Adaptación de modelos de simulación estándar a modelos virtuales y/o sistemas de entrenamiento distribuidos, con representación del movimiento continúo de entidades.

Pau Fonseca i Casas
Departamento de Estadística e Investigación Operativa. Universidad Politécnica de Cataluña.
Barcelona, Catalunya, 08034, SPAIN

y

Josep Casanovas
Departamento de Estadística e Investigación Operativa. Universidad Politécnica de Cataluña.
Barcelona, Catalunya, 08034, SPAIN

y

Jordi Montero
Departamento de Estadística e Investigación Operativa. Universidad Politécnica de Cataluña.
Barcelona, Catalunya, 08034, SPAIN

RESUMEN

La representación de los modelos de simulación ha ido ganando importancia a medida que los modelos crecían en complejidad y era necesario dotar a los mismos de mayor potencia explicativa.
No obstante, con la complicación de la lógica que rige el comportamiento de los modelos de simulación, llegar a modelos mas refinados y realistas -y con una representación que también cada vez sea más realista- requiere cada vez mayor potencia de cálculo. Por otro lado el aumento del nivel de detalle en las interfaces de los sistemas de simulación, permite usar estos no solamente para la toma de decisiones sino también como herramientas de entrenamiento.
El sistema desarrollado por el LCFIB denominado LeanSim® [1], permite representar de forma distribuida sus modelos de simulación, y usar estas "ventanas" de representación como puntos de entrenamiento. A partir de la experiencia ganada durante el desarrollo del proyecto, se generalizó el uso del cliente de representación, dando lugar a la creación del sistema VRABox®, que permite el uso de clientes de representación en la mayoría de los simuladores, dotando a los mismos de la capacidad de representar el modelo en formato de realidad virtual y, siempre que sea posible, usar esta representación para interactuar con el modelo.

Palabras clave . Simulación, Realidad virtual, VRML, Sistema distribuido, Sistema de entrenamiento, tiempo real, LeanSim, VRABOX®.

1. INTRODUCCIÓN

Tradicionalmente, el principal objetivo de un proyecto de simulación es la obtención de datos que permitan comparar las diferentes alternativas planteadas. No obstante, cada vez más, los modelos de simulación se usan no solamente como una valiosa herramienta para comparar estas alternativas, sino también como instrumentos extremadamente útiles para la comprensión al detalle de los diferentes elementos que configuran el sistema.
Es en este punto, donde la representación de los modelos en formato de realidad virtual, así como la capacidad de entrenar a personal especializado a través del modelo de simulación deviene una característica muy interesante a tener en cuenta dentro de las características de los sistemas que permiten construir modelos de simulación.
La posibilidad de usar los modelos de simulación como sistemas de entrenamiento o representación, en formato de realidad virtual del modelo, tiene no obstante, una doble problemática.
El primer problema hace referencia a la mera representación virtual del modelo. Para ello se debe asociar, si puede ser de una forma simple y rápida, a cada elemento del modelo de simulación una representación virtual y permitir la modificación de la misma a partir de la lógica del simulador.
El segundo problema, y normalmente de mas difícil resolución en los sistemas de simulación genéricos, es dotar a la representación virtual de la capacidad de modificar el modelo de simulación a partir de la interacción de un usuario con el modelo virtual.
Actualmente, la mayoría del software de simulación no está pensado para ofrecer una respuesta satisfactoria a estas necesidades, la mayoría de los simuladores no están pensados para poder ofrecer una representación realista del modelo que implementan, y aún menos una representación distribuida, necesaria si se desea entrenar a más de un operario, o bien observar el modelo simultáneamente desde diferentes puntos de vista sin ralentizar la ejecución de la simulación.
No obstante, el elevado coste de un modelo de simulación, no solo a nivel del software que implementa el sistema en si, sino también el asociado a la construcción de los diferentes modelos que el sistema de ayuda a la decisión usa, hace muy interesante la posibilidad de adaptar antiguos modelos de forma que puedan recoger estas nuevas necesidades.
A continuación se explicará la arquitectura planteada para poder adaptar el software existente de simulación a las estas nuevas necesidades, así como se detallarán las características precisas

que este software ha de tener para que ello pueda realizarse con éxito.

2. EL OBSERVADOR REMOTO

El sistema de simulación LeanSim® desarrollado en el Laboratorio de cálculo de la Facultad de Informática de Barcelona, permite representar los modelos de simulación de forma distribuida a partir de un cliente de representación denominado LeanClient.

Este sistema, a partir de un conjunto de elementos que se pueden incorporar a cualquier modelo de LeanSim®, permiten comunicarse con el motor de simulación (denominado LeanGen) y de esta forma crear sistemas de entrenamiento (el sistema LeanClient con los componentes de comunicaciones hacia el motor de simulación se denomina LeanTraining).

Es importante notar que el sistema de representación del modelo (LeanClient) es un elemento remoto, y que puede ejecutarse en cualquier máquina que esté conectada mediante TCP/IP con la máquina que ejecuta el modelo.

Gracias a la representación distribuida, los modelos pueden ser complejos en sus dos vertientes, en su lógica interna, y también en cuanto a los detalles de su representación, sin que este segundo aspecto repercuta en la velocidad de la ejecución del modelo de simulación.

Los sistemas LeanClient, sin capacidad de interactuar con el modelo, permitirán implementar lo que se denomina puntos de observación (del modelo), mientras que a los sistemas LeanTraining permitirán implementar los puntos de interacción (con el modelo).

El rol del observador

La arquitectura planteada, implementada en el sistema VRABox®, permite la conexión simultánea de varios puntos de observación y/o de interacción con el motor de simulación, a través de los que operaran los usuarios.

Cada uno de estos usuarios tendrá un rol específico dentro del modelo de simulación, no solamente por el hecho de observar el modelo a partir de un punto de observación o de interacción, sino también por las particularidades que estos puntos pueden tener definidas.

Estas particularidades hacen que aparezca lo que denominamos el rol del observador. Este rol determina cuáles son las acciones que un observador puede realizar dentro del modelo, así como su ámbito de visibilidad.

Por ejemplo, si tenemos un sistema, como el mostrado en la Figura 1: Sistema de evaluación., formado por un par de máquinas controladas por sendos operarios, podríamos monitorizar el sistema a partir de 4 elementos.

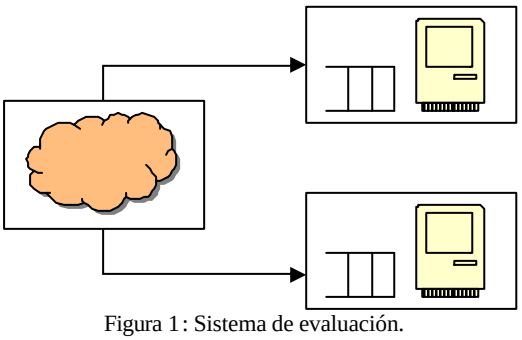

Figura 1: Sistema de evaluación.

El primer par de elementos pueden ser dos puntos de interacción que representarán las máquinas con las que los operarios trabajan. Estos dos puntos solamente pueden ver el entorno asociado a su máquina, no podrán ver nada más del modelo. Así mismo, sólo podrán interactuar con el modelo de simulación a partir de su máquina respectiva. Se puede definir otro punto de interacción que permita modificar parámetros de las dos máquinas, así como del sistema de generación de entidades que circularán hacia cada una de ellas. Este punto de interacción podría ser ocupado por una persona que quisiera evaluar el trabajo de los dos operarios a partir de modificar las condiciones del simulador. Finalmente se puede definir un punto de observación que permita a un responsable de la empresa seguir on-line la evaluación o entrenamiento de sus operarios, así como el trabajo del evaluador.

El rol que juega cada persona dentro del simulador se define en una base de datos que recoge los diferentes roles definidos para cada modelo.

3. ARQUITECTURA DE VRABOX®

A continuación se describirá brevemente la arquitectura planteada a través de su implementación concreta en VRABox®, y las condiciones que han de cumplir los simuladores para poder usar este sistema.

Es interesante notar que, a parte de los diferentes elementos que componen el sistema, existen una serie de ficheros que permiten representar adecuadamente los modelos.

Estos ficheros se estructuran en el directorio de instalación de la aplicación, y pueden ser de diferentes tipos (ficheros de AutoCad®, ficheros de imágenes gif, jpg, bmp, para decorados, ficheros con contenido multimedia para recrear animaciones, etc.).

4. ELEMENTOS

En la siguiente figura se muestran los elementos principales de VRABox®.

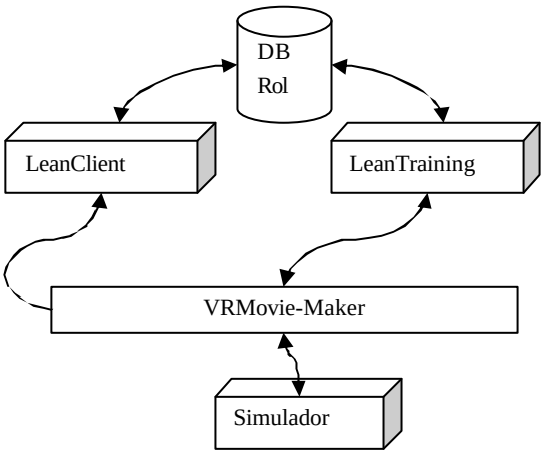

Figura 2: Elementos de VRABox®.

Estos elementos son:

1. El sistema de representación virtual (en nuestro caso LeanClient), que permitirá representar los diferentes puntos de interacción y de observación del modelo.

2. La base de datos, que permite la definición de los diferentes roles para los usuarios.
3. VRMovie-maker, que permite procesar los eventos que se crearán en el modelo de simulación y en los puntos de interacción a través de TCP/IP.
4. Elementos de interacción, que permiten que el usuario pueda, a través de ellos, interactuar con el modelo.
5. Plantillas del sistema de simulación que permitan gestionar el envío de información vía TCP/IP.

A continuación pasaremos a explicar brevemente los elementos más importantes.

5. LEANCLIENT

La apariencia del cliente de representación se puede observar en la siguiente figura.

Figura 3: LeanClient

La misión del cliente de representación es mostrar el modelo de simulación en formato de realidad virtual. Es por ello que cualquier cliente de representación tendrá dos elementos fundamentales: el motor de representación, que en nuestro caso se basa en un motor de VRML [3], y el sistema de comunicación vía TCP/IP.
No se entrará en mas detalles sobre el funcionamiento del cliente de representación, aunque puede consultarse [1].

6. VRMOVIE-MAKER: SISTEMA DE COMUNICACIÓN CON EL SIMULADOR.

El objetivo fundamental de este elemento es la comunicación entre los diferentes clientes de representación y el simulador.
La figura 4 se muestra la ventana inicial del sistema de comunicación VRMovie-Maker que se ha implementado.
Esta ventana se mostrará una vez que el simulador haya cargado el modelo de simulación que se desea representar.

Figura 4: Ventana inicial de VRMovie-Maker

Los parámetros iniciales configurables son el puerto de comunicaciones, el intervalo de refresco y el número máximo de clientes que se pueden conectar.
Una vez estos parámetros han sido definidos, es posible cargar o crear un nuevo modelo virtual para el modelo de simulación. Para ello VRMovie-Maker presenta el aspecto representado en la figura 5.

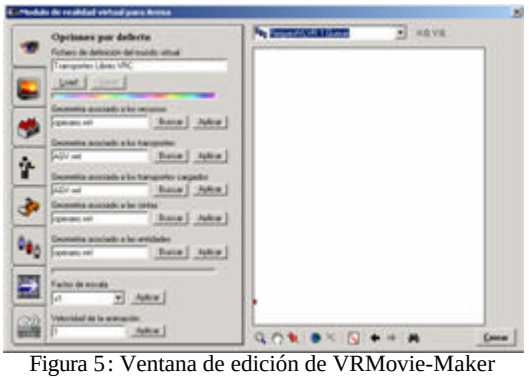

Figura 5: Ventana de edición de VRMovie-Maker

No se explicará el funcionamiento de VRMovie-Maker. Se destaca, únicamente, que la capa que representa necesitará adaptarse en función de la capacidad de definición de elementos de comunicación en el simulador. Gracias a este elemento, los diferentes clientes de representación no tendrán que modificarse, independientemente del simulador con el que se trabaje, y en el modelo de simulación simplemente deberán incorporarse unos pocos componentes que permitan enviar mensajes a este elemento, tal como se verá mas adelante en la sección 8 (plantillas).

7. ELEMENTOS DE INTERACCIÓN

Los diferentes elementos de interacción están escritos en VRML y Java [9]. Permiten que el motor de representación VRML del cliente capture las acciones del usuario, y envíe esta información al VRMovie-Maker (que hará llegar esta información al simulador).
Estos elementos de interacción poseen una parte de código común que no tiene que ser rescrita en función del proyecto, y que permite realizar las acciones mas comunes (apretar un mecanismo, arrastrar, pasar por encima, etc.). No obstante, es evidente que al estar muy vinculados a la representación física de los elementos que desean representar, es necesario modificar su estructura para cada proyecto.
Esta personalización es simplemente a nivel de representación, y únicamente involucra a los elementos representados que permiten ser manipulados por el usuario.

8. PLANTILLAS

En la mayoría de los sistemas de simulación, a no ser que permitan la comunicación con el exterior directamente vía TCP/IP, es necesario incluir elementos que nos habiliten esta posibilidad. Actualmente muchos de los sistemas de simulación permiten importar, o ejecutar código que puede realizar esta comunicación.
A partir de la escritura de este código, es necesario crear unas plantillas (siguiendo el formato de Arena®), o módulos (como se hace en Witness®), que al ser introducidos dentro del modelo de simulación no tengan ningún efecto sobre el mismo, pero que permitan enviar mensajes TCP/IP hacia el VRMovie-Maker, y gestionar los mensajes que el VRMovie-Maker pueda enviar.
Esta característica no esta siempre presente en los simuladores, y es por este motivo que no en todos será posible crear un sistema de entrenamiento.
A continuación se detallan las características a cumplir por un simulador.

9. CARACTERÍSTICAS NECESARIAS DE UN SISTEMA DE SIMULACIÓN PARA PODER USAR VRABOX®

Evidentemente no todos los simuladores existente en el mercado están preparados para poder comunicarse con VRMovie-Maker, y por tanto usar VRABox®.
Las características que ha de tener un simulador para poder usar VRABox® son:

1. **Capacidad de incluir código**: esta característica permite la ejecución de código, generalmente Visual Basic o Java que, de esta forma, permitirá establecer la comunicación con VRMovie-Maker.
2. **Capacidad de acceder al estado de diferentes elementos vía código**: Además de poder usar el código para poder generar la comunicación con VRMovie-Maker, es necesario también que se pueda acceder a las características de los diferentes elementos del modelo de simulación, como mínimo a aquellas que se quiere monitorizar.
3. **Capacidad de crear plantillas**: A partir de estas plantillas, se especificará el formato de las cadenas de texto que se enviarán a VRMovie-Maker, y se añadirán los nuevos elementos al modelo para, a partir de unas modificaciones que no afectarán en absoluto al comportamiento, poder enviar mensajes a VRMovie-Maker y que este pueda efectuar la representación. Esta característica no es necesaria, pero sí recomendable para facilitar la tarea de adaptación del modelo.

Si se cumplen estas características, será posible crear sistemas de entrenamiento de forma simple.
Para cada simulador será necesario especificar únicamente las plantillas que permitirán establecer la comunicación con VRMovie-Maker, pero el resto del sistema no deberá ser modificado en absoluto.
Si ninguna de estas características se cumple, es posible representar los modelos de forma distribuida a partir de una traza de la simulación [2]. Evidentemente será, no obstante, imposible crear sistemas de entrenamiento.

10. REPRESENTAR EL MOVIMIENTO CONTÍNUO

Debido a que las comunicaciones entre el motor de simulación y el cliente de representación se darán en una red, es importante minimizar el número de eventos necesarios para permitir la representación.
El principal problema reside en representar el movimiento en aquellos elementos que presentan movimiento continuo.
La primera solución que se puede plantear es el uso de eventos de representación. Con esta solución se escoge un determinado Δt, que indicará el intervalo de tiempo a partir del que se irán actualizando las representaciones en el cliente de representación.
El principal problema radica en que se generan una gran cantidad de eventos que no tienen nada que ver con el modelo de simulación, y que tendrán que ser transmitidos vía TCP/IP, lo que pude plantear una solución altamente ineficiente, teniendo en cuenta las características distribuidas del sistema.
Otra solución es la definición de rutas. Estas rutas especificarán los puntos por los que los diferentes elementos se podrán mover en cada instante de tiempo.
A nivel operativo, cuando el motor de simulación detecta la necesidad de crear una nueva representación para un elemento que tendrá un movimiento continuo, como por ejemplo situar un elemento al inicio de una cinta, envía no únicamente la posición actual al LeanClient (Or), sino que también envía la posición de destino (Dt) y la velocidad del movimiento (V). Es importante destacar que cada uno de esto elementos pertenece a un espacio métrico de dimensión 3, puesto que podemos movernos en las tres dimensiones.
Además del paso de estos tres parámetros por parte del simulador en el evento de inicio de representación del movimiento continuo, también debe existir una relación entre la velocidad del movimiento (v), y el paso de tiempo en el modelo de simulación. Es decir, es necesario que en la simulación se especifique un Δt fijo que indique como avanza el reloj de simulación (pero no el tratamiento de eventos, que al ser event scheduling no garantiza que el paso temporal entre eventos sea fijo [8]). Este Δt es necesario solamente para que la representación se adecue al modelo simulado, pero no provocará ningún envió extra de eventos a través de la red.
Hablar de un Δt constante para el reloj de la simulación es hablar de simulación a tiempo real, aunque evidentemente el factor multiplicativo del tiempo de simulación respecto al tiempo real puede ser de 2, 3 ó 0.5, lo que provocaría simulaciones dos o tres veces mas rápidas, o a la mitad de la velocidad de lo que sería el sistema real respectivamente.
En el sistema hay que especificar este factor de tiempo real (Rtf). VRABox® permite especificarlo cómodamente en una ventana como la que se muestra a continuación.

Figura 6: Velocidad de animación.

Con estos datos, LeanClient construye una nueva ruta, considerando un factor de escala temporal apropiado.
Pare representar el movimiento se calcula el tiempo necesario para que el elemento llegue al destino.
La distancia que se utiliza es la distancia euclidiana, por tanto la expresión usada es la mostrada en la siguiente ecuación.

$$t = \frac{\sqrt{(Or_x - Dt_x)^2 + (Or_y - Dt_y)^2 + (Or_z - Dt_z)^2}}{V * Rtf}$$

Ecuación 1: Cálculo del tiempo.

Como se ha comentado, este tiempo se usa para determinar cuánto tardará el elemento para llegar a su destino. El motor de representación (que usa VRML en nuestro caso) se encarga de realizar los cálculos necesarios para que esto sea así.
Generalizando, puede darse la situación en la que el movimiento continuo no se da únicamente entre dos puntos, sino que pasa por un conjunto de puntos intermedios antes de llegar a su destino final.
En ese caso se define una lista de la siguiente forma:

$$Anim1 = (\{p_1, t_1\}, \{p_2, t_2\}, .., \{p_i, t_i\}, .., \{p_{n-1}, t_{n-1}\}, \{p_n, t_n\}$$

Ecuación 2: pares de interpolación.

Esta lista representa los puntos de paso $p_1..p_n$ por los que tiene que circular el objeto. Cada uno de estos puntos tiene asociado un tiempo que indica el instante en el que el objeto ha de pasar.
Este método para representar elementos es comúnmente usado, por ejemplo, en VRML, pero en un entorno de simulación distribuido como VRABOX® permite reducir sensiblemente los eventos que se envían a través de la red.
Si la animación se ha de detener, se envía un evento de *stop* al LeanClient, y la animación entera es destruida. Posteriormente, cuando se reinicia la animación, ésta se reconstruye poniendo como primer elemento el último punto visitado del objeto, y como posteriores los que quedaban por visitar.

El código usado para representar el movimiento dentro de LeanClient tiene la estructura mostrada a continuación:

```
//Primeramente la creación del objeto a animar
CreateObject(Object,Tipus,false);
//ahora es necesario crear el interpolador de posiciones, que controla el movimiento y la demora en cada punto de la ruta.
CreateInterpPositions(Interpolator,Keys,KeysValues);
//Es necesario crear un reloj que controle el movimiento.
CreateTimer(Timmer,Time,"FALSE","0","0");
//Finalmente en VRML es necesario conectar los eventos que generan los diferentes objetos con los objetos que han de tratar esos eventos.
AddRoute("GLOBAL","time",Timmer,"set_startTime");
AddRoute(Timmer,"fraction_changed",Interpolator,"set_fraction");
AddRoute(Interpolator,"value_changed",Object,"set_translation");
```

Un detalle interesante es que en la creación del reloj que controla la duración de la animación, se le pasa como parámetro el tiempo total de la animación, calculado a partir de la ecuación 1.

```
CreateTimer(Timmer,Time,"FALSE","0","0");
```

El principal problema de esta aproximación es que, puede existir un desfase entre el motor de simulación y el cliente de representación, (LeanClient), es decir se podría estar mostrando el pasado del modelo de simulación. Por lo tanto, es posible que los objetos involucrados en movimientos continuos no tengan tiempo de acabar su movimiento.

Otro problema es que el evento que modificaría la posición de un objeto al finalizar su movimiento continuo se demore, con lo que habría objetos que se quedarían fijos al final de su animación.
Estos dos casos denotan que la red provoca demoras importantes en la transmisión de la información.
Lo normal, no obstante, es que las demoras en la transmisión de los paquetes de información no sean excesivamente grandes, y que permitan que la apariencia del modelo representado en el LeanClient se ajuste al modelo ejecutado en el simulador.

En la siguiente tabla se muestran los tiempos que se tarda en transmitir un paquete de información.

Trans-Atlantic Path		
NY	70	Londres
Wash	95	Frankfurt

Tabla 1: Tiempo en ms del recorrido a través del Atlántico

Trans-Pacific Path		
LA	101	Tokyo
LA	191	Singapur
SF	151	Sydney
SF	139	Hong Kong

Tabla 2: Tiempo en ms del recorrido a través del Pacifico

El tiempo medio de demora de un paquete de información es de aproximadamente 34 ms [14].

Obviamente la cantidad de eventos que fluirán a través de la red por unidad de tiempo es proporcional a Rtf, por lo tanto en redes más eficientes, sin tener en cuenta el tiempo necesario para efectuar la representación, se podrían representar modelos que se ejecutaran más rápido.

11. ESTUDIO DE SIMULADORES

A continuación se analizan algunos sistemas de simulación discreta, y se indica en qué nivel se podría trabajar con cada uno de ellos.

Simulador	Representación distribuida a partir de la traza	Representación distribuida en tiempo de ejecución	Capacidad de entrenar
Witness	Sí	No	No
Arena®	Sí	Sí	Sí
Simprocess	Sí	No	No
LeanSim	Sí	Sí	Sí

Tabla 3: Sistemas de simulación

12. EJEMPLO DE REPRESENTACIÓN PARA ARENA®

Se escogió Arena® porque permite incluir código VisualBasic, más la creación de plantillas y el acceso a determinadas características de sus objetos de simulación. Además Arena® carece de un motor de representación en formato de realidad virtual no basado en la traza, por lo que representaba un candidato adecuado para probar la arquitectura planteada.
Como se ha comentado, Arena® permite el uso de VisualBasic, pero en lugar de implementar las rutinas de comunicación de esta forma, lo que se hizo fue crear una librería dinámica que

permite realizar esta comunicación, simplificando enormemente el código necesario dentro de los diferentes elementos.
En Arena® se puede indicar el uso de librerías dinámicas dentro del modelo de simulación a partir de la ventana que se muestra en la siguiente figura.

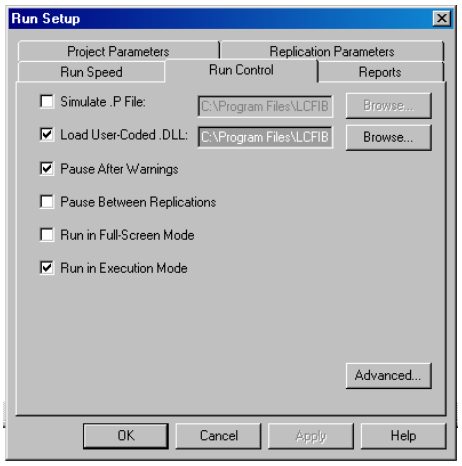

Figura 7: Carga de la DLL en Arena®.

Los elementos del modelo que permitirán enviar la información fuera del modelo serán los elementos TASK, que formatearán el texto cargándolo con la información que se desea enviar, mientras que el bloque ARRIB será el encargado de gestionar la recepción de mensajes.
En la siguiente figura se muestra un sencillo ejemplo en el que se indica el uso del bloque TASK y ARRIB.

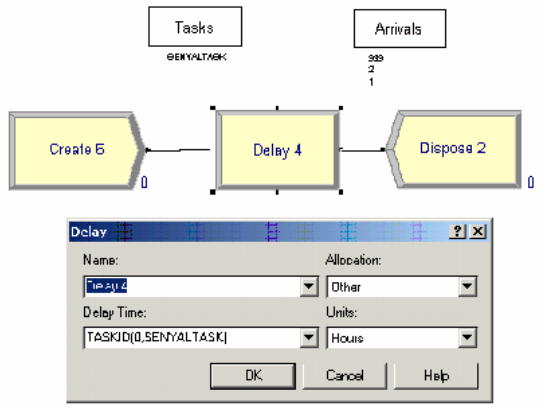

Figura 8: Ejemplo de uso de TASK y ARRIB en Arena®.

Así mismo, es necesario especificar el factor de velocidad de representación de la animación para poder obtener animaciones realistas. En la siguiente figura se muestra la ventana de Arena® que permite realizar esta operación.

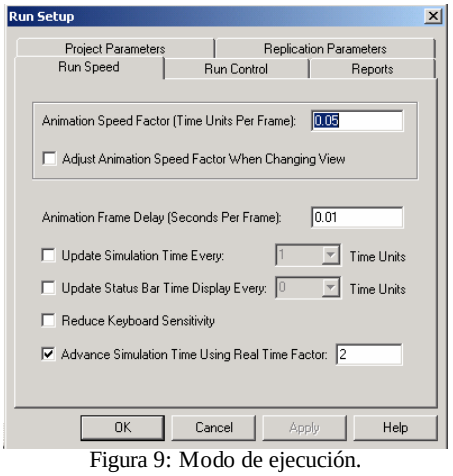

Figura 9: Modo de ejecución.

A partir de este punto, ya puede establecerse la comunicación con Arena.
Para ejecutar los modelos, simplemente debemos tener cargado el modelo de simulación de Arena®, y a partir de un LeanClient, que puede estar situado en cualquier otro PC conectado via TCP/IP con el PC que contiene el modelo, lanzar un evento de inicio de simulación. Este evento inicia la ejecución de la simulación en la máquina que tenga el modelo Arena®, e iniciará la captura de eventos de representación en el LeanClient para poder mostrar la evolución de la misma.

13. CONCLUSIONES

En el presente artículo se ha visto que siempre es posible crear representaciones virtuales distribuidas de modelos de simulación, creados a partir de simuladores estándar, pero no siempre será posible crear sistemas de entrenamiento a partir de estas representaciones.
Los sistemas de simulación más comunes no tienen capacidad para representar en formato de realidad virtual y de forma distribuida sus modelos. Esta deficiencia limita el tipo de aplicaciones para las que se pueden usar sus modelos, limitando enormemente su uso por ejemplo para sistemas de entrenamiento.
Con la metodología planteada aquí, y concretamente con el paquete VRABox® es posible dotar a cualquier simulador de la capacidad de representar los modelos de forma distribuida en formato de realidad virtual, permitiendo además a algunos simuladores la posibilidad de crear sistemas de entrenamiento.
Como se ha visto VRABox® reposa sobre una tecnología estándar de representación (VRML) lo que facilita enormemente la tarea al equipo que desarrolla los modelos virtuales, así como al equipo que implementa las diferentes partes del modelo de simulación.

14. REFERENCIAS

[1] Pau Fonseca i Casas, Josep Casanovas I García, Jordi Montero i García; LeanSim: Un sistema de simulación para el entrenamiento de personal especializado dentro de sistemas complejos; **Memorias de la 2ª Conferencia Iberoamericana en sistemas, cibernética e informática CISCI 2003, Volumen I**.

[2] Josep Casanovas i García, Wilfredo Pérez Ribero, Jordi Montero i García, Pau Fonseca i Casas; Simulación de recepción y expedición, y áreas de picking en una planta de producción farmacéutica; **6th IEEE International Conference on Emerging Technologies and Factory Automation ETFA 99**.

[3] Andrea L. Ames ; David R. Nadeau ; John L. Moreland 2000 **VRML 2.0 Sourcebook**, 2E. 688

[4] Microsoft Foundation Classes version 6.0 for Visual C++. **Microsoft Development Network. Visual Studio 6.0**.

[5] **Microsoft Development Network Library**. Visual C++ Section. MSDN.

[6] Antoni Guasch, Miquel Àngel Piera, Josep Casanovas, Jaime Figueres, **Modelado y simulación. Aplicación a procesos logísticos de fabricación y servicios**. Ediciones UPC, 2002

[7] Ulises Cortés García, Javier Béjar Alonso, Antonio Moreno Ribas **Inteligencia Artificial**, Ediciones UPC,1994

[8]. Law, A. M., Kelton, W. D. **Simulation modeling and analysis**. McGraw-Hill, 2000.

[9] Jaime Jaworski, **Java 1.2 al descubierto**. Editorial Prentice Hall, 1999

[10]. **The Direct X Programmer Reference**. Graphics and Multimedia Services. Microsoft Developer Network. 1998-1999.

[11] Geoffrey Gordon. **System simulation**; Editorial Prentice Hall 1978.

[12] Michael Luck, Peter McBurney, Chris Preist; **Agent Technology: Enabling Next Generation Computing**; AgentLink 2003

[13] Richard Murch, Tony Johnson; **Intelligent Software Agents**; Prentice Hall 1999.

15. DIRECCIONES DE INTERNET

[14] http://ipnetwork.bgtmo.ip.att.net/network_delay.html

[15] http://www.evl.uic.edu/pape/CAVE/