A DEVS–based End-to-end Methodology for Hybrid Control of Embedded Networking Systems

Gabriel Wainer [*] **Rodrigo Castro** [**] **Ernesto Kofman** [**]

[*] *Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada. (e-mail: gwainer@sce.carleton.ca)*
[**] *CIFASIS–CONICET. Control Department, Universidad Nacional de Rosario, Argentina. (e-mail: {rodrigoc,kofman}@fceia.unr.edu.ar)*

Abstract: We present a formal Modeling and Simulation (M&S) methodology for hybrid control of networking systems. The method is used for analysis, design and implementation of Quality of Service (QoS) control systems in Network Processor (NP)-based applications. We apply continuous Control Systems Theory to enforce Admission Control strategies into discrete–event network traffic. This represents a hybrid system modeling problem, that has to be treated formally to guarantee the applicability of the continuous control theoretical results into discrete–event systems. We show that using DEVS (Discrete Event System Specification), in combination with Quantized State Systems (QSS) numerical methods for the approximation of continuous systems, offers numerous advantages: these frameworks provide the means to accurately analyze and design hybrid models for Admission Control and they can be seamlessly integrated into a unified formal framework. It also enables the transition between the DEVS–based simulation and the deployment of the obtained hybrid models into the target networking platform.

Keywords: DEVS, Hybrid Systems, Embedded Networking, Admission Control.

1. INTRODUCTION

Embedded networking systems include protocols and algorithms embedded in specific–purpose devices with limited resources. Control of Real-Time Computing Systems is a new area where traditional continuous control system techniques are applied to computing and/or networking systems, which are considered as the subjects to be controlled Hellerstein (2004). In these systems, the control objective is to keep performance metrics (delay, jitter, throughput, resource consumption, deadline miss ratio, etc.) within certain required bounds, usually specified by Quality of Service (QoS) requirements. The control actions trigger discrete events involving the allocation of available resources (pools, buffers, queues, slots, etc.) among different competing resource consumers (tasks, packets, jobs, requests, etc.) which are also discrete in nature.

Although these control systems are typically designed using ad–hoc techniques, classic control theory has been recognized as a way to exploit the theoretical and methodological background of the discipline (Arzen et al., 2006). Based on control theory, stability and transient response can be easily and systematically analyzed, and controllers can be designed following different optimal and/or robust criteria. In the last decade, control theory has been applied to many computing and networking systems, including active queue management schemes (Hollot et al., 2001a,b), network routers (Christin et al., 2002) and high-performance web servers (Robertsson et al., 2003), among many others.

One of the difficulties of applying control theory to these systems is related to the heterogeneous nature of the models used to analyze them. Real–time Computing and networking systems are usually described by discrete event dynamic systems (Cassandras and Lafortune, 2004), modeled with languages such as timed Petri Nets, timed Automata or timed Finite State Machines. Instead, classic control theory is based on continuous–time models (differential equations) or discrete–time models (difference equations). Although there are many techniques for dealing with control of discrete–event systems (Cassandras and Lafortune, 2004), their goal (i.e., safety, deadlocks) is not performance control (where stability, transient response and robustness play a key role).

In order to bridge the analytical gap between event based and time based control techniques, systems are usually approximated by either continuous or discrete–time models. Then controllers are designed and analyzed in these domains. However, this usually leads to over–simplifications and errors, and controllers must be checked (and eventually redesigned) with more accurate models (i.e., using discrete–event M&S). Although there are very advanced tools for this phase (e.g., ns-2 (Issariyakul and Hossain, 2008), OMNET++ (**?**)) their integration with the control theoretical models is very complex. This usually forces ad-hoc customizations, without a supporting formalism guaranteeing the correctness of these procedures.

As we can see, the resulting design process involves working with models of different nature, described with different languages and obtained by specialists of different

fields. This usually leads to switching between M&S tools and techniques, which yields many practical difficulties. Moreover, the implementation of the control algorithms in embedded platforms poses additional challenges as the algorithms usually must be translated and adapted to the target architecture.

In this work we propose an integrative methodology to deal with these issues, exploiting the features of Discrete Event System Specification (DEVS, Zeigler et al. (2000)), a M&S framework that can describe and simulate all kind of hybrid systems. We illustrate the methodology applied to the analysis, design and implementation of a traffic QoS control system in an Intel IXP2400 network processor (Intel, 2004). We show the definition of a case study on Admission Control strategies to network traffic based on non linear control theory. The Admission Control algorithm is designed as a sequence of M&S tasks. We first model the network with a continuous–time approximation (designing a continuous–time control law). We then find a discrete–time approximation of the control law (verifying the discrete–time controller applied to the original discrete–event system). We finally deploy the controller into an embedded network processor. This end–to–end methodology is completely based on a unified DEVS M&S framework.

2. BACKGROUND

DEVS (Zeigler et al., 2000) is a general formalism for describing systems that perform finite number of changes in finite intervals of time. A system modeled with DEVS is described as a composite of submodels, each of them being behavioral (atomic) or structural (coupled). Formally, a DEVS *atomic* model is defined by the following structure:

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

Each possible state s ($s \in S$) has an associated *lifetime* ta ($ta : S \rightarrow \Re_0^+$). If a state adopts a value s_1 at time t_1, after $ta(s_1)$ time units the system performs an *internal transition* to a new state s_2. The new state is calculated as $s_2 = \delta_{int}(s_1)$, where δ_{int} ($\delta_{int} : S \rightarrow S$) is called *internal transition function*. At the same time an *output event* y is produced with value $y_1 = \lambda(s_1)$. λ ($\lambda : S \rightarrow Y$) is called the *output function*. If an *input event* x arrives, the state changes instantaneously. The new state value depends on the input value, the previous state value, and the time elapsed e since the last transition. The new state is computed as $s_4 = \delta_{ext}(s_3, e, x_1)$ with s_3 the state at time t_3 and x_1 the input event arriving at time $t_3 + e$ (note that $e \leq ta(s_3)$). Function δ_{ext} ($\delta_{ext} : S \times \Re_0^+ \times X \rightarrow S$) is called the *external transition function*.

DEVS models can be coupled in a modular and hierarchical way. A DEVS *coupled model* is formally defined as:

$$CM = (X, Y, D, \{M_i\}, \{Z_{ij}\})$$

CM is a set of atomic components M_i ($i \in D$) interconnected through their interfaces (X, Y). The translation functions Z_{ij} convert the outputs of a model into inputs for others using I/O ports. The formailsm is closed under coupling (i.e., the coupling of DEVS models defines an equivalent atomic DEVS model).

Most networking systems present stochastic behavior. STDEVS (Castro et al., 2009) extended DEVS providing a formal specification of general stochastic discrete–event systems based on the theory of Probability Spaces. While DEVS and STDEVS can model and simulate any discrete system (discrete–time and discrete-event), continuous systems cannot be represented by the formalism due to the continuous evolution of its variables. However, although differential equations can model continuous systems, in order to simulate them we need to use classic numerical integration methods (i.e., Euler or Runge Kutta) to approximate the corresponding differential equations. These discrete–time models can be straightforwardly represented as DEVS models. Following this idea, we can simulate general hybrid systems under a unified formalism.

Classic numerical integration methods have problems with the discontinuities that are found in hybrid systems (caused by the discrete dynamics). These must be properly detected and treated; otherwise, the approximation of the continuous parts can lead to wrong results (Cellier and Kofman, 2006). Appropriate treatment and detection of discontinuities is in general difficult and computationally demanding, as it usually calls for iterations. DEVS, however, can represent discrete–time and discrete-event approximations of continuous systems. Quantized State Systems (QSS) methods provide a way to approximate a differential equation by a discrete-event model, preserving stability properties and guaranteeing error bounds. QSS methods can detect and handle discontinuities in an efficient and straightforward way, that does not require any iterations (Cellier and Kofman, 2006). Thus, they are particularly convenient to simulate hybrid systems.

Our goal is to apply these methods to improve the design of layered communication networks, where services provided by an immediate lower layer can be seen as delivered by a Service Control Node composed by *service logic* and *control logic*. In embedded networking systems, these algorithms run in special purpose processors with limited resources. The service logic is provided by *servers* (that process packets at a service rate), and incoming packets that cannot be processed are put into *queues*. The control logic tries to enforce QoS requirements under congestion (a network state where the demand of QoS levels cannot be satisfied for all traffic sources). This situation is handled by congestion control methods (Kwon and Kim, 2000). Well-known examples of congestion control techniques are the rate–based control for ATM networks and the window–based control for TCP/IP networks (which enforce QoS requirements for throughput and delay metrics).

Admission Control is a particular control logic based on the rejection of incoming packets before they are allowed to join the queues, while accepted packets are guaranteed to get serviced. Control Theory is an increasingly preferred choice to describe, analyze and design robust Admission Control algorithms. Non-linear control theory was used to design an admission controller for a Service Control Node in Kihl et al. (2003). While the authors resort to diverse M&S techniques and tools to deal with the hybrid nature of the system, we will show how to obtain the same results using an integrated methodology. This methodology has the potential to improve model reliability, sharing and

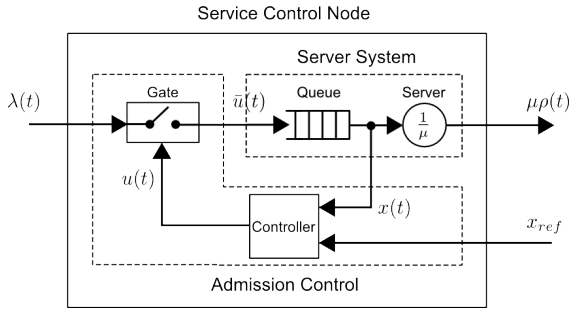

Figure 1. Service Control Node system under study.

reuse, while facilitating the transition into the target hardware.

3. CONTROL THEORY FOR ADMISSION CONTROL

We present a case study on the use of our methodology in a control theory based solution for designing an Admission Control mechanism. We will adopt the example shown in Figure 1 for a **Service Control Node**, originally presented in Kihl et al. (2003).

In our case, the block **Server System** is the plant to be controlled. It is composed by one **Queue** and one **Server**. In this example, the QoS requirement is to keep the queue length at a given reference desired value (i.e. the *set point* x_{ref} for the plant), this being the control objective. The input signal to the plant is the *admittance rate* $\bar{u}$ of packets into the queue, which is assumed to follow a stochastic Markovian process. The queue is assumed to have infinite capacity, and the instantaneous number of packets in the queue is the *output signal* x of the plant. The servicing times at the server follow some probability distribution with a mean value of $1/\mu$ seconds. The stochastic nature of both the queue admittance rate and the servicing times represent noise that will have to be compensated by the **Admission Control** block. The **Controller** block is designed to keep the number of enqueued packets x at the given desired value x_{ref}. The control action consists in opening and closing a gate at the entrance of the Server System to limit the incoming traffic (*arrival rate* λ). This is achieved by the **Gate** block which is the actuator of the controller. According to the *control signal* u, the gate selectively rejects packets from λ to obtain a controlled admittance rate $\bar{u}$.

In order to apply control theory methods to the Server System, a differential equation model can be derived to describe the system behavior. Following a *fluid flow approximation* method (Tipper and Sundareshan, 1990), the system can be modeled by the expression:

$$\frac{dx}{dt} = \bar{u}(t) - \mu G(x(t)) \tag{1}$$

where $\bar{u}(t)$ is the queue packet admittance rate, $1/\mu$ is the mean service time and $G(x)$ is a nonlinear function:

$$G(x(t)) = \frac{x(t) + 1 - \sqrt{x^2(t) + 2C^2x(t) + 1}}{1 - C^2} \tag{2}$$

C is the coefficient of variance of the service times. This approximation reproduces the steady state condition for the average number of packets in the system and the server utilization accurately. In terms of dynamic behavior, the approximation captures some stochastic properties of the system under non–stationary traffic conditions.

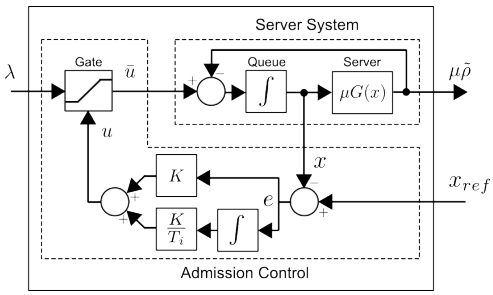

Figure 2. Continuous-time non-linear model.

The gate action can be described as follows:

$$\bar{u}(t) = \begin{cases} \lambda(t) & \text{if } u(t) > \lambda(t) \\ \max(0, u(t)) & \text{otherwise} \end{cases} \tag{3}$$

The authors proposed a PI controller for the Control Node (see Fig.2) with the following law:

$$u(t) = Ke(t) + \frac{K}{T_i}\int e(t)dt \tag{4}$$

where $e(t) = x_{ref} - x(t)$.

The system parameters for this example are set according:

$$\begin{array}{lll} \lambda = 20s^{-1} & \mu = 5s^{-1} & \mu_1 = 2s^{-1} \\ x_{ref} = 10 & C^2 = 3.7 & \mu_2 = 60s^{-1} \\ & & \alpha_1 = 0.38 \end{array} \tag{5}$$

The service processing times at the server are hyper–exponentially distributed, combining 2 exponential distributions of mean service rates μ_1 and μ_2 respectively, with a bias of $\alpha_1 = 38\%$ in favor of μ_1. This yields a mean service rate $\mu = 5\ sec^{-1}$ with a squared coefficient of variance $C^2 = 3.7$. The design is completed by finding appropriate values for K and T_i. In this case, following a linearization and pole placement procedure, they are $K = T_i = 2.4$. We will refer to the system parametrized as in Eq.(5) as the Controlled Packet Processing System (CPPS).

4. DEVS–BASED M&S METHODOLOGY FOR A CONTROLLED PACKET PROCESSING SYSTEM

This section shows the use of our DEVS–based M&S methodology to assist the processes of analysis, design, verification, implementation, and validation of the PI-controller for the CPPS specified in the previous section. The main results of the original work of Kihl et al. (2003) are reproduced as a step for validating our approach.

Analysis, design and verification are supported by *PowerDEVS* (Kofman et al., 2003), a DEVS–based tool for Hybrid System M&S. Implementation and deployment in the target platform are supported by the DEVS–based M&S software *ECD++* (Embedded CD++), an extension of the general purpose CD++ tool with capabilities for running in real-time, embedded on special-purpose processors (Wainer, 2009). Thanks to their compliance with the DEVS framework, both tools can simulate the hybrid models designed, and the transition between them is straightforward.

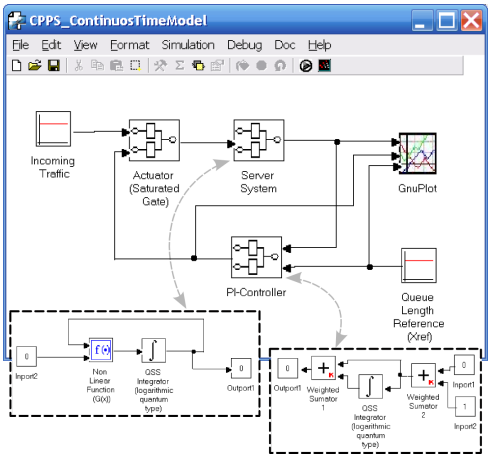

Figure 3. Continuous time CPPS model in PowerDEVS.

4.1 Continuous Control. Continuous Plant

The first goal of the process is to verify the control parameters for the CPPS by running simulations. At this point we have a continuous-time non-linear approximation of the Server System and a continuous-time specification of the controller.

The CPPS continuous-time model built in PowerDEVS is shown in Figure 3. Each block in the model represents either a Coupled DEVS model or an Atomic DEVS model. The inner details for the coupled models **Server System** and **PI-Controller** are also provided in Figure 3. The **Weighted Sumator** models implement the required multiplication factors to implement the control law (i.e., the K and T_i parameters and proper sumation/substraction signs). A **Non-Linear Function** atomic model (a component of the **Server System** coupled model) implements the non-linear expression of the right hand side of Eq.((1)). The **QSS Integrator** models locally implement the QSS methods (in this case, we selected the third order accurate QSS3 algorithm). All the mentioned models are part of the standard libraries of PowerDEVS.

We ran the simulation of this deterministic system for 30 seconds. Results are shown in Figure 4. We can see that the queue length response is slightly under damped, and stabilizes quickly to the reference value with a settling time of about 7 seconds (with a small overshoot at the beginning). These results (obtained in a full discrete–event framework) show a qualitative close match to the simulation results shown in Kihl et al. (2003) (obtained with a discrete–time numerical integration approach) for the same parameters and signal ranges.

4.2 Discrete Control. Continuous Plant

The second goal of the design process is to obtain a controller that can be implemented with an algorithm in a digital computer. For this aim we translate the continuous-time controller into a discrete-time controller, while keeping (for now) the continuous-time version of the plant. By applying the Euler method to the control law (4) we obtain the following discrete-time expression for the PI-controller:

$$z_{k+1} = z_k + e_k h \quad u_k = K(e_k + \frac{z_k}{T_i}) \tag{6}$$

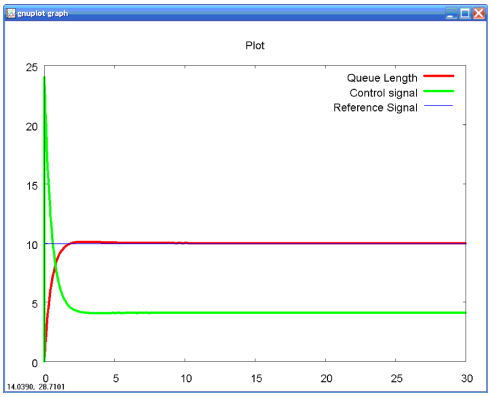

Figure 4. Continuous time CPPS simulation results.

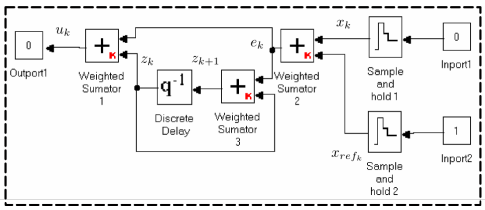

Figure 5. Discretized PI-Controller in PowerDEVS.

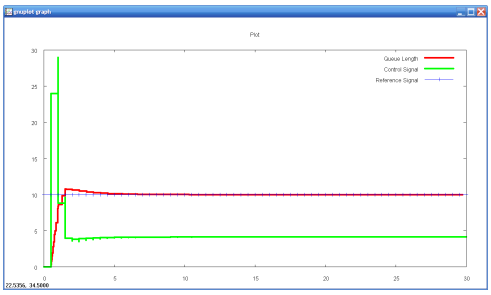

Figure 6. Discrete Time CPPS simulation results.

where $k \in Z$ is the discrete-time index and $h \in \Re^+$ is the time step size such as $z_k = z(t = kh)$. Now we proceed to replace the previous **PI-controller** DEVS coupled model by its discretized counterpart, which will implement Eq.(6). The block diagram of the discretized controller is shown in Figure 5, where the discrete versions of the input variables x_k and x_{ref_k} are obtained by means of **Sample and Hold** blocks, and the delay operation on the controller variable z is performed by a **Discrete Delay** block. These blocks are DEVS atomic models parametrized with a common sampling period of $h = 0.5$ seconds.

We ran the simulation of the new system with the discretized control for 30 seconds. The results are shown in Figure 6. The analysis of the results shows that the response reproduces closely the properties of the continuous–time simulation results of Figure 4, noting that in the discrete–time case the queue length response is slightly more under damped (with a more pronounced overshoot at the beginning). This is due to the control delay introduced by the discretized controller. The system stabilizes to the reference value in a time qualitatively similar to the continuous controller case, with a settling time of about 7 seconds. With these results we can adopt the discrete-controller design as a satisfactory solution.

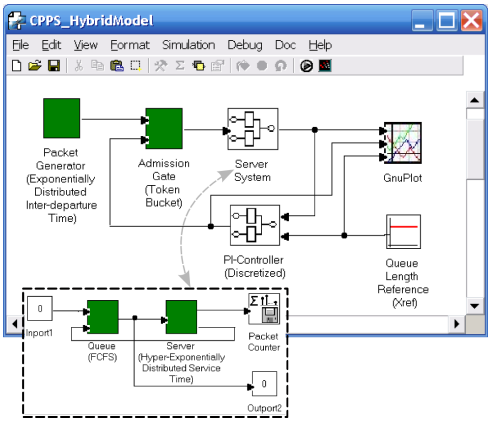

Figure 7. Hybrid CPPS model in PowerDEVS.

4.3 Discrete Control. Discrete Event Plant

Now, following a standard M&S design cycle, we should verify our controller design against a more accurate representation of the real system, getting rid of all kind of approximations as much as possible. This constitutes the third goal of our design process.

The real M/G/1 Server System is in fact a stochastic DEDS, and can be straightforwardly represented and executed under the DEVS framework. Thus, one of the advantages of our full DEVS–based methodology arises at this stage of the M&S process, where the verification activity against the real system involves a simple model replacement task (without switching formalisms or design tools). Then, the continuous–time continuous–variable components of our original model are replaced by their discrete–event counterparts. By doing so we obtain the Hybrid CPPS model shown in Figure 7. Traditional M&S methodologies, on the contrary, would usually involve some kind of special-purpose language-specific implementation of the Hybrid CPPS, i.e. the M/G/1 system and the controller algorithm. In the case of Kihl et al. (2003) it consisted in a custom code implemented by the authors in the C language.

The Hybrid CPPS model in Figure 7 implements the following DEVS atomic models for the networking system: a **Packet Generator**, an **Admission Gate**, a **Queue** and a **Server**. These models handle discrete entities individually (which represent network packets) on a continuous-time basis. The stochastic behavior of the network traffic is considered explicitly at the packet inter–generation times and the servicing times, so no approximation is involved. The queueing of the packets is achieved by a *First In-First Out* (FIFO) policy at the **Queue** block. A token bucket algorithm is implemented at the **Admission Gate** block, which is exactly like the one that would be implemented on a real-world target embedded piece of code. It generates internal tickets at a rate commanded by the control signal u coming from the Controller. Individual incoming packets will be accepted if there exist available tickets for them. Otherwise, the packets are rejected. The control signal will be updated at the time base imposed by the discrete-time controller (the control period) which in our case is set to h=0.5 from previous design phases.

Given the stochastic nature of the system we performed a set of 1000 simulations (of 30 seconds each) and obtained the average statistics of interest. This task was automated with the PowerDEVS–Scilab (Scilab, 2009) integrated environment for simulation and numerical computation. The results are shown in Figure 8, where the Queue Length measure is provided for the average of the set of simulations, and also for a single representative realization of the set. Both curves were obtained by sampling the queue lenght values every 1 second. The dotted curve shows a satisfactory regulation of the average Queue Length around the desired reference value of 10 packets (as specified by the system parameters in (5)). A qualitative comparison of our stochastic simulation results with those obtained in Kihl et al. (2003) for the discrete–event experiments (under the same conditions) shows a close match in terms of time response and excursion values for both the average and the particular realization curves.

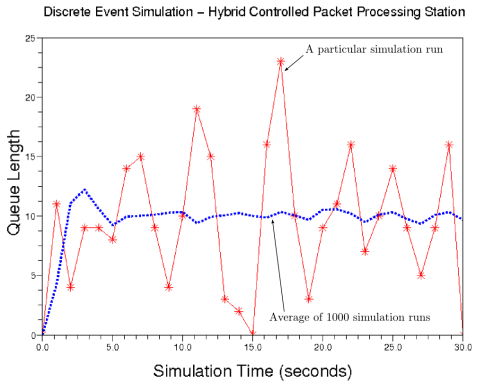

Figure 8. Hybrid Discrete Event CPPS simulation results.

4.4 Implementation and Hardware-In-The-Loop Execution

Our methodology allows to implement the obtained DEVS–based controller embedded in a real-world Network Processor Unit (NPU), a target chip for high-performance networking applications. We want to avoid the recoding of the solution designed, and port our Controller design from PowerDEVS to a DEVS–based executive capable of running into the NPU. To this aim we ported the ECD++ toolkit to run in real–time into a NPU, and developed libraries to communicate the DEVS models with the high speed network processing layers. The embedded DEVS models will then be able to control real-world traffic in a Hardware-in-the-Loop (HIL) fashion.

We used an Intel IXP2400 NPU, an OC-48/2.5 Gbps. line rate chip structured in two processing levels: the *Slow Data Path* with an Intel XScale Core processor, and the *Fast Data Path*, with 8 multithreaded dedicated microengines (ME). Both types of processing hardware (plus units for memory, control and I/O) coexist in the same chip, constituting a networking System on Chip (SoC). By means of supporting software libraries, we can orchestrate tasks between the Core processor and the MEs. Figure 9 shows the mapping between the Hardware Architecture and the Software Architecture. In a typical application, the MEs are programmed to do the line rate speed processing, taking advantage of special purpose hardware designs for latency hiding. The Core processor

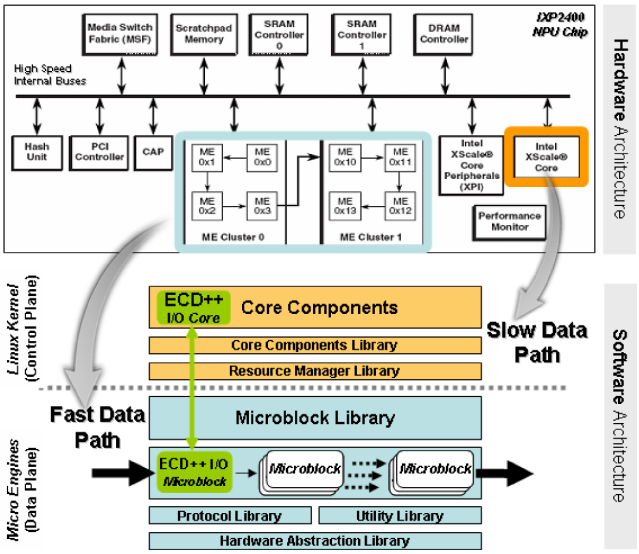

Figure 9. ECD++ embedded on the Intel IXP2400. Hardware and Software conceptual architectures.

usually performs control, management and exception tasks delegated by the MEs.

As depicted in Figure 9 ECD++ runs at the Slow Data Path. We developed an *ECD++ I/O Core* library as a collection of *Core Components* (Linux Kernel modules) which are used by the DEVS models in ECD++ to communicate with *microblocks* (at the Fast Data Path). Microblocks are the specialized software modules that perform protocol-specific tasks running multithreaded and pipelined inside the MEs. We developed an *ECD++ I/O microblock* to communicate with the DEVS models at the Core. With this infrastructure we are able to replace those DEVS models that represent the real system by equivalent I/O operations to the actual real-world system. In our case study, for the hybrid CPPS model depicted in Figure 7, only the Control model (and its Reference signal) will be the retained DEVS models running in ECD++. They communicate with the real-world networking system (Fast Data Path) using two signals: the sensing of the Queue Length signal (ME to Core) and the actuation of the Control Rate signal (Core to ME).

5. CONCLUSIONS

In this work we showed how to utilize the DEVS M&S framework as the basis of an end–to–end methodology for applying control theory into a network admission control problem, which yields an stochastic hybrid system. We showed how the usual challenges that arise in hybrid systems due to the needs for switching between M&S paradigms disappear by following a full DEVS–based methodology. Thanks to the QSS numerical methods, continuous systems can be seamlessly integrated with discrete-event systems, providing a very suitable common framework to integrate heterogeneous disciplines in the area of control of real-time computing and networking systems. Our approach enabled a seamless transition between DEVS–based tools for the final implementation of the obtained models into an Intel IXP2400 NPU. This fact greatly reduced the usual risks and efforts due to recoding and adaptation of solutions when they are ported to real embedded targets. The next steps will be focused on the design of more sophisticated control strategies and the testing of the solutions with real-world network traffic.

REFERENCES

Arzen, K.E., Robertsson, A., Henriksson, D., Johansson, M., Hjalmarsson, H., and Johansson, K.H. (2006). Conclusions of the ARTIST2 roadmap on control of computing systems. *SIGBED Rev.*, 3(3), 11–20.

Cassandras, C. and Lafortune, S. (2004). *Introduction to discrete event systems*. Kluwer Academic Publishers.

Castro, R., Kofman, E., and Wainer, G. (2009). A Formal Framework for Stochastic DEVS Modeling and Simulation. *Transactions of SCS (in print)*.

Cellier, F. and Kofman, E. (2006). *Continuous System Simulation*. Springer, New York.

Christin, N., Liebeherr, J., and Abdelzaher, T. (2002). A quantitative assured forwarding service. In *IEEE INFOCOM 2002*, volume 2.

Hellerstein, J. (2004). *Feedback control of computing systems*. Wiley-IEEE Press.

Hollot, C., Misra, V., Towsley, D., and Gong, W. (2001a). A control theoretic analysis of RED. In *IEEE INFOCOM 2001*, volume 3.

Hollot, C., Misra, V., Towsley, D., and Gong, W. (2001b). On designing improved controllers for AQM routers supporting TCP flows. In *IEEE INFOCOM 2001*, volume 3.

Intel (2004). *Intel IXP2400 Network Processors*. URL `http://download.intel.com/design/network/ProdBrf/27905302.pdf`.

Issariyakul, T. and Hossain, E. (2008). *Introduction to Network Simulator NS2*. Springer.

Kihl, M., Robertsson, A., and Wittenmark, B. (2003). Analysis of admission control mechanisms using nonlinear control theory. In *Proceedings of ISCC 2003*, volume 2, 1306–1311. Kiris-Kemer, Turkey.

Kofman, E., Lapadula, M., and Pagliero, E. (2003). PowerDEVS: A DEVS Based Environment for Hybrid System Modeling and Simulation. Technical Report LSD0306, LSD, UNR.

Kwon, W. and Kim, H. (2000). A survey of control theoretic approaches in wired and wireless communication networks. In *Proceedings of the Korea-Japan Joint Workshop*, volume 1, 30–45.

OMNeT++ Community (2004). OMNeT++ Discrete Event Simulation System. `www.omnetpp.org`.

Robertsson, A., Wittenmark, B., and Kihl, M. (2003). Analysis and Design of Admission Control in Web-server Systems. In *Proceedings of ACC'03*, 254–259. Denver, Colorado.

Scilab (2009). The open source platform for numerical computation. `http://www.scilab.org`.

Tipper, D. and Sundareshan, M. (1990). Numerical methods for modeling computer networks undernonstationary conditions. *IEEE Journal on Selected Areas in Communications*, 8(9), 1682–1695.

Wainer, G. (2009). *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. CRC Press (in print).

Zeigler, B., Kim, T., and Praehofer, H. (2000). *Theory of Modeling and Simulation. 2nd. edition*. Academic Press, New York.