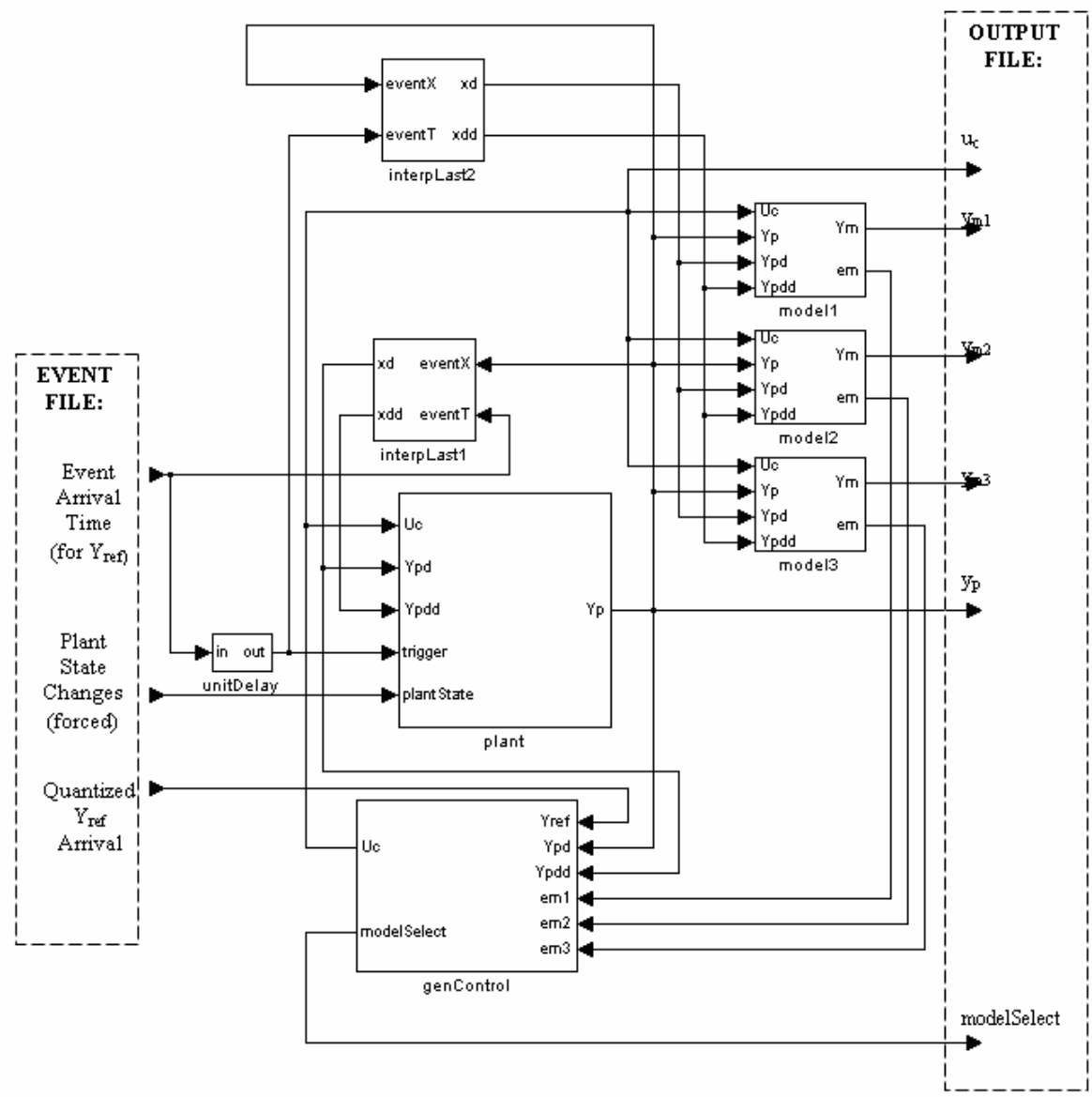

Figure 6: Conceptual Coupled Model of Multiple Model Controller

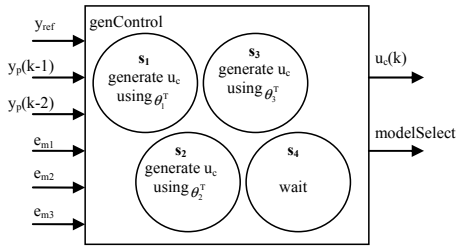

Figure 5: Conceptual Model of the Controller

Figure 5 shows the structure of a Coupled Model integrating the previously presented models as a Coupled Model (a detailed formal specification for each of the models can be found in the Appendix).

4 DISCRETE EVENT CONTROL STRATEGY PROTOTYPING

Adaptive control and multiple model control require mathematically complex calculations. Implementing such using DEVS and CD++ modeling language, requires additional design and testing. The initial problem posed is "how does one perform multiple model control, or adaptive control, when the discrete event system must model a real-time physical system?" Some of the tools we used:

- Quantized discretization
- Interpolation of states
- Triggering updates
- Unit delays, to propagate updates

As a starting point, the method of quantum discretization was implemented. The desired plant output, existing as reference signal, was declared in discrete time:

$$y_{ref}(k) = \sin(2\pi k/20) + \sin(2\pi k/10) + 2 .$$

Using this signal as an operand, quantized DEVS with hysteresis ($Q = 0.1$, $n = 2$) was applied. Given the signal being quantized is $\in (0,2)$, the normalized quantum size can be considered $\overline{Q} = 0.05$. The resulting quantized signal is a discrete time signal that contains discrete event changes. To remove the discrete time component, a list is made which contains the signal's event changes and associated event times. This list of events and event times was used for all tests performed. Figure 7 (a) shows the quantized signal and Figure 8 shows the reduction in state changes due to the discretization.

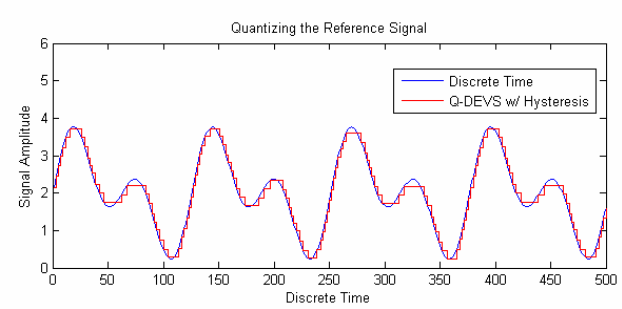

Figure 7: Discretization of Reference Signal (n=2,Q= 0.10)

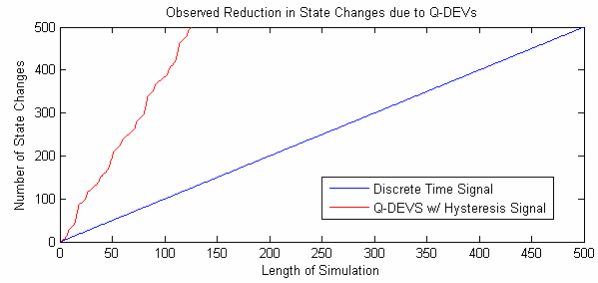

Figure 8: Reduction in State Changes Due to Discretization

First, discrete event, single model, adaptive control will be performed. The plant to be controlled has the time-invariant parameter values $p^{\mathrm{T}} = [0.6 \quad 0.2 \quad 0.1]$. A discrete event controller was implemented using RLS and certainty equivalence control. The adaptive model has the initial parameter estimates defined $\theta_{init}^{\mathrm{T}} = [1.1 \quad -0.3 \quad -0.4]$.

In observing the produced figures, it is clear that the control error remains roughly the same, despite the difference in quantum size for discretization of the reference signal.

These simulations allow for a discussion of parameter excitation. The RLS adaptive algorithm was able to converge faster when the quantum size was smaller. This is inherent, as increased excitation increases performance of adaptive algorithms. It is worth noting that this discrete event implementation of adaptive control of overcome the issue of controller windup. Controller windup, or the parameter burst phenomenon, occurs in discrete time when long periods pass without excitation while adaptation continues. Using discrete event notation, adaptation does not occur unless there are event changes.

Now discrete event simulation of the multiple model control will be performed. The conceptual model used is given in Figure 6. The plant has the possible plant states $p_1^{\mathrm{T}} = [0.6 \quad 0.2 \quad 2.0]$, $p_2^{\mathrm{T}} = [0.1 \quad 0.8 \quad 2.5]$, and $p_3^{\mathrm{T}} = [0.2 \quad 0.5 \quad 1.0]$. The models and controller have the available parameter estimates $\theta_1^{\mathrm{T}} = [0.6 \quad 0.2 \quad 2.0]$, $\theta_2^{\mathrm{T}} = [0.1 \quad 0.8 \quad 2.5]$, and $\theta_3^{\mathrm{T}} = [0.2 \quad 0.5 \quad 1.0]$.

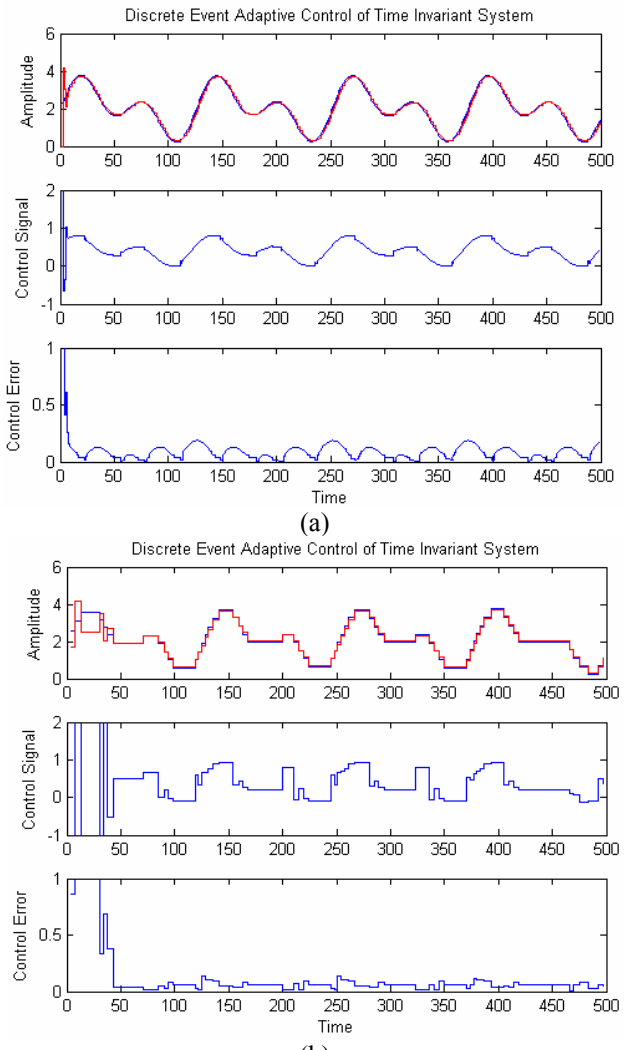

Figure 9: Adaptive Control (a) Using Q=0.02; (b) Q=0.2

An event file is used to input the reference signal events and their associated event time. The event file also forced the plant state changes (parameter jumps).

In the simulation given in Figure 10, the multiple model controller is handicapped and forced to *always* use the first plant identification model, θ_1^T.

Using the fixed parameter controller, stable control was achievable for plant states P_1 and P_3. At plant state P_3, the closed loop system becomes unstable, eventually yielding unbounded plant outputs.

In the simulation given in Figure 11, the multiple model controller is allowed to operate as designed, and switch among its plant identifying models. The simulation displays the advantages of multiple model control.

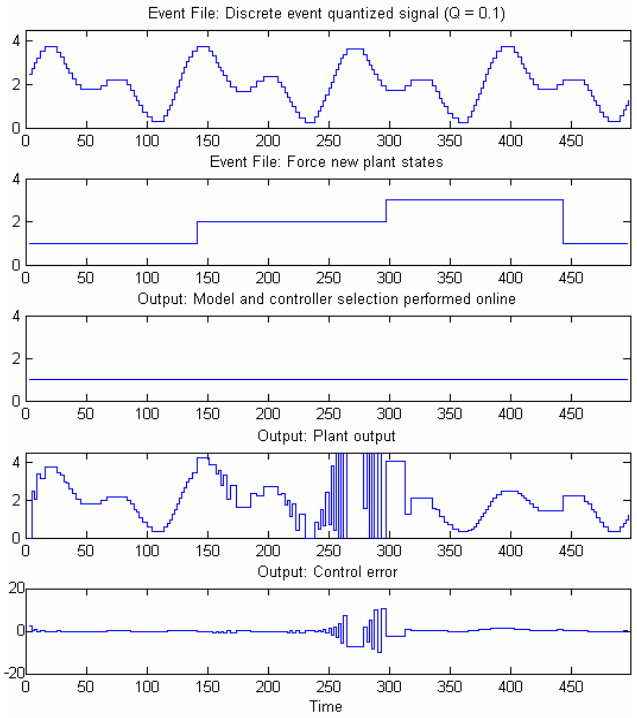

Figure 10: CD++ Simulation with Parameter Jumps, Using Only One Plant Identifying Model

Because a perfectly matching identification model was designed *a priori*, the controller was able to find it and use its parameters. For this deterministic scenario, control error existed only at the time period coinciding with each jump in plant parameters. During simulation initialization, the instantaneous error of each model was zero, requiring several reference signal arrival events and corresponding triggered plant outputs to identify which controller was most suitable. During operation, only at time 355 did a false model switch occur. The source of the false switch was due to two models having almost zero modeling error.

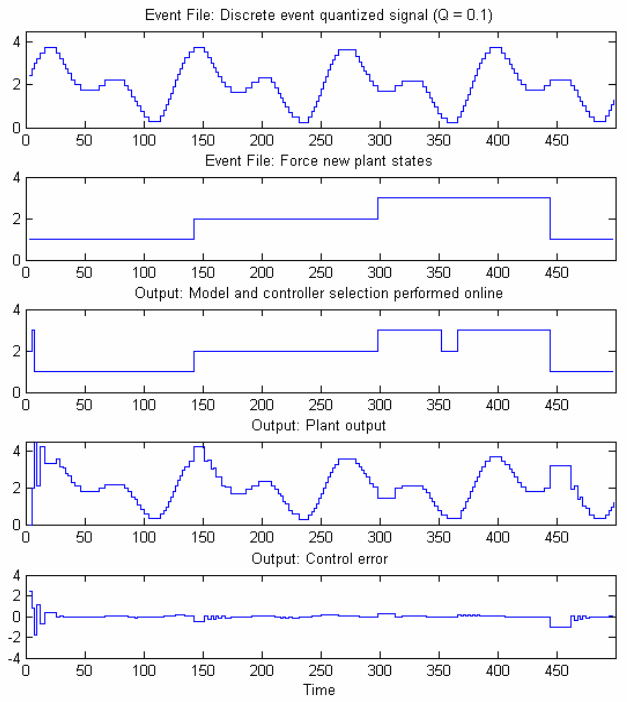

Figure 11: CD++ Simulation Containing Parameter Jumps, Using a Fixed Controller

5 CONCLUSION

The benefit of performing discrete event control using adaptive control and multiple model control was explained and demonstrated in this paper. The design problem was researched and discussed, before being implemented in two phases. In the first phase, we tested the proposed methods of discretization, interpolation, unit delays, and triggered subsystems. The front-end discretization provided a drastic reduction in data volume. Using true discrete event inputs and updates, adaptive control of a time-invariant plant was demonstrated. The discrete event implementation of adaptive control could overcome the issue of controller windup and parameter estimate burst. In the second phase, CD++ was used to fully implement a discrete event controller. Three plant identification models were implemented to identify jumps in plant parameters and select controllers accordingly. Simulations, using CD++, showed that the properties of continuous and discrete time multiple model controllers could be employed using discrete event systems with CD++.

APPENDIX: ATOMIC MODEL DEFINITIONS

Plant

X = {Uin Ypdin Ypddin Trigger plantState}
Y = {Ypout} **S** = {createOutput pState}

Vars = {p11 p12 p13 p21 p22 p23 p31 p32 p33 U Ypd Ypdd Yp scrap}
Internal Transition Function:
passivate
External Transition Function:

```
if (msg.port() == Uin) {
    U = msg.value();
    holdIn( active, Time( 0.1) ) ;    }
if (msg.port() == Ypdin) {
    Ypd = msg.value();
    holdIn( active, Time( 0.1) ) ;    }
if (msg.port() == Ypddin) {
    Ypdd = msg.value();
    holdIn( active, Time( 0.1) ) ;    }
if (msg.port() == Trigger) {
    if (createOutput == 0) {
    createOutput = 1;
    scrap = msg.value();
    holdIn( active, Time( 0.1) ) ;          }
}
if (msg.port() == plantState) {
    pState = msg.value();
    holdIn( active, Time( 0.1) ) ;        }
```

Output Function:

```
if (createOutput == 1) {
    createOutput = 0;
    if (pState==1) { Yp=p11*Ypd+p12*Ypdd+p13*U;}
    if (pState == 2) { Yp = p21*Ypd+p22*Ypdd+p23*U;}
    if (pState == 3) { Yp = p31*Ypd+p32*Ypdd+p33*U;}
    send output Yp to Port Ypout         }
```

Model1 (Model2 and Model3 are instances of Model1)

X = {Uin Ypin Ypdin Ypddin} **Y** = {Ymout Eout}
S = {haveU haveYp haveYpd haveYpdd}
Vars = {p1 p2 p3 U Ym Yp Ypd Ypdd E}
Internal Transition Function:
passivate
External Transition Function:

```
if (msg.port() == Uin && haveU == 0){
    haveU = 1;
    U = msg.value();
    holdIn( active, Time( 0.1) ) ;              }
if (msg.port() == Ypin && haveYp == 0) {
    haveYp = 1;
    Yp = msg.value();
    holdIn( active, Time( 0.1) ) ;    }
if (msg.port() == Ypdin && haveYpd == 0) {
    haveYpd = 1;
    Ypd = msg.value();
    holdIn( active, Time( 0.1) ) ;    }
if (msg.port() == Ypddin && haveYpdd == 0) {
    haveYpdd = 1;
    Ypdd =msg.value();
    holdIn( active, Time( 0.1) ) ;    }
```

Output Function:

```
if (haveU==1 && haveYp==1 && haveYpd==1 &&
    haveYpdd == 1) {
    haveU =haveYp = haveYpd = haveYpdd = 0;
    Ym = p1*Ypd+p2*Ypdd+p3*U;
        E = (Yp-Ym)*(Yp-Ym);
        send output Ym to Port Ymout
        send output E to Port Eout       }
```

UnitDelay

X = {theIn} **Y** = {theOut} **S** = {haveSignal}
Vars = {theSignal}
Internal Transition Function:
passivate
External Transition Function:

```
if (msg.port() == theIn && haveSignal == 0) {
    haveSignal = 1;
    theSignal = msg.value();
    holdIn( active, Time( 0.5) ) ;    }
```

Output Function:

```
if (haveSignal == 1) { //have the new event
    haveSignal = 0;
        send output theSignal to Port theOut }
```

InterpLast

X = {evXin xTin} **Y** = {xdout xddout}
S = {waitingforData haveNewEvent haveNewEventTime}
Vars = {evX oldEvX xT oldxT slopeX xtd xtdd}
Internal Transition Function:
passivate
External Transition Function:

```
if (msg.port() == evXin && haveNewEvent == 0) {
    haveNewEvent = 1;
    oldEvX = evX;
    evX = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
if (msg.port() == xTin && haveNewEvent == 1) {
    haveNewEventT = 1;
    oldxT = xT;
    xT = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
```

Output Function:

```
if (haveNewEvent == 1 && haveNewEventT == 1) {
     haveNewEvent = haveNewEventT = 0;
    slopeX = (evX-oldEvX)/(xT-oldxT);
    xtd = evX - slopeX;
    xtdd = evX - 2*slopeX;
        send output xtd to Port xdout
        send output xtdd to Port xddout
    }
}
```

GenControl

X = {Yrin Ypin Ypdin Em1in Em2in Em3in}
Y = {Uout, modelSelect} **S** = {haveYr haveYp haveYpd}
Vars = {q11 q12 q13 q21 q22 q23 q31 q32 q33 Yr Yp Ypd U em1 em2 em3 bestModel}
Internal Transition Function:

passivate

External Transition Function:

```
if (msg.port() == Yrin && haveYr == 0) {
    haveYr = 1;
    Yr = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
if (msg.port() == Ypin && haveYp == 0){
    haveYp = 1;
    Yp = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
if (msg.port() == Ypdin && haveYpd == 0) {
    haveYpd = 1;
    Ypd = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
if (msg.port() == Em1in) {
    em1 = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
if (msg.port() == Em2in) {
    em2 = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
if (msg.port() == Em3in) {
    em3 = msg.value();
    holdIn( active, Time( 0.001) ) ;      }
```

Output Function:

```
if (haveYr == 1 && haveYp == 1 && haveYpd==1) {
    haveYr = haveYp =haveYpd = 0;
    bestModel = 2; // initial guess
    U = (Yr-q21*Yp-q22*Ypd)/q23;
    // calc U as if Model 2 was best
    if (em1<em2 && em1<em3) {  // Model 1 is best
        bestModel = 1;
        U = (Yr-q11*Yp-q12*Ypd)/q13;
    if (em3<em2 && em3<em1) { // Model 3 is best
            bestModel = 3;
        U = (Yr-q31*Yp-q32*Ypd)/q33;     }
        send output U to Port Uout
        send output bestModel to Port modelSelect   }
```

REFERENCES

Campbell, A. 2005. Improvements to Stochastic Multiple Model Control: Hypothesis Test Switching and a Modified Model Arrangement. M.A.Sc. Thesis. Carleton University, Canada. August.

Glinsky, E.; and G. Wainer. 2004. Modeling and simulation of systems with hardware-in-the-loop. In *Proceedings of the Winter Simulation Conference*. Washington, DC. IEEE Press.

Glinsky, E.; and G. B. Wainer. 2004. Model-Based Development of Embedded Systems with RT-CD++. In *Proceedings of the WIP session*, *IEEE Real-Time and Embedded Technology and Applications Symposium*. Toronto, ON. Canada.

Kofman, E. 2003. Discrete Event Based Simulation and Control of Continuous Systems. Ph.D. Thesis. Universidad Nacional de Rosario, Argentina. August.

Kofman, E. 2003b. Quantized-State Control. A Method for Discrete Event Control of Continuous Systems, Latin American Applied Research (LAAR Journal), 33(4). pp 399-406.

Kofman, E. 2003c. Discrete Event Control of Time Varying Plants, Technical Report LSD0303, LSD, Universidad Nacional de Rosario.

Narendra, K. S., O. A. Driollet, M. Feiler, and K. George. 2003. Adaptive control using multiple models, switching and tuning, *Int. J. Adapt. Control Signal Process.*, vol. 17, pp. 87–102.

Wainer, G. 2002. CD++: a toolkit to define discrete-event models. *Software, Practice and Experience*. Wiley. Vol. 32, No. 3. pp. 1261-1306.

Wainer, G.; E. Glinsky, and P. MacSween. 2005. Model-Driven Architecture of Real-Time Systems. In *Model-driven Software Development - Volume II of Research and Practice in Software Engineering*. S. Beydeda and V. Gruhn eds., Springer-Verlag.

Zeigler, B. 2005. Continuity and Change (Activity) are Fundamentally Related in DEVS Simulation of Continuous Systems, LNCS, Vol. 3397, Springer-Verlag.

Zeigler, B. 1998. DEVS. Theory of Quantization. DARPA Contract N6133997K-007. University of Arizona.

Zeigler, B; T. Kim, and H. Praehofer. 2000. Theory of M&S. New York.

AUTHOR BIOGRAPHIES

ALEXANDER S. CAMPBELL received his B.Sc. degree with honours in Electrical Engineering from Queen's University, Kingston, Ontario, Canada in June 2003. He received his MASc. degree in Electrical Engineering from the department of Systems and Computer Engineering at Carleton University, Ottawa, Ontario in November 2005. He is currently working at March Networks Corporation, Ottawa, Ontario. His research interests include intelligent systems, discrete event and adaptive control, and digital signal processing. His email address is <alexc@sce.carleton.ca>.

GABRIEL WAINER received the M.Sc. (1993) and Ph.D. degrees (1998, with highest honors) of the Universidad de Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. In July 2000, he joined the Depart-

ment of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada), where he is now an Associate Professor. Previously, he was Assistant Professor at the Computer Sciences Department of the Universidad de Buenos Aires, and a visiting research scholar at the University of Arizona and LSIS, CNRS, France. He is author of a book on real-time systems and another on Discrete-Event simulation and over 110 research articles. He was PI of several research projects (NSERC, Precarn IRIS, IBM Scholars, Usenix, CFI, CONICET, ANPCYT). He is Associate Editor of the Transactions of the SCS, and the International Journal of Simulation and Process Modeling. He is a member of the Board of Directors of the SCS, a chairman of the DEVS standardization study group (SISO), and Associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences and chair of the Ottawa M&SNet. His current research interests is related with modelling methodologies and tools, parallel/distributed simulation and real-time systems. His e-mail and web addresses are <gwainer@sce.carleton.ca> and <www.sce.carleton.ca/faculty/wainer>.