UNA ARQUITECTURA DISTRIBUIDA PARA EL DESARROLLO DE SIMULADORES DE ENTRENAMIENTO

Boroni Gustavo, Vénere Marcelo
ISISTAN - Universidad Nacional del Centro, Pinto 399, Tandil – TEL.: 02293-442202
gboroni@exa.unicen.edu.ar, venerem@exa.unicen.edu.ar

Resumen. En el presente trabajo se propone una arquitectura distribuida orientada al desarrollo de simuladores de entrenamiento simultáneo de varios operadores con la supervisión de un único instructor. Se analizaron diferentes técnicas modeladoras de especificaciones arquitectónicas, sobre las cuales se evaluaron esencialmente los factores robustez y performance, de forma tal que problemas en un proceso o equipo no afecte al resto del sistema. Por último, se diseño un algoritmo que emplea diversos mecanismos de sincronización y coordinación de procesos distribuidos y en paralelo, el cual permite al sistema mantener los datos consistentes dentro del ambiente de simulación. ***Palabras clave*** - simulación distribuida, sincronización y coordinación de procesos, procesos en paralelo.

I. Introducción

Una de las áreas que ha logrado situarse en una clara posición de liderazgo y competitividad dentro del sector de electrónica e informática es, sin duda, la simulación orientada al entrenamiento. Tradicionalmente, la defensa ha sido el motor de desarrollo en este campo, al demandar las aplicaciones más importantes para los complejos sistemas militares y que posteriormente han sido trasladadas a entornos civiles, donde han adquirido un creciente protagonismo [1][11][8].

El porque reemplazar el mundo real por una versión "virtual" obedeció básicamente a tres razones:

- *Seguridad*. El mundo real es peligroso. Los estudiantes no deberían arriesgar su vida (o la de otras personas) o asustarse en el proceso de aprender su oficio.
- La *simplicidad, facilidad y compresión de tiempo* que se logran con la versión virtual. Las simulaciones pueden ser una manera conveniente y convincente de sintetizar la realidad.
- *Economía*. Las simulaciones son en general menos costosas que aprender en el mundo real.

Haciendo una clasificación elemental, se puede hablar de dos tipos de simuladores: los entrenadores, cuyo objetivo es facilitar el conocimiento de un sistema y la toma de decisiones; y los operacionales, en los que se busca predecir el futuro, a partir de una situación conocida, para poder establecer líneas de acción.

En los últimos años, la utilización de simuladores entrenadores comenzó a tomar gran importancia en áreas tales como, operadores de centrales industriales, grúas y maquinarias complejas, centrales nucleares y toda clase de unidades de transporte (aviones, barcos, submarinos, etc.). La ventaja del uso de estos simuladores es que resulta sumamente costoso y en algunos casos imposible entrenar a los operadores sobre los equipos verdaderos. No obstante, la mayoría de los simuladores que se encuentran hoy en día sólo posibilitan el entrenamiento de a un operario por vez y esto claramente marca una desventaja. Esto puede verse en el adiestramiento de pilotos de avión los cuales no pueden realizar ejercicios de manera conjunta (por ejemplo: teniendo varios simuladores de avión).

A partir de esta idea surgió la propuesta de diseñar una arquitectura de software que permita implementar simuladores capaces de entrenar de manera simultánea a varios operadores monitoreados por un instructor. El problema tiene similitudes con los puestos múltiples de los juegos en red, sin embargo no están pensados como herramientas de entrenamiento con la inclusión de un puesto para el instructor, donde el mismo lleve la administración de toda la simulación e incluso pueda tomar el control de cualquier puesto en caso de ser necesario.

Un ejemplo de esto último puede ser el entrenamiento de varios pilotos de aviones comerciales de manera simultánea (**Fig. 1**). Por un lado, en cada puesto de entrenamiento se representará una verdadera cabina de avión, sobre la cual el piloto va a realizar un plan de vuelo específico (una simulación particular por cada puesto de entrenamiento). Por otro lado, el puesto de instructor tendrá todos los medios necesarios para llevar el control de toda la simulación (una simulación general que involucra a todos los puestos). El mismo podrá además seleccionar un escenario donde volarán los aviones, configurar los distintos tipos de aviones (uno por cada puesto de entrenamiento), monitorear el desempeño de los pilotos, y generar eventos especiales tales como cambios en los factores climáticos, inhabilitaciones para aterrizar, etc.

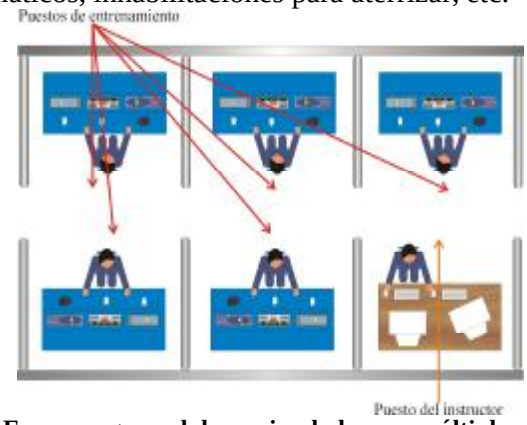

Fig. 1. Esquema general de un simulador con múltiples puestos.

Sin embargo en el afán de obtener beneficios a partir de simulaciones, no se debe dejar de lado la efectividad del adiestramiento. Para ello, se necesitará que sobre los puestos de entrenamiento los operarios dispongan de la funcionalidad equivalente y en lo posible puedan sentirse en el ambiente real. En este sentido los requerimientos pueden ser resumidos en:

a. Simular los procesos físicos intervinientes.
b. Utilizar elementos de realidad virtual para que el ambiente sea percibido lo menos sintético y artificial posible [5].
c. "*Que el simulador resultante sea una aplicación de tiempo real*". De no cumplirse este punto no podrá representarse la realidad [10].

II. Diseño arquitectónico

Obtener "a la primera" una correcta especificación de la arquitectura no es una tarea fácil, sobre todo si debe satisfacer requerimientos tales como rendimiento y fiabilidad. En general, resulta dificultoso mejorar el grado de satisfacción de un requisito sin perjudicar a otro, ni modificar la funcionalidad del sistema. Sin embargo en el trabajo de Bosch y Molin *et al* [3] se analiza este problema y se proponen técnicas que consiguen especificaciones arquitectónicas garantizando la invariabilidad de la funcionalidad. Estas técnicas están organizadas en cinco categorías y definen diferentes puntos de partida para construcción de una arquitectura o modelo: imposición de un estilo arquitectónico, imposición de un patrón arquitectónico, aplicación de un patrón de diseño, conversión de un requisito no funcional a funcional y división de requisitos.

Luego de analizar los requerimientos impuestos sobre cada categoría, se llegó a la conclusión que la que más se adaptaba al problema propuesto es la imposición del estilo. La principal razón es que la estructura de soporte para el modelo genérico se corresponde de manera casi directa con la arquitectura tipo *dispatch* o *repartidor*. En un aspecto amplio el repartidor realiza lo siguiente: centraliza la información procesada por los clientes, genera datos a partir de modelos propios, recalcula el nuevo estado del sistema, y por último, redistribuye el resultado para quienes lo necesiten. Más concretamente las partes que componen la arquitectura son:

- Cliente: es la interfaz que se encarga de ocultar la complejidad del sistema al usuario. Las principales funcionalidades que tendrán asociados son:
  - Capturas de información: capturar el tráfico de la red generada por otras maquinas.
  - Generación de información: generar parte del tráfico.
  - Consumo de información: recibir cierto tipo de tráfico.
- Repartidor (*Dispatch Server*): es el elemento que coordina a todos los clientes, y que además, captura, procesa y envía información al sistema.

El esquema propuesto comprende a varios puestos conectados por una LAN, donde se considera clientes a los distintos **tipos** de procesos con los cuales el repartidor establece una comunicación, y no a la cantidad de procesos que se están ejecutando. Es decir, cada cliente es un conjunto de procesos que tienen la misma funcionalidad. Por ejemplo en un esquema con cinco puestos de entrenamiento idénticos sobre los que se ejecutan tres procesos clientes, el repartidor verá solamente tres clientes y no quince.

La **Fig. 2** muestra el esquema utilizado: el repartidor se encuentra en uno de los equipos del puesto instructor, y abre un puerto de comunicación por cada tipo de cliente habilitado. A su vez, un puesto de entrenamiento está formado por uno o más equipos sobre los que se ejecutan varios procesos, donde no todos son necesariamente tipo cliente. Por ejemplo: sobre uno de los equipos de entrenamiento (puesto 1 de la figura) puede ejecutarse un proceso cliente que calcula e informa una posición (cliente 2), otro proceso cliente que lee las posiciones de los demás usuarios (cliente 3) y un proceso no cliente que actualiza la interfaz.

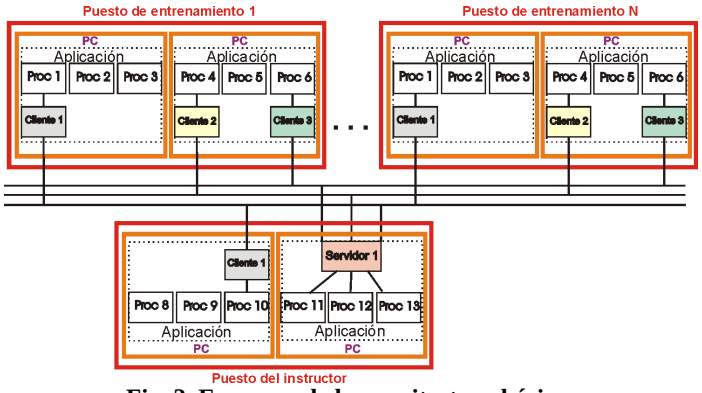

Fig. 2. Esquema de la arquitectura básica.

Sobre esta propuesta se tiene además un proceso instructor que actúa como centralizador y sincronizador de los datos generados por el sistema completo (También se lo menciona como proceso administrador de la simulación). Las funcionalidades más importantes que están asociadas a este proceso son:

- Lograr que los puestos de entrenamiento tengan los datos actualizados;
- Centralizar toda la información del sistema y distribuirla a los distintos puestos, dada una base de tiempo programable.

III. Administración de la simulación y sincronización de procesos

Anteriormente se mencionó que el procesamiento del sistema completo va a estar dividido en una simulación general y una simulación particular por puesto de entrenamiento. A su vez cada simulación particular puede estar compuesta por una serie de procesos distribuidos. Para que esta estructura funcione y que el sistema tenga datos consistentes es necesario desde el administrador de la simulación un mecanismo que permita sincronizar los procesos vinculados a los clientes [2].

La solución fue utilizar el tiempo global de respuesta del sistema (GVT = *Global Virtual Time*) tal como se define en Martín *et al* [7], el cual representa el tiempo mínimo en el que todo el sistema se encuentra consistente con respecto a sus datos. Teniendo en cuenta que dentro del puesto del instructor existirá un proceso que va a administrar toda la simulación, podemos decir que el GVT es el tiempo asociado al ciclo del método que va a sincronizar toda la simulación, y que va a generar el próximo paso de tiempo.

Como en los sistemas distribuidos es difícil determinar la consistencia de los datos (dada la distribución de los mismos) y además, la misma puede requerir de tiempos diferentes para que exista, no resulta trivial determinar que valor va a tener el tiempo GVT. También es importante mencionar que GVT va a estar determinado por el tipo de problema a resolver y la cantidad de procesos clientes que se estén ejecutando. Por todo lo anterior, no existe una fórmula que determine que valor debe tener GVT. La manera de determinar dicho valor será a través de pruebas sobre el sistema que se implemente, variando la cantidad de procesos clientes que interesen en el ciclo de ejecución.

Dado el requerimiento de tiempo real de este tipo de sistemas, el objetivo es entonces, minimizar GVT manteniéndolo por debajo de un cierto valor de diseño asociado al sistema que se desea implementar. Sobre esta idea se diseño una variante del método de sincronización *Time Warp* propuesto por Ling Yi-Bing *et al* [6]. *Time Warp* es un método de sincronización para sistemas distribuidos, el cual tiene por objetivo final acelerar en todo momento el tiempo de respuesta del sistema completo, independientemente del tiempo local de cada proceso involucrado (*Time Warp* es también conocido como "*optimistic synchronization method*").

A partir de un problema de sincronismo de procesos se puede optar como solución por dos mecanismos de sincronización, uno es el método conservativo (*conservative method*) y el otro es el método optimista (*optimistic method*). En el método conservativo el proceso receptor permanecerá inactivo mientras no se cumplan todas las precondiciones para su ejecución. En este caso, el proceso del instructor permanecería ocioso hasta que llegue la información de todos los procesos clientes asociados a los puestos de entrenamiento. En el método optimista el proceso receptor continuará con su ejecución aún en el caso de que no se cumpla alguna precondición.

Claramente se observa que el método que minimiza GVT es el método optimista, sin embargo hay que tener en cuenta que de no cumplirse con alguna precondición aparecerá un problema de inconsistencia de los datos. Una solución para esto último es la sugerida por Damani *et al* [4] que propone dos alternativas: la primera es que el proceso receptor realice un *rollback* general a partir del cual reanuda la ejecución de todos los procesos (*aggressive cancellation*). La segunda el *rollback* será realizado por el proceso que haya fallado (*lazy cancellation*), por lo que el proceso receptor continuará con la ejecución suponiendo que la falla será solucionada y no va a influir en la consistencia de la información. Para cualquier opción, en caso de que el proceso siga fallando puede optarse por darlo de baja.

La solución fue utilizar como forma de clasificación *optimistic method + lazy cancellation*, y sobre las mismas se diseño el mecanismo de sincronización resultante. Este emplea al método *Time Warp* como base, con el agregado de:

- un esquema de procesos con prioridades asociados a la lectura/escritura de puertos, cálculos auxiliares y actualización de la interfaz;
- rangos de tiempo en los cuales dichos procesos tendrán vigencia (por ejemplo: un rango de tiempo donde se leen datos generados por procesos que han sido distribuidos).
- un esquema de semáforo que coordina la atención de las tareas antes mencionadas.

A continuación se muestra en pseudocódigo, el algoritmo del método vinculado a la administración de la simulación y sincronización de procesos.

Mientras la simulación este corriendo
{ 1. Para una base tiempo t1, cambiar a modo recepción de mensajes {Setea mayor prioridad a los procesos asociados a la lectura de los puertos de comunicación, y habilita los mismos para su acceso}
2. Realizar todos los cálculos propios del administrador. Por ejemplo: en el puesto del instructor se calcula una parte de la simulación general (duración: t2);
3. Para una base tiempo t3, cambiar a modo envío de mensajes {Setea mayor prioridad a los procesos asociados a la escritura de los puertos de comunicación, y habilita los mismos para su acceso}
4. Actualizar la interfaz (duración: t4);
5. Detectar puestos con posibles problemas y dar de baja aquellos con problemas confirmados}

Del algoritmo anterior se desprende que GVT ≈ t1 + t2 + t3 + t4. En el caso de que se hayan recibido datos actualizados de todos los puestos requeridos antes de finalizar la base de tiempo t1, se dará por concluida dicha etapa, minimizando de esta forma siempre que se pueda el tiempo GVT. De no cumplirse esta condición entrará en vigencia el rango de tiempo t1 y GVT tendrá el valor máximo permitido por diseño.

IV. Conclusiones

A través del análisis de las diferentes alternativas propuestas en Bosch y Molin *et al* [4], se determinó que la imposición de un estilo distribuido sobre una arquitectura tipo *dispatch* se adaptaba más fielmente al tipo de simulador propuesto. Una vez elegido el estilo se procedió a realizar una descripción detallada de la arquitectura propuesta, cómo estaba constituida y que procesos se deberían sincronizar para que funcione correctamente.

Por ultimo, se propuso un algoritmo de sincronización y administración de procesos que emplea las técnicas de sincronización *optimistic method + lazy cancellation.*

V. Bibliografía

[1] Basdogan, Ho, Srinivasan, Small and Dawson. "Force Interactions in Laparoscopic Simulations: Optic Rendering of Soft Tissues". Proceedings of the Medicine Meets Virtual Reality Conference, San Diego, CA Jan., 1998.

[2] Bimber O., Encarnacao LM., Stork A. "A multi-layered architecture for sketch-based interaction within virtual environments". Computers & Graphics - UK, 2000, Vol 24, Iss 6, pp 851-867.

[3] Bosch J. y Molin P. "Software Architecture Design: Evaluation and Transformation". En in Proceedings of the IEEE Engineering of Computer Based Systems Symposium (ECBS99), Dec 1999.

[4] Damani Om. P., Garg K. "Fault-Tolerant Distributed Simulation". Workshop on Parallel and Distributed Simulation, 1998.

[5] Lathan CE., Tracey MR., Sebrechts MM., Clawson DM., Higgins GA. **"**Using virtual environments as training simulators: Measuring transfer". Handbook of Virtual Environments: Design, Implementation, and Applications (Series: HUMAN FACTORS AND ERGONOMICS SERIES), 2002, pp 403-414.

[6] Ling Yi-Bing, Fishwick A. "Asynchronous Parallel Discrete Event Simulation". IEEE Transactions on systems, man and cybernetics. Vol. XX, 1995.

[7] Martín P., Rümekaster M. "Tolerant Synchronization for Distributed Simulations of Interconnected Computers Networks". Workshop on Parallel and Distributed Simulation, 1997.

[8] Muñoz, G. EL CAOI: nueve años capacitando a operadores de centrales termoeléctricas, Boletín IIE, Vol. 17, núm. 4, pp. 169-172.

[9] Park S, Lee S, Moon I. "Superstructure of yOTS - The network-based chemical process operator training system for multiple trainees". Korean Journal of Chemical Engineering, 2001, Vol 18, Iss 6, pp 788-795.

[10] Wang SW., Zheng L. "Dynamic and real-time simulation of EMS and air-conditioning system as a 'living' environment for learning/training". Automation in Construction, 2001, Vol 10, Iss 4, pp 487-505.

[11] Ziegler, R; Fisher, G; Muller, W; Gobel, M. "A Virtual Reality Medical Training System". In Ayache, N. (ed.), Proceeding of CVRMed'95, pp. 282-286.