Algoritmos Conservadores Para Simuladores Achatados DEVS y Cell-DEVS

Shafagh Jafer[1], Gabriel Wainer[1], Matías Bonaventura[2] y Rodrigo Castro[2]

[1] Dept. of Systems and Computer Engineering.
Carleton University, Centre of Visualization and Simulation (V-Sim),
1125 Colonel By Dr. Ottawa, ON, Canada.

[2] Departamento de Computación.
Universidad de Buenos Aires, FCEyN,
Pabellón I, Ciudad Universitaria, Buenos Aires, Argentina.

Resumen. Presentamos una arquitectura achatada para simuladores de modelos DEVS y CELL-DEVS, con el objetivo de permitir ejecución paralela mediante un mecanismo de sincronización conservador**.** Primero mostramos cómo una arquitectura achatada reduce los costos de comunicación entre los nodos. Luego proponemos un mecanismo de bloqueo para suspensión de Procesos Lógicos. Finalmente presentamos nuevos algoritmos basados en mensajes nulos para evitar errores accidentales y deadlocks. Los algoritmos se implementaron en el kernel WARPED, y mostramos cómo este método puede ser utilizado por cualquier simulador paralelo DEVS y Cell-DEVS.

1 Introducción

Dentro de las técnicas de modelado y simulación existentes, el formalismo DEVS (Discrete Event System Specification) [Zei00] provee un enfoque a eventos discretos que permite la construcción modular de modelos jerárquicos. DEVS es un entorno formal de trabajo basado en conceptos de sistemas dinámicos genéricos que permite el reuso de modelos y el ocultamiento de información mediante la construcción de modelos jerárquicos y modulares. Cell-DEVS [Wai09] es una extensión a la teoría de autómatas celulares [Wol86] que utiliza DEVS para representar cada celda. Este formalismo permite definir comportamientos celulares complejos con instrucciones simples y construir espacios celulares n-dimensionales para representar modelos a eventos discretos complejos. Desafortunadamente, cuando se intentan resolver problemas de simulación de modelos complejos (con DEVS, Cell-DEVS u otras técnicas), suelen existir problemas serios de utilización de recursos (en particular, la falta de suficiente memoria para ejecutar todo el sistema, o tiempos de ejecución muy prolongados). Para resolver estos problemas, diversas técnicas de simulación paralela y distribuida proveen mecanismos que permiten mejorar la utilización de recursos [Fuj01]. La comunidad propuso dos tipos de algoritmos de sincronización: los Conservadores (o pesimistas, también llamados Chandy-Misra-Bryant) [Bry77, Cha79] y los Optimistas (en particular, Time Warp) [Jef85]

A su vez, DEVS fue extendido para permitir la ejecución de eventos simultáneos. Parallel DEVS o P-DEVS [Cho94], permite una ejecución más eficiente de modelos en entornos paralelos y distribuidos conservando las propiedades principales del formalismo original y extendiéndolo para resolver las restricciones de procesamiento secuencial. Asimismo, Parallel Cell-DEVS [Wai00] resuelve restricciones de simula-

ción en serie permitiendo un mayor grado de paralelismo (incluyendo transiciones sin demora y múltiples eventos simultáneos en puertos externos).

Si bien se existen varios métodos que integraron DEVS con algoritmos Time-Warp [Nut06, Kim00], no existen métodos combinados con algoritmos conservadores. En este trabajo proponemos un método para implementar mecanismos de sincronización conservadores en el kernel de simulación WARPED [Mar99] de forma tal de proveer nuevos mecanismos de simulación paralela y distribuida para modelos DEVS y Cell-DEVS.

2 Background

DEVS [Zei00] es un formalismo genérico para describir cualquier tipo de sistema discreto que realice un número finito de cambios en intervalos finitos de tiempo. Un sistema modelado con DEVS se describe como una composición jerárquica de modelos, que pueden ser comportamentales (atómicos) o estructurales (acoplados). Formalmente, un **modelo atómico** DEVS se define por la siguiente estructura:

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

Cada posible estado $s \in S$ tiene asociado un *tiempo de vida* calculado por $ta: S \rightarrow R_0^+$. Si $S=s_1$ (en el instante t_1), luego de $ta(s_1)$ unidades de tiempo el sistema realiza una transición interna $s_2=\delta_{int}(s_1)$. $\delta_{int}: S \rightarrow S$ se denomina *función de transición interna*. Al mismo tiempo se produce un evento de salida $y_1=\lambda(s_1)$. La función $\lambda: S \rightarrow Y$ se denomina *función de salida*. Cuando un modelo recibe un evento de entrada $x \in X$, el estado cambia instantáneamente. El nuevo estado dependerá del evento de entrada, del estado anterior, y del tiempo transcurrido e desde la última transición, o sea, $s_4 = \delta_{ext}(s_3,e,x_1)$, siendo s_3 el estado en el instante t_3 y x_1 el evento que llega en el instante t_3+e (con $e \leq ta(s_3)$). La función $\delta_{ext}: S \times R_0^+ \times X \rightarrow S$ se denomina *función de transición externa*. Los modelos DEVS pueden ser acoplados modularmente y jerárquicamente. Un **modelo acoplado** DEVS se define formalmente según:

$$CM = < X_{self} , Y_{self} , D , \{M_i\} , \{I_i\} , \{Z_{ij}\} , Select >$$

CM es un conjunto de modelos M_i ($i \in D$) acoplados por medio de sus interfaces de E/S, *X* e *Y*. El conjunto *Ii* define los modelos influenciados por cada *Mi*. La función de traducción Z_{ij} convierte las salidas de los modelos en entradas de otros ($X_i \rightarrow Y_j$). *Select* es una función *priorización* que resuelve conflictos ante simultaneidad entre los M_i.

Esta definición de DEVS puede provocar situaciones ambiguas cuando a) la función *Select* de un modelo acoplado decide la prioridad de eventos entre sus modelos atómicos con transición interna en un mismo instante *ta*(s), y b) un modelo atómico recibe un evento externo en el exacto momento *ta*(s) en que tiene planificada su transición interna. En el caso a) se produce un efecto de serialización que podría no reflejar correctamente el comportamiento del sistema real. En el caso b) se puede optar por priorizar a la transición interna o a la externa indistintamente, quedando la otra serializada para el próximo ciclo de simulación, lo cual nuevamente podría no ser representativo del comportamiento del sistema real. Para evitar estas restricciones se propuso el formalismo P-DEVS [Cho94], que preserva todas las propiedades y caracte-

rísticas de acoplamiento jerárquico de DEVS, y se redefinen los modelos atómicos de la siguiente forma:

$$M = < X_M , Y_M , S , \delta_{ext} , \delta_{int} , \delta_{conf} , \lambda , ta >$$

X_M e Y_M aceptan la llegada simultánea de eventos externos $E=\{x_{1...}x_n / x \in X_M\}$, haciendo posible la evaluación conjunta $\delta_{ext}(s,e,E)$ de eventos. Si sucede una colisión temporal entre el instante de llegada de E y el tiempo de transición interna $e=ta(s)$, el modelador puede elegir mediante $\delta_{conf}(s,E)$ si desea el resultado $\delta_{ext}(\delta_{int}(s),e,E)$ o el resultado $\delta_{int}(\delta_{ext}(s,e,E))$. Es decir, en P-DEVS esta decisión se hace explícita en el modelo, resolviendo una de las causas de serialización de DEVS. A su vez, un modelo acoplado puede ahora activar a todos sus modelos atómicos inminentes en forma simultánea (es decir, no se necesita la función *Select*).

Un autómata celular [Wol86] es una cuadrícula regular n-dimensional infinita, donde cada celda posee un valor finito y un aparato de cómputo, que es responsable de actualizar el estado de las celdas utilizando una regla local que utiliza un conjunto finito de celdas cercanas (llamadas *vecindad* de una celda). Este formalismo de simulación a tiempo discreto actualiza el valor de todas las celdas en cada paso de tiempo. Como por lo general no todas celdas necesitan actualizarse en cada paso de tiempo, esto provoca una notable pérdida de tiempo. **Cell-DEVS** [Wai09] resuelve este problema definiendo cada celda como un modelo atómico DEVS. Cada celda utiliza una definición explícita de comportamiento temporal y puede ser especificada como:

$$TDC = < X , Y , S , N , delay , d , \delta_{int} , \delta_{ext} , \tau , \lambda , D >$$

donde N es el conjunto de eventos de entrada que activan la función de cálculo local (τ). Los cambios de estado pueden ser transmitidos a otros modelos luego de cumplido el tiempo d. Un modelo Cell-DEVS acoplado se define creando una matriz de celdas atómicas de algún tamaño y dimensión y conectando un número de celdas entre sí usando la relación de vecindad, como puede verse en la Figura 1. **CD++** es una herramienta de modelado que implementa las teorías de DEVS y Cell-DEVS utilizando los formalismos originales [Wai09].

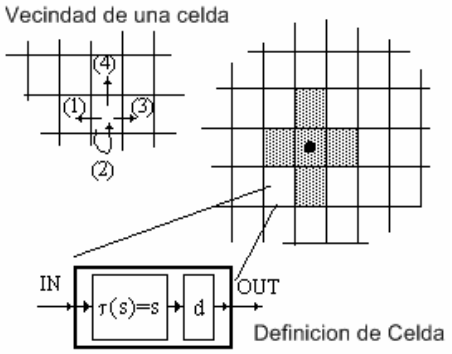


Fig. 1. Descripción informal de un modelo acoplado Cell-DEVS.

Los mecanismos de simulación paralela de eventos discretos dividen la simulación en un conjunto de *Procesos Lógicos* (PL) que intercambian mensajes entre sí. Para evitar errores de causalidad en el pasaje de mensajes, se definieron diversos mecanismos de sincronización, que pueden clasificarse en *conservadores* y *optimistas*. Los conservadores *evitan* la posibilidad de errores de causalidad, basándose en alguna estrategia para determinar cuándo es seguro procesar un evento. Los optimistas usan una solución de *detección y recuperación* invocando un mecanismo de rollback.

La idea de los mecanismos *conservadores* es determinar cuándo es seguro procesar un evento. Si un PL tiene un evento no procesado con un sello de tiempo (y ningún otro con sello de tiempo menor), y es imposible recibir un evento con sello de tiempo inferior, entonces el evento se puede procesar de forma segura. En [Cha79] se define que un PL procesa todos los mensajes con el mismo sello de tiempo en todos los enlaces de entrada (todo mensaje futuro tendrá un sellos de tiempo posterior). El procedimiento se repite mientras haya mensajes no procesados. Para evitar deadlocks se propuso un algoritmo basado en mensajes *nulos* que representan una *promesa* de que no se enviará un mensaje con sello de tiempo menor que el sello del mensaje nulo.

Los métodos *optimistas* [Jef85] detectan y recuperan los errores de causalidad, invocando un procedimiento de recuperación (explotando un mayor grado de paralelismo). Time Warp [Jef85] chequea si un mensaje recibido en un PL tiene sello de tiempo menor al del reloj del PL (error de causalidad: mensaje *dispersor*). En este caso, se produce un rollback, y se deshacen los efectos de todos los eventos prematuros (o sea, eventos procesados con mayor sello de tiempo que el dispersor). Para ello se almacena el estado del proceso (que se restaura durante el rollback). Un rollback, además, puede anular un mensaje transmitido, enviando un *antimensaje* que elimina al original cuando llega a su destino. El algoritmo Time Warp no entra en deadlock, porque los procesos individuales no entran en deadlock mientras tengan entradas.

Aunque diversas herramientas implementaron DEVS en entornos paralelos y distribuidos (por ejemplo DEVS-C++ [Zei96], DEVS/Grid [Seo04], DEVSCluster [Kim04], ADEVS [Nut06]), ninguna usa mecanismos conservadores. Por ello decidimos implementar una versión de CD++ puramente conservadora y paralela basada [Cha79] con el fin de comparar los resultados obtenidos con algoritmos optimistas.

3 Mecanismo de Simulación DEVS

En DEVS, la simulación se lleva a cabo por *procesadores* que pueden ser de dos tipos: *simulador* o *coordinador*. Los simuladores ejecutan los modelos atómicos, mientras que los coordinadores manejan los acoplados. Los simuladores invocan las funciones de transición de los modelos atómicos. Por otro lado, el coordinador tiene la responsabilidad de traducir los eventos de E/S de sus hijos y calcular sus próximos tiempos de transición. Además existe un *coordinador raíz*, que es responsable de llevar adelante la simulación y avanzar el tiempo de simulación virtual.

Parallel-CD++ usa un coordinador Raíz como planificador global para todos los nodos en la simulación. Basado en esta estructura, todos los eventos con el mismo sello de tiempo se planifican para ser procesados simultáneamente en los nodos disponibles [Tro03]. Este simulador introdujo dos tipos diferentes de coordinadores; uno cabecera y otro proxy para reducir la comunicación entre procesos. Al comienzo de la

simulación hay un PL en cada máquina, incluyendo uno o más procesadores DEVS (o sea, no todos los hijos de un mismo coordinador se sitúan en un mismo PL). Un coordinador se comunica con sus procesadores hijos mediante mensajes internos al proceso si estos residen dentro de un mismo PL, y mediante mensajes entre procesos si se sitúan en un PL remoto, como vemos en la Figura 2. El simulador, el coordinador cabecera y los coordinadores proxy están diseñados de forma tal que al recibir un mensaje cualquier respuesta tiene el mismo sello de tiempo (el coordinador raíz es el único procesador DEVS que avanza el tiempo). La única restricción necesaria es que dos o más eventos enviados de un mismo origen a otro destino deben preservar su orden. Este simulador paralelo nunca puede producir errores de causalidad y por ende, no se utiliza ningún mecanismo de sincronización a nivel de los PL.

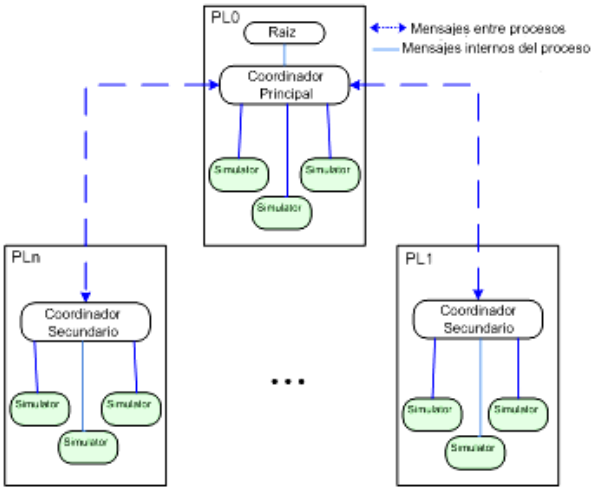


Fig. 2. Arquitectura del simulador Parallel-CD++ [Tro03].

4 Dominio de Aplicación: Control Distribuido

Nuestro objetivo a largo plazo es la aplicación de estos algoritmos para el diseño de aplicaciones distribuidas con el propósito de controlar la calidad de servicio (Quality of Service - QoS) en flujos de paquetes. La idea es usar DEVS embebido en un procesador de red (Network Processor - NP) para implementar control de tráfico de bajo nivel, teniendo en cuenta tanto la evolución instantánea de performance como también las políticas de QoS de alto nivel. Esta tarea plantea diversos desafíos de diseño e implementación. El problema abarca varios dominios (con distintas dinámicas temporales y lenguajes de especificación), desde políticas de QoS asignadas a distintos flujos de datos (que cambian pocas veces por día), algoritmos de perfilado de tráfico (que modifican estrategias de asignación de recursos en el orden de segundos), hasta los algoritmos de actuación (que toman decisiones granulares a nivel de paquetes individuales de red). Este escenario hace complejo el diseño y test integral del sistema. En especial, verificar y validar los efectos producidos por la introducción de cambios en algoritmos de un nivel específico sobre todo el sistema es muy complejo.

Para enfrentar esta complejidad, proponemos usar una metodología incremental basada en modelado y simulación, haciendo prototipado rápido de la aplicación distribuida. Cuando los prototipos tienen un comportamiento satisfactorio, se pasa de la simulación unificada a la implementación distribuida, simplemente desplegando los módulos del modelo hacia sus nodos de ejecución reales en la infraestructura destino. Estos nodos pueden ser nodos simples, nodos de un cluster, procesadores embebidos, o cualquier combinación entre ellos. En la Figura 3 se muestra la metodología.

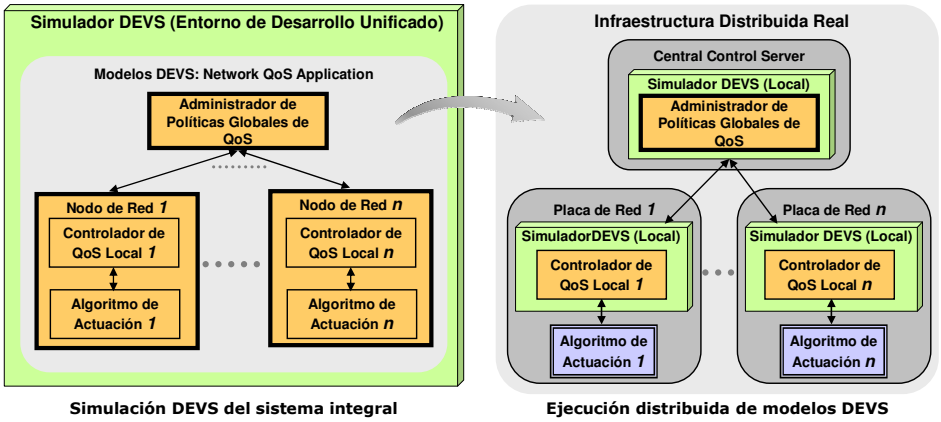


Fig. 3. Modelado y posterior despliegue en un entorno distribuido.

En cada nodo de ejecución de la infraestructura destino debe existir software capaz de a) ejecutar los modelos que le son asignados y b) comunicar los modelos con el sistema externo (hardware o software) con el cual interactúa la aplicación. En la transición desde la simulación unificada hasta el despliegue distribuido, siempre se utiliza el modelo original. El software de simulación pasa del rol de *motor de simulación* al rol de *motor de aplicación* operando en modalidad System-In-The-Loop. El simulador paralelo aquí propuesto permitirá ejecutar una variedad de tests con alta performance.

4.1 Control de QoS con el Simulador ECD++ y el Procesador IXP2400

Como caso de estudio de la metodología, diseñamos un sistema de control de QoS que acepta políticas globales y monitorea la tasa de descarte de paquetes (Drop Rate – DR) y otras métricas de performance del flujo real de red. Dependiendo de las políticas globales y del estado de DR, el sistema envía información de control actualizada a los algoritmos de bajo nivel que deciden y ejecutan el descarte de paquetes. La información de monitoreo y de control desde y hacia estos los algoritmos de bajo nivel se realiza a través de puertos de entrada/salida del simulador. Para esta aplicación, utilizamos Embedded CD++ (ECD++), una versión extendida de CD++ con capacidad de ejecutar en tiempo real embebida en procesadores de propósito específico. Instalamos ECD++ en el procesador Intel IXP2400 (con capacidad de línea de hasta OC-48/2.5 Gbps). La arquitectura IXP2400 está estructurada en 2 niveles de procesamiento en el mismo chip. El nivel 1 (*Camino Lento*) usa un procesador de propósito general (Intel Xscale Core) para resolver algoritmos complejos y administración de sistema. El ni-

vel 2 o (*Camino Rápido*) tiene 8 procesadores RISC (*microengines* – ME) especializados en el procesamiento paralelo de paquetes de datos a alta velocidad. La orquestación de los módulos de software en los distintos niveles de hardware se realiza mediante librerías estándar que conforman la Internet Exchange Architecture (IXA). La arquitectura IXP2400 permite construir motores de reglas flexibles y reconfigurables residentes en el NP, que pueden ser adaptadas dinámicamente sin interferir con la capacidad del NP de sostener su performance nominal de procesamiento de paquetes.

4.2 ECD++ y la Arquitectura del Procesador IXP2400

Un NP es System on Chip (SoC) para procesamiento de paquetes que concentra múltiples dispositivos de propósito específico en un único circuito integrado, simplificando y flexibilizando el desarrollo de hardware para telecomunicaciones. En nuestro caso, utilizamos una placa de red PCI Radisys ENP-2611, basada en el NP Intel IXP2400, con 3 ports ópticos Gigabit Ethernet.

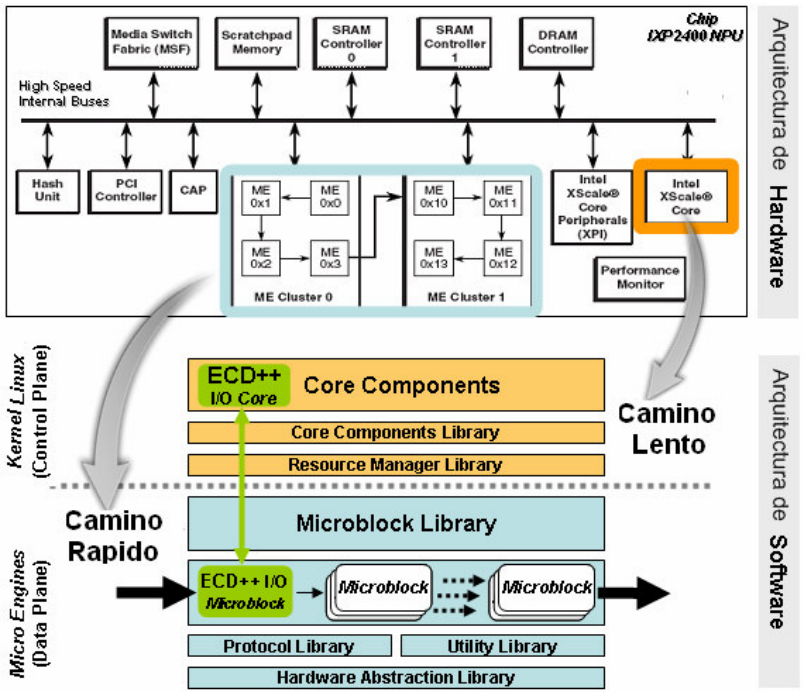


Fig. 4. El procesador IXP2400 y el motor ECD++.

Portamos ECD++ al procesador XScale, ejecutando como un Core Component (con un kernel Linux) como se ve en la Figura 4. ECD++ interactúa con el pipeline de MEs por medio de la arquitectura en capas IXA (Core Component, Resource Manager y Microblock), logrando comunicación controlada a registros y espacios de memoria compartidos. Los microblocks ejecutan en las MEs e implementan los algoritmos de

manipulación de paquetes. Para esta aplicación desarrollamos una librería ECD++ I/O Core, una colección de Core Components utilizados por ECD++ para comunicar los modelos DEVS con los microblocks. También desarrollamos un ECD++ I/O microblock para comunicarse con los modelos DEVS en el Core.

El algoritmo de bajo nivel que implementamos es Random Early Detection (RED) que descarta paquetes entrantes acorde a ciertas probabilidades asociadas al número de paquetes encolados esperando a ser procesados [Flo93]. Esta probabilidad crece linealmente desde 0 cuando se cruza un umbral mínimo de paquetes encolados (Queue Minimum Threshold - QmT) y alcanza su valor máximo 1 cuando se alcanza un umbral máximo (Queue Maximum Threshold - QMT).

Nuestro sistema de control de QoS envía comandos de control a RED indicando que QmT y QMT deben ser reajustados a nuevos valores. El modelo QoS Controller toma decisiones y eventualmente envía comandos de control al algoritmo RED. La Figura 5 muestra una representación de este modelo. Los estados reflejan la condición del sistema de procesamiento de paquetes que combina la métrica DR con la cantidad de tiempo ininterrumpido T (persistencia) en la cual se sostiene un valor dado de DR.

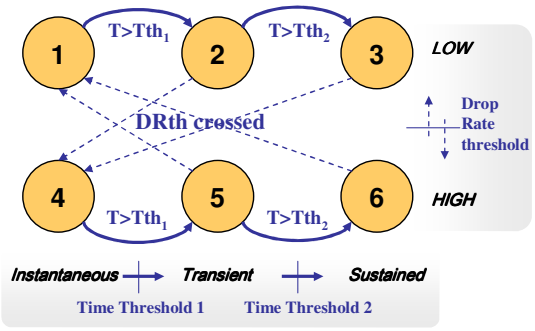


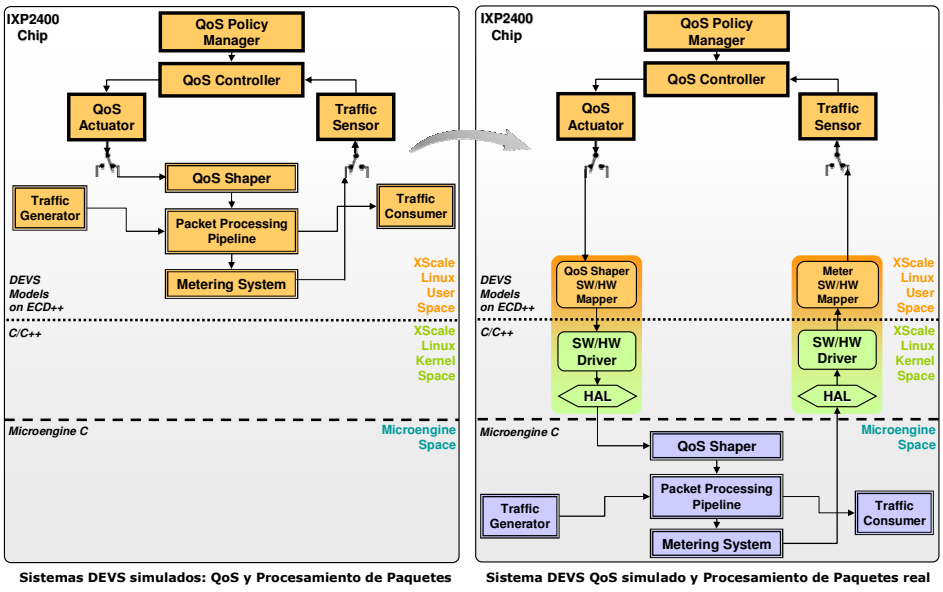
Fig. 5. Modelo DEVS atómico QoS Controller.

El controlador usa un umbral de tasa de descarte de paquetes (DR threshold – DRth) para clasificar las condiciones de tasa (alta o baja). Inicialmente, si DR≤DRth, pasamos al estado 1 y el tiempo T comienza a incrementarse. Si en cambio DR>DRth, pasamos al estado 4. Se definen 2 umbrales de tiempo Tth1 y Tth2 para distinguir entre 3 posibles valores de persistencia de tasa DR: Instantánea, Transitoria y Sostenida. Cuando T cruza esos umbrales, el modelo evoluciona por los estados 1→2→3 o 4→5→6 (dependiendo si el sistema se encuentra en nivel Bajo o Alto). Cuando el sistema cruza el umbral DRth, se reinicia T y el estado toma el valor 1 o 4. Cuando ocurre un cambio de política global de QoS se modifican los umbrales DRth, Th1 y Tth2, obteniendo así una adaptación del QoS Controller con un nuevo comportamiento en las acciones de control de RED; el cual adecuará sus parámetros QmT y QMT.

Esta metodología de diseño fue utilizada para todos los componentes del sistema. Como primera etapa de un ciclo de desarrollo incremental, se verificó el comportamiento del sistema simulado en ECD++ bajo un escenario simplificado, completamente modelado en DEVS y ejecutando en el XScale Core. En la Figura 6 (a) vemos un diagrama en bloques de los modelos que componen la aplicación de control. Los modelos QoS Actuator y Traffic Sensor en el nivel de baja velocidad se encargan de

enviar comandos de control y sensar el estado de DR respectivamente. Estos bloques se comunican con sus contrapartes en el nivel de alta velocidad (o Packet Processing system), consistiendo en los modelos QoS Shaper y Metering System.

Habiendo verificado la funcionalidad básica del QoS Controller en un escenario simplificado y con parámetros fijos, se pasó a la etapa de verificar la simulación comunicando los niveles de baja velocidad y de alta velocidad. En la Figura 6 (b) el hardware real de procesamiento de paquetes (las ME) reemplaza al conjunto de modelos que emulaban sus funciones (Generator, Shaper, Pipeline, Meter, Consumer).



(a) (b)

Fig. 6. Desarrollo incremental de sistema tipo System-In-The-Loop.

4.3 Mejoras Necesarias en el Software de Simulación

Como se discutió anteriormente, si bien logramos aplicar ECD++ para la simulación embebida en un nodo de red, operando en modo System-In-The-Loop, la característica distribuida de una aplicación como esta tiene mayores exigencias que el software de simulación no puede resolver en la actualidad. En particular, es necesario optimizar la performance de ejecución local en un nodo, así como la eficiencia y correctitud de la comunicación inter-nodos.

Estos requerimientos están relacionados con la escasez de recursos que suele ser característica de los sistemas embebidos. Este hecho, en combinación con la intención de ejecutar los modelos en tiempo real, hace que sea imprescindible optimizar los mecanismos de simulación locales. Para ello, proponemos una arquitectura achatada de simulación (en lugar de una jerárquica). El requerimiento de eficiencia de comunicación entre nodos es vital, debido a que la correctitud de la aplicación depende de una comunicación correcta (funcional y temporalmente) entre sus módulos distribuidos. Para ello, planteamos incorporar estrategias de simulación paralelas conservadoras.

5 Arquitectura del Simulador Conservador

El primer paso para definir el simulador conservador propuesto, es modificar la arquitectura jerárquica definida del simulador en [Tro03] por una achatada. La utilización de un mecanismo de simulación achatado en vez del jerárquico tradicional reduce los gastos innecesarios de comunicación al reducir al mínimo el número de mensajes intercambiados, y se ha demostrado que los simuladores achatados tienen una performance significativamente mejor que los jerárquicos [Kim00, Gli06, Kim04]. Por otro lado, aunque el simulador jerárquico en [Tro03] reduce la comunicación usando dos coordinadores especializados, los costos de comunicación siguen siendo altos.

5.1 Arquitectura Achatada del Simulador Conservador Propuesto.

El simulador abstracto fue rediseñado para reflejar las dos modificaciones propuestas (un simulador central por uno paralelo conservador, y el uso de una estructura achatada). La Figura 7 muestra la nueva arquitectura achatada.

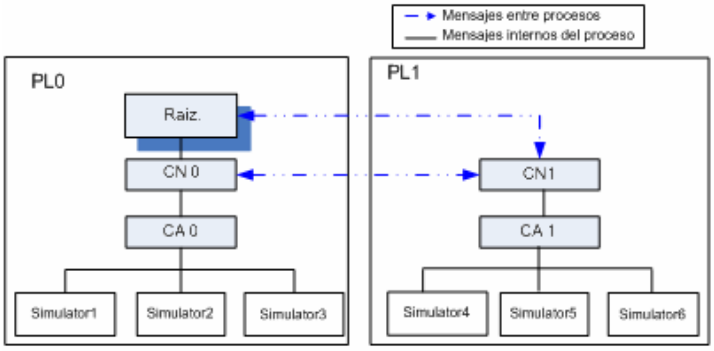


Fig. 7. Arquitectura achatada para el simulador conservador propuesto.

Como se ve en la Figura, los PL son creados en cada máquina encapsulando los procesadores DEVS. Se crea sólo un coordinador raíz en la máquina 0 (PL0) que interactúa con otros Coordinadores de Nodo (CN) mediante mensajes entre procesos (para CNs remotos) y mensajes dentro del mismo proceso (para CNs locales). El coordinador raíz es responsable de comenzar la simulación y realizar operaciones de E/S (logs, entrada/salida). También se crea sólo un CN en cada máquina, que actúa como controlador central local en su PL. El CN es el coordinador padre del Coordinador Achatado (CA) y envía los mensajes remotos recibidos del coordinador raíz u otro CN hacia el CA. Los simuladores son los procesadores hijos del CA local. Estos ejecutan los modelos atómicos DEVS y Cell-DEVS (son responsables de ejecutar las funciones definidas en sus modelos asociados). Cuando un simulador necesita comunicarse con otro simulador remoto que reside en otro PL, éste le envía el mensaje a su CA, quien a su vez se lo reenvía a su CN. Una vez que el mensaje se encuentra en el CN, este es enviado al CN destino. Incluso cuando dos simuladores son locales (están situados dentro del mismo PL), es necesario enviar sus mensajes a través de su CA padre. No existe comunicación directa entre simuladores.

5.2 Definición de los Mensajes

La simulación avanza gracias a la interacción de mensajes entre procesos. Existen dos clases de mensajes: de *contenido* y de *control.* La primer clase incluye el mensaje *externo* (*q*) y el mensaje de *salida* (*y*). La segunda clase incluye el mensaje de *inicialización* (*I*), el mensaje de *sincronización* (@), el mensaje *interno* (*) y el mensaje *listo* (*D*). Los mensajes externos y de salida son usados para intercambiar información de simulación entre modelos; los mensajes de inicialización comienzan la simulación; los mensajes @ y los internos disparan las funciones de salida y transición respectivamente en los modelos atómicos DEVS; y los mensajes D sincronizan los modelos.

Para implementar el algoritmo conservador del simulador propuesto hemos utilizado el kernel de WARPED [Mar99], un kernel de simulación de dominio público desarrollado originalmente en la Universidad de Cincinnati. WARPED provee una implementación del algoritmo Time Warp y múltiples extensiones. WARPED es un kernel de simulación gratuito, portable, modificable y extensible. En nuestro caso, sólo hemos utilizado los servicios que provee para facilitar la distribución de los ejecutables en los nodos. Como nuestro propósito es construir un simulador conservador, hemos utilizado WARPED como una capa intermedia para la creación de objetos de modelo (objetos de simulación), entidades que intercambian mensajes (con sello de tiempo), y responden a eventos aplicándolos al estado interno. De esta manera, el kernel fue utilizado para proveer las funcionalidades de enviar y recibir eventos entre los objetos de la simulación. El kernel también provee una definición simple de tiempo (que puede ser redefinida) y funciones para realizar operaciones de E/S consistentes.

Nuestra implementación del algoritmo conservador se encuentra situada en la capa del kernel de WARPED y está separada del código central del simulador. Es por esto que puede verse como una nueva extensión al kernel de WARPED. Este puede ser adoptado por otros investigadores que deseen utilizar el kernel de WARPED como capa de sincronización conservadora para sus simuladores basados en DEVS.

6 Diseño del Simulador Conservador CD++

Los algoritmos conservadores de sincronización Chandy-Misra-Bryant [Bry77] [Cha79] evitan la ocurrencia de errores de causalidad: si un PL tiene un evento sin procesar con un sello de tiempo t, y se garantiza que no puede recibirse otro evento con sello de tiempo menor, no existe posibilidad de que ocurran errores de causalidad. Si un PL tiene una lista de los eventos sin procesar de todos los otros PLs, puede procesar de forma segura el evento con menor sello de tiempo, ya que puede asegurarse que los próximos eventos llegarán con sello de tiempo mayor. Mientras se encuentren eventos sin procesar de todos los otros PL, este ciclo puede repetirse y la sincronización está garantizada. Sin embargo, si esta condición no se cumple, existe un riesgo de deadlock. Para resolver este tipo de deadlocks, la solución es buscar el *lookahead* del modelo (es decir, el menor sello de tiempo entre todos los eventos que el proceso puede programar en el futuro). La información de lookahead es acarreada entre los PLs utilizando *mensajes nulos*. Cada PL usa la información de lookahead que recibe de los otros PL para deducir un límite inferior en el sello de tiempo de los eventos futuros. Como resultado, el PL sabe que eventos es seguro procesar. La principal contra de este enfoque de sincronización es el tiempo consumido por el envío de mensajes nulos, que degradan la performance de simulación. En las próximas secciones describimos los detalles de diseño de nuestro simulador conservador.

6.1 Mecanismo de Planificación

Como mencionamos en la sección anterior, usamos el kernel de WARPED como capa intermedia para proveer el mecanismo básico de planificación en cada PL. Cada PL tiene un solo planificador LTSF (Lowest Time Stamp First) y una sola cola (inputQ) que mantiene todos los mensajes de entrada para todos los objetos de simulación (simuladores locales). Esta cola única mantiene por lo tanto los eventos sin procesar para cada PL. Además de los eventos de entrada, que llegan de otros PLs y el entorno, la cola inputQ también mantiene aquellos eventos que son enviados de un simulador local a otro simulador local. Este mecanismo de colas provee una estrategia de mensajes sencilla que sólo es responsable de una única cola para todos los objetos de simulación de un PL. Cada ciclo de simulación comienza por remover el primer elemento de inputQ y procesarlo. Los coordinadores DEVS descriptos en la Sección 4.1 son responsables de enviar mensajes al simulador local correspondiente utilizando el campo *destination_id* del evento de entrada. Por otro lado, cuando un objeto de simulación situado en un PL envía un mensaje a otro objeto remoto situado en un PL diferente, el director de comunicación es responsable de entregar este mensaje colocándolo en la cola inputQ del PL destinatario. Todo este proceso de enviar y recibir eventos con sello de tiempo entre objetos de simulación necesita ser estudiado de cerca para evitar errores accidentales y lograr un simulador conservador paralelo confiable. Un factor importante que debe ser considerado es que en este mecanismo conservador cada PL tiene su propio reloj virtual local y por lo tanto, si la sincronización no se realiza de forma adecuada, se producirán errores accidentales y los resultados de la simulación serán incorrectos. Dado que en DEVS se intercambian diferentes mensajes entre las unidades de procesamiento local y las remotas (coordinadores y objetos de simulación) es muy importante primero comprender como estos diferentes tipos de mensajes interactúan y si sólo afectan a los objetos de simulación locales o remotos. Estos diferentes paradigmas de mensajes son descriptos en detalle a continuación.

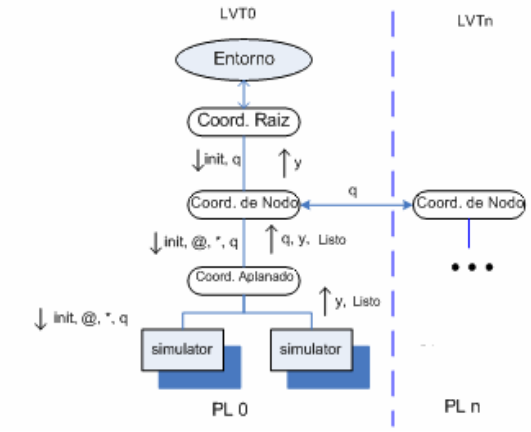


Fig. 8. Intercambio de mensajes en el simulador conservador CD++.

6.2 Definición de Mensajes

Como se discutió con anterioridad, se utilizan cinco tipos de mensajes entre el simulador, el CA, el CN y el coordinador raíz. La figura 8 muestra como estos mensajes se propagan durante la ejecución. Como se puede observar, el único mensaje que puede llegar de un PL remoto es el mensaje externo (*q*). El resto de los mensajes es local a cada PL y por lo tanto no puede ocasionar errores accidentales en otros PLs. Cuando un simulador (objeto de simulación) quiere enviar un mensaje a otro objeto remoto, primero envía un mensaje de salida (*y*) a su CA, luego el CA traduce este mensaje a un mensaje externo (*q*) y lo envía a su CN. Una vez que el CN recibe este mensaje, que contiene la información de que simuladores corren en cada PL, reenvía el mensaje (*q*) al PL del simulador destino a través del director de comunicaciones.

6.3 Implementación en WARPED

El algoritmo conservador presentado en la sección anterior (basado en la técnica de mensajes nulos), fue implementado en WARPED. Como el algoritmo de simulación DEVS es independiente del mecanismo de sincronización que se utiliza en su capa inferior, podemos intercambiar diferentes mecanismos de sincronización sin la necesidad de modificar el código del simulador. A continuación presentamos las dos modificaciones principales hechas al kernel de WARPED para lograr nuestros objetivos.

6.4 Mecanismo de Bloqueo de los PLs

Hemos mostrado que hay un solo planificador LTSF situado en cada PL que planifica la próxima ejecución del elemento superior de inputQ. Sin embargo, la ejecución sólo debe suceder si el PL ha recibido un mensaje de todos los otros PLs, para poder asegurarse que en el futuro no llegará un mensaje con sello de tiempo menor. Este mecanismo fue implementado basándose en el mecanismo de mensajes presentado en la sección 4.2, que muestra que el único mensaje que puede recibir un PL de otro PL es el mensaje externo (q). Por esta razón, en cada ciclo de ejecución, cuando el elemento superior de inputQ es un mensaje (q), el PL puede procesar este evento si y solo si ha recibido un mensaje (q) de todos los otros PL. Estos mensajes (q) pueden ser mensajes reales con información significativa y signo positivo, o bien pueden ser mensajes artificiales con signo negativo y sin información real (mensajes nulos). Estos mensajes artificiales son enviados solo con el propósito de sincronización. De esta manera, si el PL no ha recibido un mensaje (q) de todos los otros PLs, éste se bloqueará esperando que los otros PLs envíen sus mensajes (q). Implementamos este mecanismo de la siguiente manera:

```
Mientras(!finSimulacion)
   Si(inputQ->tope.tipo == 'q'){
      llegoMensajeDelPL(inputQ->tipo.PLOrigen);
      Si(mensajesRecibidosDeTodos == true)
          procesar(q);
      sino
          bloquearEstePL();
   }
```

Fig. 9. Mecanismo de bloqueo para los PLs del simulador conservador CD++.

6.5 Mecanismos de Lookahead y Mensajes Nulos

Cuando en un PL un objeto de simulación envía un mensaje de salida (traducido por el CN a un mensaje q) a un objeto de simulación remoto, significa que el PL des-

tino no recibirá un mensaje con sello de tiempo menor del PL origen. Por lo tanto, si bien sólo un simulador es el destinatario del mensaje, es suficiente para todo el PL que recibe el mensaje como para asegurarse que no recibirá en el futuro mensajes rezagados del PL que envió el mensaje. En consecuencia, cuando un PL envía un mensaje (q) a uno o más PLs, debe enviar también una copia artificial de ese mensaje al resto de los PLs que no son destinatarios originales del mensaje (q). Esto se hace enviando un mensaje (q) negativo que contiene el valor de lookahead del PL origen. El valor de lookahead le asegura al PL receptor que no recibirá ningún mensaje del PL origen por la duración especificada en el valor del lookahead. El valor de lookahead para un PL es el tiempo mínimo entre todos los mensajes de salida que serán enviados por sus simuladores locales menos el tiempo virtual local actual del PL. En la siguiente figura se muestra el algoritmo para el cálculo del lookahead y el mecanismo de envío de mensajes nulos.

```
EsperandoTodosLosMensajesListo();
     lookahead = tiempoMinMsjSalida(msjsDeSalida) - LVTActual;
```

Fig. 10. Computo del valor de lookahead para el simulador conservador CD++.

```
Si(msjAEnviar.tipo == 'q'){
   ParaTodos(PLs que no son destino del msj q){
      enviarMensajeNulo(lookahead);
   }
}
```

Fig. 11. Mecanismo de envío de mensajes nulos para el simulador conservador CD++.

El cómputo del lookahead se realiza en el CA que es el coordinador padre para todos los simuladores locales del ese PL. Sin embargo, la detección de la necesidad y el envío de mensajes nulos se realizan en la capa de WARPED. Como ya se ha mencionado, esto es para mantener el simulador separado del mecanismo de sincronización que se aplica en la capa del kernel WARPED.

6.6 Ejemplo de Ejecución

En esta sección mostramos la utilización de estos mecanismos. Inicialmente, mostramos la ejecución en tres nodos (PL0, PL1, PL2). Consideramos el caso en el que PL1 ya ha recibido mensajes externos (q) de los PL0 y PL2 y es momento de enviar un mensaje @ (mensajes 1.1 y 1.2) a los simuladores locales inminentes s2 y s3. En respuesta a los mensajes @, s1 y s2 envían sus salidas a través de un mensaje (y) que contiene su valor actualizado, el tiempo de esta salida, y el simulador destino del mensaje (y) (mensajes 1.3 y 1.4). Una vez que el CA recibe todos los mensajes de salida, calcula el lookahead como el tiempo mínimo de los mensajes de salida menos el LVT (el tiempo virtual local del PL0 es 100). Por lo tanto el valor del lookahead es el sello de tiempo del primer mensaje de salida que es el mensaje 1.3: (y, 110, s1) menos el LVT1 que es 100, por lo que el lookahead es 10. Ya que el destino del mensaje (y) es únicamente el simulador remoto situado en el PL0, un mensaje artificial (q) conteniendo el valor del lookahead y un valor negativo debe ser enviado del CN del PL1 hacia todos los otros PL que no eran los destinatarios originales del mensaje. Por lo tanto el CN de PL1 enviará un mensaje (q) al PL0 con valor positivo (q, 110, s1), y un mensaje (q) con valor negativo (mensaje nulo) al PL2 (-q, 10) donde el signo negativo indica que es un mensaje nulo y 10 indica el valor del lookahead del PL1. Cuando el

PL0 y el PL2 reciben estos mensajes (q), sólo pueden procesarlos sí ya han recibido un mensajes (q) de todos de los otros PLs.

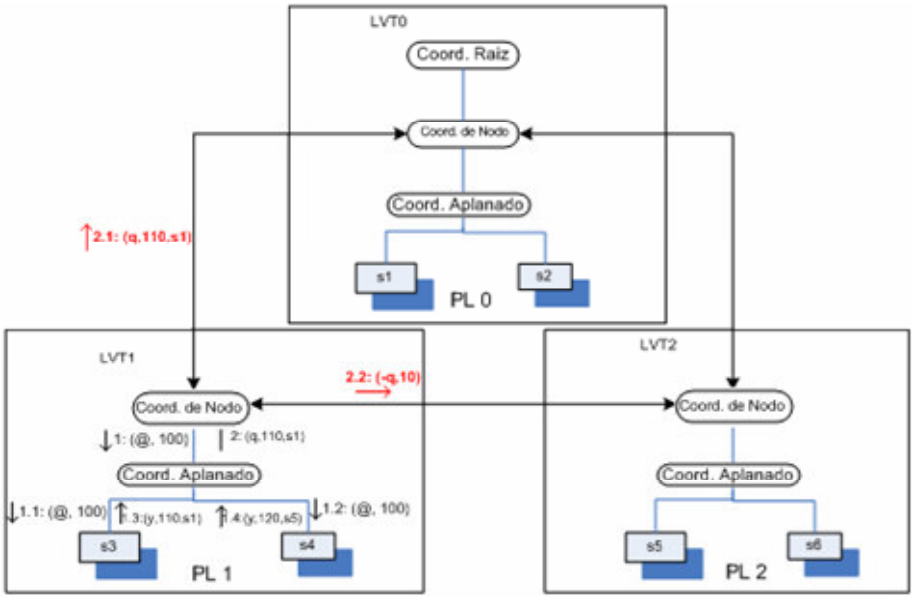


Fig. 12. Ejecución del el simulador paralelo conservador propuesto.

7 Conclusiones

En este artículo mostramos cómo atacar el problema de la ejecución de modelos DEVS y Cell-DEVS en ambientes paralelos y distribuidos. Inicialmente desarrollamos una implementación en ambientes centralizados (CD++) que fue extendida a una arquitectura multi-core como la disponible los procesadores de red Intel IXP. ECD++ fue implementado en este entorno, proveyendo un mecanismo para desarrollo de aplicaciones avanzadas usando una metodología basada en modelos, y utilizando simulación en todos los pasos del desarrollo. Mostramos brevemente cómo esta metodología puede utilizarse para el desarrollo de algoritmos de control de QoS en entornos distribuidos. Cuando estos modelos crecen en complejidad, las arquitecturas de simulación tradicionales tienen problemas de performance. Por ese motivo, la arquitectura jerárquica original del simulador fue reemplazada por una achatada para reducir los costos de comunicación entre los nodos. Mostramos cómo dicha arquitectura achatada simplifica los mecanismos de comunicación y pasaje de mensajes entre PLs remotos al permitir que cada PL se comunique directamente con otro PL sin necesidad de utilizar un sincronizador central. También presentamos una nueva extensión al kernel de WARPED que permite soportar un mecanismo de sincronización conservadora basado en los algoritmos clásicos de envío de mensajes nulos y lookahead. El mecanismo de sincronización se implementó en la capa de WARPED para proveer un protocolo conservador que no esté acoplado al simulador y por lo tanto permita intercambiar fácilmente los algoritmos de sincronización sin modificar el código del simulador. Por último, probamos como el algoritmo propuesto evita errores accidentales y deadlocks mostrando la ejecución un caso de estudio.

Los próximos pasos son realizar un análisis preciso de performance para probar la capacidad del simulador conservador propuesto en términos de cantidad de nodos de la simulación, complejidad y tamaño del modelo. También compararemos este simulador conservador con el simulador paralelo anterior que implementa un mecanismo de sincronización centralizada. Extenderemos el simulador para permitir una simulación hibrida conservadora-optimista como por ejemplo Local Time Warp [Raj93].

8 Referencias

[Bry77] Bryant, R. E. "Simulation of packet communication architecture computer systems". Massachusetts Institute of Technology. Cambridge, MA. USA. 1977.

[Cha79] Chandy, K.; Misra, J. "Distributed Simulation: A Case Study in Design and Verification of Distributed-Programs." IEEE Transactions on Software Engineering, 440-452. 1979.

[Cho94] Chow, A.C.; Zeigler, B.P. "Parallel DEVS: A parallel, hierarchical, modular modeling formalism." Proceedings of the Winter Simulation Conference. Orlando, FL. USA. 1994.

[Flo93] Floyd, S. and Jacobson, V., Random Early Detection gateways for Congestion Avoidance V.1 N.4, August 1993, p. 397-413.

[Fuj01] Fujimoto, R.M. "Parallel and Distributed Simulation Systems." Proceedings of the Winter Computer Simulation Conference. Phoenix, AZ. USA. 2001.

[Gli06] “Advanced Parallel simulation techniques for Cell-DEVS models”. G. Wainer, E. Glinsky. In Simulation News Europe. EUROSIM. Special Issue on Parallel and Distributed Simulation. Vol. 16, No. 2. September 2006. pp. 25-36.

[Jef85] Jefferson, D. "Virtual Time". ACM Transactions on Programming Languages and Systems. 7(3):405-425. 1985.

[Kim00] Kim, K.; Kang W.; Sagong, B.; Seo, H. "Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One." Proceedings of the 33rd Annual Simulation Symposium. Washington DC, USA. 2000.

[Kim04] Kim, K.; Kang, W. "CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One". International Conference on Computational Science and Its Applications. Assisi, Italy. 2004.

[Mar99] Martin, D. E.; McBrayer, T. J.; Radhakrishnan, R.; Wilsey, P. A. "WARPED - A Time Warp Parallel Discrete Event Simulator (Documentation for version 1.0)". Available at: http://www.ececs.uc.edu/~paw/warped/doc/index.html. 1999. [Accessed April, 2009].

[Nut06] Nutaro, J. ADEVS website. Available at: http://www.ece.arizona.edu/~nutaro. [Accessed July, 2007].

[Raj93] Rajaei, H., Ayani R., and Thorelli, L-E., "The Local Time Warp Approach to Parallel Simulation". Proc. of Workshop on Parallel and Distributed Simulation, San Diego, 1993.

[Seo04] Seo C.; Park, S.; Kim, B.; Cheon, S.; Zeigler, B. "Implementation of distributed high-performance DEVS simulation framework in the Grid computing environment". Advanced Simulation Technologies Conference (ASTC). Arlington, VA. USA. 2004.

[Tro03] “Implementing Parallel Cell-DEVS”. A. Troccoli, G. Wainer. In Proceedings of 36th IEEE/SCS Annual Simulation Symposium. Orlando, FL. U.S.A. 2003.

[Wai00] Wainer, G. "Improved cellular models with Parallel Cell-DEVS". Transactions of the Society for Computer Simulation International. Vol. 17, No. 2, pp. 73-88. 2000.

[Wai09] Wainer, G. “Discrete-Event Modeling and Simulation: a practitioner’s approach”. CRC Press. 2009.

[Wol86] Wolfram, S. "Theory and applications of cellular automata". Vol. 1. Advances Series on Complex Systems. World Scientific. Singapore. 1986.

[Zei96] Zeigler, B.; Moon, Y.; Kim, D.; Kim, J. G. "DEVS-C++: A high performance modeling and simulation environment". The 29th Hawaii Intl. Conference on System Sciences. 1996.

[Zei00] Zeigler, B.; Kim, T.; Praehofer, H. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". Academic Press. 2000.