UN SIMULADOR DISTRIBUIDO PARA ENTRENAMIENTO DE OPERARIOS

Boroni Gustavo
gboroni@exa.unicen.du.ar

Vénere Marcelo
venerem@exa.unicen.edu.ar

ISISTAN-PLADEMA, Universidad Nacional del Centro, Tandil, Argentina
TEL: +54-2293-442202
FAX: +54-2293-442202

Resumen - En el presente trabajo se propone un sistema de realidad virtual orientado al entrenamiento simultaneo de varios operadores con la supervisión de un único instructor.
Para su desarrollo se utilizaron elementos de procesamiento distribuido y conceptos de tiempo real para la sincronización de los diferentes procesos. Se analizó especialmente el factor robustez del sistema, de forma que problemas en un proceso o equipo no afecte la performance del resto del sistema.
Se incluye la descripción de una implementación al caso específico del entrenamiento en navegación de barcos en base a información de un radar, cartografía y un equipo GPS.

Palabras clave - simulación distribuida, sincronización, procesos en paralelo, tiempo real.

I. Introducción

En los últimos años, la utilización de simuladores comenzó a tomar un gran auge en áreas vinculadas al entrenamiento de personal. La primera aplicación de esta idea fue en la década del '70 con los simuladores de vuelo utilizados para entrenar pilotos de aviones [2], pero hoy en día se encuentran herramientas para adiestramiento de todo tipo, tales como operadores de centrales industriales, grúas y maquinarias complejas, centrales nucleares y toda clase de embarcaciones (barcos, submarinos, etc.). La difusión de esta idea es hoy tan amplia que existen incluso casos en que para obtener la licencia el operador debe cumplir un cierto número de horas de trabajo en simulador. La ventaja es clara, resulta sumamente costoso y en algunos casos imposible entrenar a los operadores sobre los equipos verdaderos.

Esta metodología permite además la posibilidad de entrenar en forma simultanea a varios operadores y que los mismos sean monitoreados por un solo instructor. Este instructor podrá proponer escenarios, monitorear el desempeño y generar situaciones nuevas, e incluso grabar todo el ejercicio para una posterior reproducción y análisis.

Para que el entrenamiento sea efectivo los operarios deberán disponer de la funcionalidad equivalente y en lo posible sentirse en el ambiente real, para lo cual debe recurrirse a la simulación computacional de los procesos físicos intervinientes, y a elementos de realidad virtual, para que los mismos sean percibidos naturalmente y no como algo sintético y artificial.

Los sistemas de realidad virtual emplean distintas metodologías para conseguir el efecto deseado y a grandes rasgos podemos dividirlos en dos grandes grupos: sistemas inmersivos y sistemas no-inmersivos. Estos últimos son aquellos donde el monitor es la ventana hacia el mundo virtual y la

interacción se realiza por medio del teclado, micrófono, mouse o joystick. Resultan obviamente limitados aunque pueden resultar muy útiles para un entrenamiento inicial, pero nunca para licenciar operarios de equipos críticos.

Cuando el requerimiento de realismo es alto deben recurrirse a los sistemas inmersivos y dentro de estos podemos diferenciar dos categorías: sistemas puramente virtuales donde el usuario utiliza accesorios como visores o cascos, guantes, trajes especiales, o incluso habitaciones especialmente equipadas mediante cuatro pantallas en forma de cubo que rodean al observador, donde el usuario debe utilizar lentes y un dispositivo de seguimiento de movimientos de la cabeza; y por otro lado sistemas híbridos que utilizan hardware mientras resulta posible y software cuando no queda más remedio.

Por ejemplo, los simuladores de vuelo más completos entran en esta última categoría, ya que se instrumenta una verdadera cabina de avión con todos sus controles, y se reemplazan las ventanas por monitores. Otro caso similar es el que se presenta en este trabajo como aplicación, que es el entrenamiento de navegantes de grandes embarcaciones y se describe en mayor detalle en la sección VI.

En el presente trabajo analizamos las diferentes posibilidades de implementar un sistema de simulación y realidad virtual para entrenar operarios y proponemos lo que consideramos una arquitectura genérica adecuada para el desarrollo de este tipo de sistemas. Se incluye también una aplicación al caso particular de entrenamiento de navegantes de barcos en base a información radar.

II. Análisis de arquitecturas posibles

Como ya mencionamos, un sistema genérico de simulador para entrenamiento de operarios tendrá un puesto de instructor y varios puestos de entrenamiento que podrán estar separados físicamente. Esta estructura conduce naturalmente a la utilización de una arquitectura tipo cliente/servidor donde los puestos de entrenamiento serán vistos como uno o más clientes, mientras que el instructor podrá ser parte del servidor o un cliente más.

Se pueden considerar diferentes alternativas de implementación de esta arquitectura teniendo en cuenta donde se llevará a cabo el procesamiento del sistema y si el mismo estará distribuido o no entre los puestos. Las alternativas analizadas son las siguientes:

- *presentación distribuida de la interfaz.* Los puestos de entrenamientos sólo implementan la interfaz de los operarios, llevando toda la carga de procesamiento al servidor. Es decir, tanto los procesos comunes como cada uno de los procesos correspondientes a los operarios son ejecutados en el puesto del instructor;
- *presentación remota.* La implementación de un módulo intermedio sobre el cual se centraliza todo el procesamiento e información del sistema. El puesto del instructor es considerado un cliente más y sólo implementa la interfaz del usuario instructor;
- *procesamiento distribuido.* La desagregación de los procesos del modulo intermedio y su re-acomodamiento en cada uno de los puestos. Esto quiere decir que el procesamiento del sistema completo va a estar distribuido en procesos comunes sobre el puesto del instructor y procesos particulares sobre cada puesto de entrenamiento.

Para analizar estas alternativas se diseñaron e implementaron prototipos de las mismas. El procesamiento que llevan a cabo es una sencilla transferencia y visualización de mensajes. Se

consideraron parámetros de consistencia, performance en cuanto al tiempo de respuesta del sistema completo, mantenibilidad y tiempo asociado a la actualización de la interfaz. A partir de dichos parámetros se analizaron las ventajas y desventajas de cada prototipo cuyos resultados se resumen en la tabla I.

Prototipo	Ventajas	Desventajas
1. Presentación distribuida	1. Todos los puestos tienen información consistente porque sólo uno de estos se encarga de generarla y distribuirla. 2. Reduce la exigencia sobre los puestos de entrenamiento, que solo deberán encargarse de la interfaz con el usuario.	1. La sobrecarga de procesamiento en un equipo degrada la performance del sistema. 2. El sistema es poco mantenible al centralizar todo el procesamiento. 3. La actualización de la interfaz y la interacción que el usuario instructor realiza sobre la misma es demasiado lenta.
2. Presentación remota	Se mantienen las ventajas de la alternativa 1 y se mejora la actualización de la interfaz del instructor.	La carga sobre el equipo encargado del procesamiento sigue siendo alta, lo que limita su escalabilidad a mayores cantidades de puestos.
3. Procesamiento Distribuido	1. La consistencia en la información se logra porque sólo un puesto se encarga de coordinarla y distribuirla (no de generarla). 2. Mejor performance en la visualización de resultados sobre el instructor por la disminución de la carga de cálculo. 3. Más mantenible y modificable al distribuir la carga de procesamiento. 4. Fácilmente escalable a varios puestos.	1. Requiere que los equipos correspondientes a los operarios tengan la performance adecuada, ya que deberán encargarse del procesamiento de simulación además de la interfaz. 2. Requiere que estos equipos tengan performance similar para no degradar la respuesta global del sistema.

Tabla 1. Comparación de los prototipos implementados.

Claramente se observa que el tercer prototipo, esquematizado en la Figura 1 es el que permite obtener mejores resultados. En la próxima sección describimos de forma más detallada cada una de sus partes y como interactúan en conjunto.

III. Arquitectura propuesta del simulador para entrenamiento

El simulador para entrenamiento comprende varios puestos conectados por una LAN utilizando el protocolo de comunicación TCP/IP. Como se analizó en el punto anterior la opción que se presenta como más conveniente es la de procesamiento distribuido, pero aún dentro de este esquema existen distintas formas de aplicar el modelo cliente/servidor. La opción implementada considera clientes a los distintos **tipos** de procesos 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 sobre los que se ejecutan los mismos tres procesos clientes en cada uno, el servidor verá solamente tres clientes y no quince.

La Figura 1 muestra el esquema utilizado: el servidor estará en uno de los equipos del puesto instructor, y abrirá un puerto de comunicación por cada cliente habilitado; a su vez, un puesto de entrenamiento podrá estar formado por uno o más equipos sobre los que se ejecutan varios procesos, donde no todos son necesariamente tipo clientes. Por ejemplo sobre uno de los equipos de entrenamiento (equipo 3 del 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.

El proceso instructor va a actuar como centralizador y sincronizador de los datos generados por el sistema completo. Se busca que todos los puestos de entrenamiento tengan permanentemente los datos actualizados, que a partir de los mismos realicen sus respectivos cálculos, y que dada una base de tiempo programable, el puesto del instructor centralice toda la información del sistema y la distribuya a los distintos puestos. Estos puertos funcionarían en forma sincrónica con una dada base de tiempo, pero además, el instructor podrá enviar eventos temporales asincrónicos a los puestos de entrenamiento, ya sea en forma grupal o en forma individual.

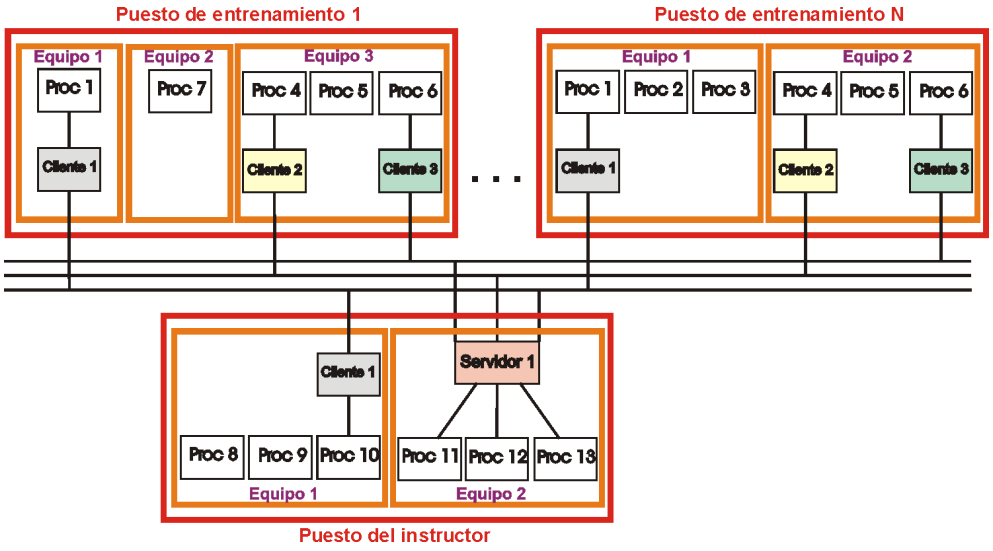

Figura 1. Esquema del prototipo base.

Para la comunicación de los procesos con la red se emplea un tipo de equipo lógico denominado “middleware”, que controla las conversaciones entre el proceso cliente y el proceso servidor (Figura 2). La interfaz que presenta es la estándar de los servicios de red y permite que los procesos "piensen" en todo momento que se están comunicando con una red.

IV. Sincronización de procesos

Como mencionamos anteriormente el procesamiento del sistema completo va a estar dividido en una serie de procesos distribuidos en diferentes equipos de una LAN, que además se ejecutarán en forma paralela. Estos procesos son asincrónicos ya que corren en diferentes equipos, simulan modelos matemáticos distintos y difieren en la interfaz que soportan entre otras cosas. Esto causa que para obtener un resultado en conjunto los procesos deben coordinarse o ser sincronizados por el administrador de la simulación, funcionalidad que hemos asociado al puesto del instructor. Cada uno de estos procesos se ejecuta según el tiempo local del sistema, el cual va a estar dado por las operaciones de cálculo que necesita realizar, la actualización de la interfaz y el tipo de equipamiento

en el que se encuentra procesando. Este tiempo local puede diferir o no con respecto al tiempo local de los restantes y por lo tanto si queremos que el sistema tenga todos sus datos consistentes vamos a necesitar un mecanismo o método que nos permita realizar una sincronización de los procesos involucrados.

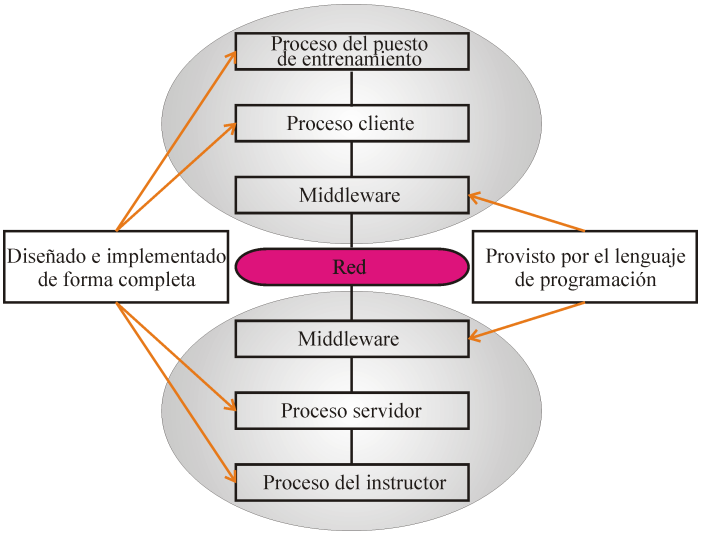

Figura 2. Procesos de comunicación para el Simulador Radar.

Utilizaremos el tiempo global de respuesta del sistema (GVT = *Global Virtual Time*) tal como se define por ejemplo en Martín et al [7], el cual representa el tiempo máximo en el cual todo el sistema se encuentra consistente con respecto a sus datos. Dado el requerimiento de tiempo real de este tipo de sistemas, nuestro objetivo será entonces minimizar GVT manteniéndolo por debajo de un cierto valor de diseño asociado al sistema que se desea implementar. Para ello se utilizó 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 del problema de sincronismo de los procesos del sistema podemos optar como solución 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 nuestro caso, el proceso del instructor permanecería ocioso hasta que llegue la información de todos los procesos asociados a los puestos de entrenamiento. En el método optimista el proceso receptor continuará con su ejecución aun 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 se debe resolver el problema de la consistencia de los datos al no cumplirse con alguna precondición. Una solución es la sugerida por Damani et al [3] que propone las siguientes 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*); en 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 última clasificación, *optimistic method + lazy cancellation*, es la que mejor se adapta como solución para nuestro caso y fue la estrategia adoptada.

V. Diseño de clases del puesto del instructor

Para la construcción del diseño de la aplicación se optó por el paradigma orientado a objetos ver [1] y [4] por ejemplo. Los motivos de esta decisión se basan fundamentalmente en las virtudes que éste ofrece, tales como la facilidad para su extensión y mantenimiento entre otras, y en la correspondencia inmediata entre los componentes del simulador y los objetos que podrían representarlos. La herramienta utilizada para desarrollar el simulador es Delphi 5.0.

A partir de la división de los procesos de comunicación y la separación entre la interfaz y el modelo de la aplicación, se diseñaron los siguientes paquetes de clases (*Package*):

- *Package SimulationComponents*: este paquete abarca las clases propias del modelo del simulador (clases asociadas a los componentes del ejercicio y la clase encargada de la administración de la simulación).
- *Package Observer*: involucra las clases asociadas a componentes visuales y al *pattern observer*, a través del cual vinculamos el modelo con la interfaz.
- *Package Middleware*: todas las clases asociadas al módulo *middleware* dentro de la arquitectura de comunicación.

A. Package SimulationComponents

"*BasicElement*": esta clase representa los distintos componentes que pueden aparecen en un simulador (un puesto de entrenamiento, un componente propio de un simulador pero que corre en el puesto del instructor, etc.). Dentro de sus métodos abstractos se encuentra *calculate(),* el cual se utilizará para obtener los datos actualizados de los componentes mencionados, en el caso de los puestos de entrenamiento serán los datos que los mismos envían, en el caso de un componente propio del instructor serán los datos que arroje el modelo matemático asociado.

"*TrainingPlace*": es subclase de "*BasicElement*" y su principal funcionalidad es la de mantener el vínculo entre el puesto del instructor en un puesto de entrenamiento.

"*ListBasicElement*": es la clase contenedora de "*BasicElement*". Se diseño básicamente para llevar control sobre listas de componentes que pueden aparecer (ejemplo: lista de puestos de entrenamiento). En caso de ser necesario las clases que hereden de la misma implementarán el método *calculate()* el cual se utilizará para determinar el comportamiento en forma conjunta de un tipo de componente.
"*SimulationManager*": es la clase responsable de la administración de la simulación. Su funcionalidad consiste en llevar el control del tiempo de la simulación, propagar las señales correspondientes al paso de tiempo al resto de los objetos del modelo para que calculen su nuevo estado, administrar los procesos asociados al envío-recepción de mensajes a través de la red y mantener actualizada la interfaz.

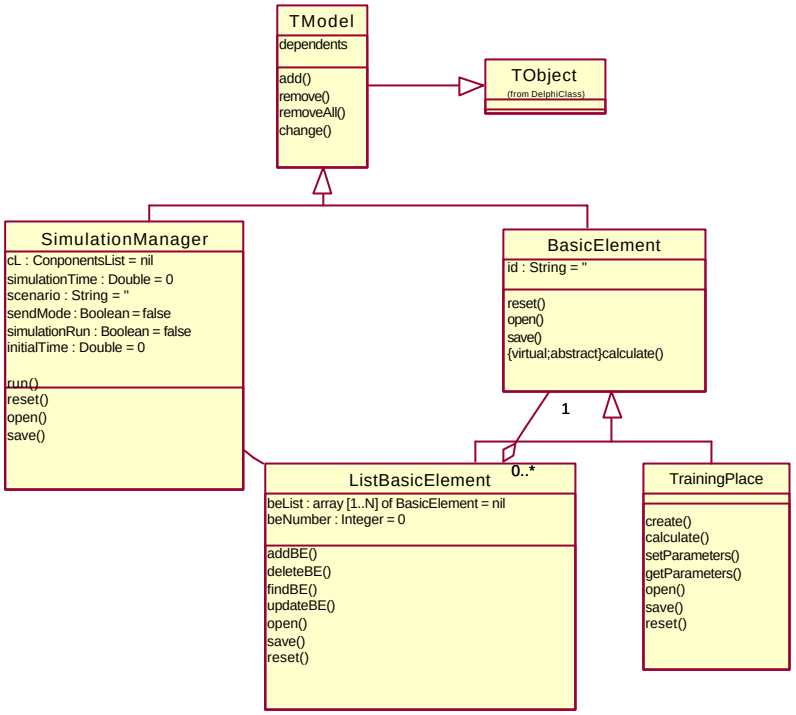

Figura 3. Diagrama de clases del Package SimulationComponents

B. Package Observer

La interfaz debe ser actualizada a medida que cambian los valores de las variables durante la simulación. Se decidió implementar el *"Pattern Observer"* para hacer eficiente este proceso [5]. La clase *"TObserver"* hereda de *"TObject"*, de manera que las instancias de *"TObserver"* tienen referencia al componente visual que deben modificar ante los cambios de su modelo asociado. A continuación se muestra el diagrama de clases junto con las clases *"TModel"* y *"TObserver"* asociadas al pattern.

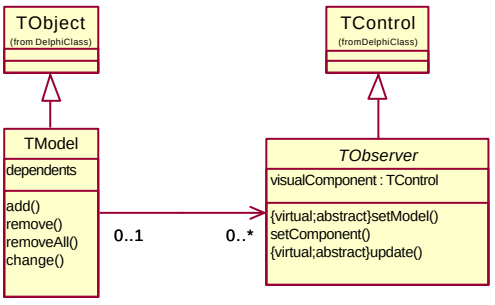

Figura 4. Diagrama de clases del Package Observer.

C. Package Middleware

La mayoría de las clases de este paquete es provista por el lenguaje de programación, en consecuencia sólo se desarrollaron las clases inferiores de la jerarquía asociadas a la arquitectura cliente/servidor: *ClientSocket* (proceso cliente) y *ServerSocket* (proceso servidor). Dentro de la funcionalidad que poseen se puede encontrar:

1. construir-descomponer los mensajes que se intercambiarán ambos procesos (en el cliente y en el servidor);
2. controlar la secuencia de envíos de mensajes (en el servidor);
3. detectar la salida del sistema de algún proceso (en el cliente y en el servidor).

Para la funcionalidad 3 en el caso del servidor, se implementó un conjunto de direcciones IP válidas (una dirección IP por cada módulo). Este conjunto se utiliza para determinar la cantidad de puestos activos y el número de conexiones abiertas entre el servidor y los clientes.

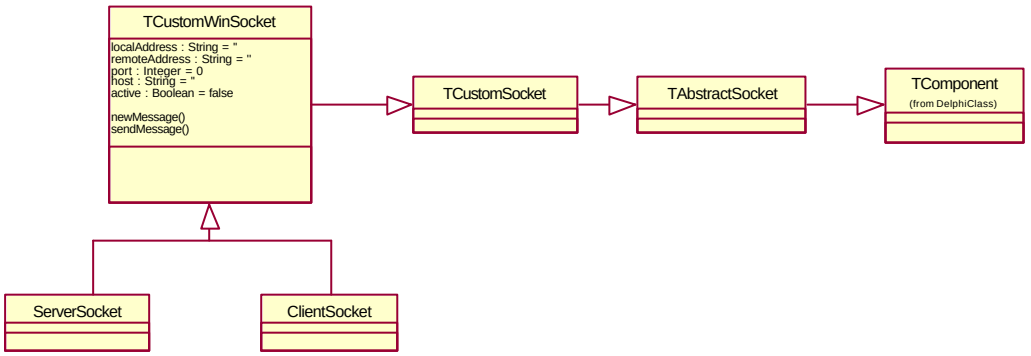

Figura 5. Diagrama de clases del Package Middleware.

VI. Aplicación al caso entrenamiento de navegantes

Mostramos a continuación una aplicación de la arquitectura presentada, al caso del entrenamiento de navegación por radar de pilotos de barcos mercantes. Esta operación se realiza desde un lugar del buque llamado puente de mando, lugar relativamente amplio que cuenta con una vista panorámica hacia la proa del navío (Figura 6). El navegante dispone además de uno o dos radares, cartografía en papel o un sistema computarizado de consulta cartográfica, una ecosonda para tener información de profundidad, un equipo GPS, girocompás y sistema de comunicaciones. Los controles del barco son básicamente el timón y el control de potencia de motores. Se dispone también de varios instrumentos adicionales que indican por ejemplo revoluciones por minuto de los motores, velocidad de desplazamiento, ritmo de caída, posición del timón y otras variables. Los mismos pueden ser tanto analógicos como digitales.

Por lo tanto la implementación de este sistema deberá considerar la simulación de todos estos componentes. Dado que el objetivo es el entrenamiento en navegación en base a imagen radar, no se incluye los procesos asociados a la simulación del puente visual. Esta situación es muy común, ya que es lo que ocurre en altamar, situaciones climáticas adversas o por la noche.

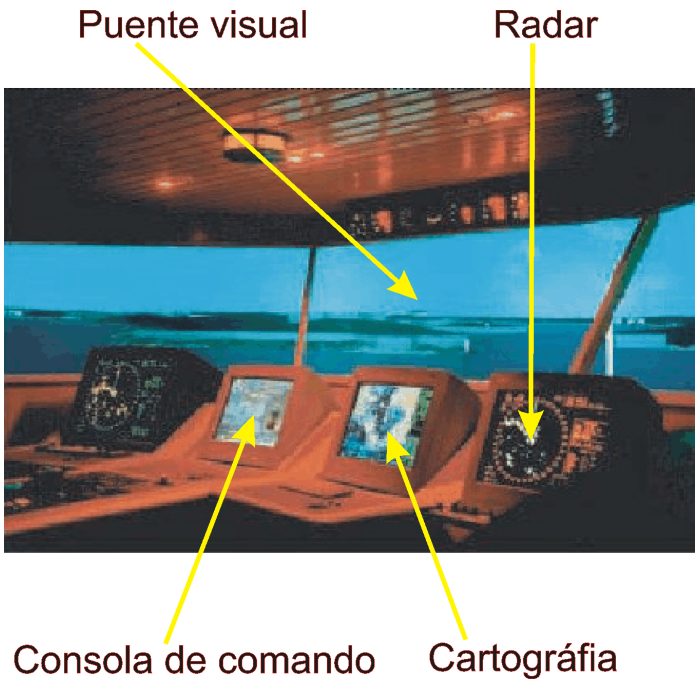

Figura 6. Puente de mando de un buque.

El radar es una herramienta capaz de generar una imagen del entorno en el cual se encuentra el buque, la que se genera a partir de la escucha de los ecos producidos por los obstáculos que lo rodean al ser iluminados por una onda emitida desde el mismo radar. La simulación del radar es relativamente compleja y costosa en términos computacionales. Los detalles del modelo implementado pueden verse Otheguy et al [8].

El simulador para entrenamiento de navegación por radar está entonces compuesto por un puesto "instructor" y como mínimo dos puestos de entrenamiento, denominados "buques propios" (la implementación actual cuenta con cuatro puestos). Los buques propios deben poseer los medios necesarios para que el cursante tenga la posibilidad de gobernar el buque, además de la pantalla de radar/ARPA, para efectuar los ejercicios de observación, punteo y navegación, mientras que el puesto del instructor debe disponer de las herramientas para crear escenarios y monitorear el ejercicio.

Las operaciones que el instructor puede realizar son:

- organizar un ejercicio seleccionando un escenario (mapa digitalizado de una carta de navegación) sobre el que navegarán los buques, agregando elementos tales como boyas, condiciones climáticas (viento, lluvias locales, etc.) e incluso generar tráfico colocando buques adicionales (blancos) con cursos pre-definidos o dirigidos interactivamente;
- monitorear un ejercicio y generar nuevas situaciones (que llamaremos eventos) a los distintos buques propios. Además, podrá visualizar en tiempo real el estado del radar de cualquier cursante, con el fin de determinar si está siendo correctamente utilizado;
- reproducir una simulación completa de un ejercicio con el fin de evaluar lo acontecido durante la ejecución del mismo.

El puesto de entrenamiento es el lugar que permite al cursante tener la sensación de estar en un "puente de mando real", el cual para este caso esta provisto de tres elementos básicos: módulo radar (consola radar), módulo comando (consola comando) y módulo cartografía (consola cartografía).

Físicamente el puesto de entrenamiento se representó mediante tres PCs, una por cada módulo que lo conforma, mientras que para el módulo instructor se utilizaron dos PCs, una para el aplicativo en si y la otra para poder visualizar el radar de un puesto de entrenamiento a elección.

Dada la diferente funcionalidad de los módulos asociados al puesto del instructor, el proceso cliente se categorizó en tres tipos diferentes de clientes tal como se indica en la Figura 7.

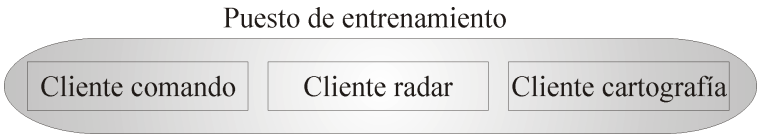

Figura 7. Esquema de clientes sobre un puesto de entrenamiento.

Para conectar y comunicar los módulos del simulador, se utilizaron los siguientes tipos de conexión:

- Punto a Punto entre la carta electrónica y la consola de comando, a través de un puerto serial.
- Punto a Punto entre el módulo instructor y el radar instructor, a través de la LAN.
- Punto a Punto entre el módulo instructor y los módulos comandos, utilizando la LAN.
- Punto a Punto entre el módulo instructor y los módulos radar, utilizando la LAN.

A. Diseño de clases del módulo instructor

Como ya mencionamos *"BasicElement"* representa los distintos componentes que aparecen en una simulación (buques propios, boyas, blancos y lluvias). Para representar dichos componentes se implementaron las subclases *"Boat"*, *"Buoy"*, *"Target"* y *"Rain"*. En este caso la información en común que poseen son la latitud, longitud, velocidad y dirección, la cual fue anexada a la clase "*BasicElement*".

Con respecto a los tipos de conexiones, sobre la clase *"Boat"* se mantienen los vínculos entre le módulo instructor y los módulos radar y comando, mientras que la vínculo entre el módulo instructor y el radar del instructor esta referenciado en la clase *"SimulationMananger"*.

B. Análisis de performance

Para determinar si el simulador se comporta como un sistema de tiempo real, analizamos la cantidad de instancias que se pueden llegar a generar, que tipo de operaciones matemáticas emplean y en base al equipamiento utilizado, como es el tiempo de respuesta del simulador. Para el módulo instructor se pueden llegar a simular 15 buques propios, 30 lluvias compuestas por 20 vértices, 30 blancos con un máximo de 20 vértices por trayectoria y 20 boyas, lo que hace un total de 155 instancias de *"BasicElement"*, número que tiene como tope máximo según la configuración del usuario. Para los módulos radar y comando el número de clases más relevantes es de aproximadamente 20, generándose una instancia por clase. Esto totaliza 175 instancias para el sistema completo con un sólo puesto de entrenamiento y 455 para el caso de 15 puestos. Debe tenerse en cuenta que la mayor parte de estas instancias involucran modelos y operaciones matemáticas complejas, algunas de las cuales son sumamente costosas. Dada la complejidad expuesta los tiempos de ejecución son muy aceptables, siendo que en equipos Pentium III de 700 MHz se logró mantener todo los componentes actualizados en alrededor de un segundo y medio, lo

cual esta muy por debajo de las especificaciones impuestas por la Armada Argentina para este tipo de simulador.

En cuanto a la robustez del sistema, la misma se logró detectando y aislando problemas en componentes individuales, tales como recepción de datos corruptos o no respuesta de un componente durante un cierto tiempo, y evitando que el resto del sistema se vea afectado por problemas en un componente.

VII. Conclusiones

A través de la implementación del caso particular del simulador para entrenamiento de navegación, se pudo comprobar que los principales requisitos (tiempo real, robustez, extensibilidad, etc.) que tiene este tipo de simuladores, se cumplieron satisfactoriamente.

El tiempo de respuesta del sistema en este caso particular se mantuvo por debajo de 1.5 segundos, que por tratarse de entrenamiento en navegación de barcos es totalmente aceptable, ya que en este lapso de tiempo el escenario presenta muy pocos cambios.

Actualmente el simulador esta siendo utilizado en la Escuela Náutica de la Armada Argentina para entrenamiento de personal. Hasta ahora la aplicación no ha requerido modificaciones importantes desde la fecha de entrega (octubre de 2001), a pesar de haber sido utilizado intensamente en el dictado de cursos.

Está prevista la extensión del sistema para incorporar un puente visual, un equipo sonar para la detección de peces y equipos para el manejo de redes de arrastre.

VIII. Bibliografía

[1] Booch G., Rumbaugh J. and Jacobson I. "*The Unified Modeling Language*". User Guide. Addison-Wesley, 1999.

[2] Club de Realidad Virtual, "*Cronología Realidad Virtual*". RV-UNAM. Enero de 1997. http://exodus.dcaa.unam.mx/virtual/history1.html

[3] Damani Om. P., Garg K., "*Fault-Tolerant Distributed Simulation*". Workshop on Parallel an Distributed Simulation, 1998.

[4] Jacobson I., Christeron, M. and Overgaard, G. "*Object-Oriented Software Engineering: A Use Case Driven Approach*". Reading: Addison-Wesley, 1992.

[5] Larman Cray. "*Applying UML and Patterns*". Prentice Hall PTR 1998.

[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 an Distributed Simulation, 1997.

[8] Otheguy I., Soriano M., Boroni G. y Vénere M. "Simulación en tiempo real de un radar de barrido horizontal", Proceedings of First South-American Congress on Computational Mechanics MECOM2002, (2002).