Instituto Politécnico Nacional

Centro de Investigación en Computación

Arquitectura para la construcción de mundos virtuales colaborativos basada en Java RMI y VRML

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A EL

ING. GUADALUPE MANUEL ESTRADA SEGOVIA

D I R E C T O R D E T E S I S

DR. ROLANDO QUINTERO TELLEZ

México D. F., febrero de 2008

Índice

Índice de tablas y figuras

Capítulo 2: Estado del Arte

Capítulo 3: Esquemas de implementación del modelo Alma-Cuerpo

Capítulo 4: Análisis y diseño de la arquitectura propuesta

Capítulo 5: Caso de Estudio

Resumen

La presente tesis presenta una implementación de la arquitectura denominada alma-cuerpo, la cual es diseñada para construir mundos virtuales distribuidos y de colaboración en Internet. El modelo alma-cuerpo describe la utilización de objetos distribuidos como representantes de los elementos que pueblan un mundo virtual, de manera que encapsulan su comportamiento y mantienen su estado en una copia única para todos los escenarios replicados; el uso de objetos distribuidos separa la construcción de un ambiente distribuido de los problemas de comunicación, y el desarrollo es natural al paradigma orientado a objetos.

En este trabajo se proponen, implementan y analizan diferentes modelos de sincronización y actualización que sirven como extensiones a la implementación del modelo alma-cuerpo orientados a mejorar su desempeño, las cuales van desde el esquema original que consiste en la actualización de los escenarios mediante escrutinio o el monitoreo iterativo del estado de los escenarios (polling), hasta implementaciones en las que los objetos del lado del servidor conocen a sus clientes e invocan sus métodos (callbacks), así como delegación del comportamiento predecible a los clientes.

Los modelos propuestos son implementados utilizando abstracciones computacionales bien definidas, tales como hilos, monitores y callbacks, lo que da la pauta para que puedan ser implementados en casi cualquiera de los lenguajes de programación actuales.

Del proceso de analizar cada uno de los modelos, parte fundamental en este trabajo, se presentan como resultado las diferentes pruebas y mediciones de desempeño realizadas, las cuales miden diferentes parámetros, tales como el procesamiento de la CPU, el tráfico en la red y el grado de consistencia entre los objetos que pueblan los escenarios.

La estrategia descrita da como resultado un esquema comparativo muy valioso para el diseño de mundos virtuales distribuidos que usen la arquitectura alma-cuerpo, ya que cada modelo resulta útil en función de las tareas que deban realizarse.

El uso de esta alternativa para construir aplicaciones de realidad virtual para colaboración resulta ser muy accesible en cuanto al costo, ya que está basado en su totalidad en el uso de tecnologías estándares y de libre distribución, como son la plataforma Java y el estándar para descripción de mundos virtuales en el Web: VRML.

Abstract

The present thesis introduces an implementation of the architecture named soul-body, which is designed to build distributed and collaborative virtual worlds over the Internet. The soul-body model describes the use of distributed objects as representations of the elements populating a virtual world, encapsulating their behavior and maintaining their state in an only copy for every replicated scenery; the use of distributed object separates the distributed environment building from the communication problems, and the development is natural to the object oriented paradigm.

In this work, different synchronization and actualization models are proposed, implemented, and analyzed, serving these as extensions to the soul-body model implementation and oriented to improve its performance. These extensions are such those from the original scheme consisting in the scenery actualization through scrutiny or polling, to implementations using callbacks, as well as delegation of behaviors to the clients for being these predictable.

The proposed models are implemented using well defined computer abstractions, such as threads, monitors and callbacks, so the models can be implemented in almost every programming language today.

A fundamental part of this work was the process of analyzing every proposed model; such analysis and the results of the different tests and development measurements made are presented here, measuring different parameters such as CPU processing, net traffic and consistency degrees between the objects populating the different sceneries.

The described strategy gives as result a very valuable comparative schema for the distributed virtual worlds design using the soul-body architecture, given that every model turns out very useful for the intended purpose given to it.

The use of this alternative to build collaborative virtual applications results very cost-effective, being totally based on standard and free distributed technologies, such as Java and the standard for virtual worlds description on the Web: VRML.

En este capitulo sirve de introducción y se definen los objetivos de este trabado de tesis, así como el alcance, la justificación y la estructura del documento.

Capítulo 1: Introducción

1.1 Introducción

En los últimos años, gracias a la investigación y al avance de tecnologías de comunicación, Internet se ha convertido no sólo en un medio para compartir información, sino para colaboración entre las personas que lo usan. Hoy en día existe una gran variedad de aplicaciones mediante las cuales usuarios en distintas partes del mundo pueden compartir información mediante ambientes de colaboración, utilizando cualquier tipo de interfaz gráfica para comunicarse con otros usuarios.

Gran parte del éxito de Internet se ha debido a la evolución de los navegadores y su capacidad de solicitar y desplegar información; en un principio sólo podía accederse a información estática e imágenes sin movimiento, en la actualidad es posible generar información de manera dinámica, acceder a la misma desde bases de datos heterogéneas, observar videos, animaciones e incluso navegar en ambientes tridimensionales, conocidos como Mundos Virtuales (MV).

Con los avances tecnológicos actuales se puede crear un entorno virtual en la Web y participar en él. Los participantes pueden adoptar diferentes formas e incluso puede asignárseles comportamientos, propiedades, accesorios o habilidades no naturales, así como también manipular el entorno para simular un ambiente desconocido. El lenguaje considerado popular para construir MV en Internet es VRML (*Virtual Reality Modeling Language*, Lenguaje de Modelado de Realidad Virtual), el cual permite definir un mundo o escena virtual con un lenguaje descriptivo y sencillo de utilizar. Actualmente, se liberó la especificación X3D la cual realiza mejoras sustanciales a VRML, inclusive revisándola ahora existen un conjunto de tecnologías que se han agregado a esta especificación, con soporte distribuido de cambios de estado. Sin embargo, aún no existe alguna implementación de referencia y mucho menos visualizador para esta especificación.

El comportamiento de una escena realizada en VRML es tal que el estado del MV es replicado en cada sitio donde existe una copia de la escena, con las desventajas que esto implica; los archivos de escenas virtuales son descargados a cada cliente como una “copia”, la cual es ejecutada en un *plugin* del navegador, por lo que cuando la escena es manipulada, los cambios son visibles sólo al cliente que descargó la escena.

En el presente trabajo de tesis se realiza la implementación de una arquitectura para la construcción de MV de múltiples usuarios (conocidos también como mundos virtuales distribuidos o compartidos) a través de la Web, dicha arquitectura se denomina “alma-cuerpo” [1] y está basada en su totalidad en el uso de tecnologías estándares y libres como VRML97 y la plataforma Java, aunque no son las únicas tecnologías a las que aplica ya que la arquitectura es una especificación para el modelado de mundos virtuales distribuidos.

Se proponen también diferentes extensiones a la arquitectura general, con el objetivo de incrementar el desempeño de las aplicaciones, proporcionar consistencia en la información que se le presenta al usuario (el estado del MV), y reducir el tráfico de red utilizado, disminuyendo así el tiempo de respuesta al usuario. Dichas extensiones son conocidas en este trabajo como esquemas de implementación, debido a que pueden ser tomadas como el molde para la construcción de entornos virtuales según las necesidades y condiciones requeridas.

La arquitectura usa la tecnología de objetos distribuidos de Java RMI como infraestructura de comunicación, evitando así utilizar protocolos propietarios y complejos, con la ventaja adicional de que el problema puede modelarse en su totalidad con el paradigma orientado a objetos, lo que permite identificar las abstracciones del mundo real para implementarlas en el MV, además de que utilizando objetos distribuidos se encapsula el estado de los objetos que pueblan el MV en un único objeto.

1.2 Planteamiento del problema

Dada la importancia del uso de Internet como medio para compartir información y más aún de intercambiarla, es necesario explotar las tecnologías de comunicación para el desarrollo de aplicaciones de colaboración entre múltiples usuarios separados geográficamente. La creación de ambientes virtuales permiten a las personas intercambiar información de manera natural, es decir, acercándose lo más posible a como se hace en la vida real, logrando así reducir gastos en transporte, aprovechar mejor el tiempo, seguridad en la integridad física, así como la educación y el entrenamiento a distancia.

La presente investigación muestra el diseño de una arquitectura para el desarrollo de mundos virtuales de colaboración a través de Internet, basada en tecnologías de comunicación estándares, particularmente de objetos remotos con Java RMI y de la utilización del lenguaje estándar para el modelado de realidad virtual en el Web: VRML.

La arquitectura propuesta, la cual se denomina “alma-cuerpo”, se compone por los objetos remotos (almas) que representan el estado único de los elementos del mundo virtual y el conjunto de réplicas del mundo que encapsulan la geometría (cuerpo) de cada uno de los elementos que lo forman. La comunicación entre las almas y cuerpos se logra mediante EAI (*External Authoring Interface*, Interfaz de Autoría Externa) de VRML.

Se analizará a detalle la arquitectura, considerando las ventajas y desventajas de la misma. Se proponen dos extensiones al diseño de la arquitectura con relación al procesamiento, consistencia y tipo de colaboración de cada uno de los elementos que habitan el mundo virtual.

Una vez mostrada la arquitectura con sus variantes, se desarrolla una aplicación para ilustrar la implementación de la misma y que sirve como caso de prueba para la obtención de parámetros de desempeño indispensable para evaluar el modelo propuesto y determinante para mejorar aplicaciones futuras. Tales parámetros corresponden al ancho de banda, velocidad de reacción y en general, todos aquellos que están involucrados directamente con el desempeño de la aplicación.

1.3 Objetivos

1.3.1 Objetivo General

Implementar una arquitectura basada en tecnologías estándares de objetos distribuidos y de realidad virtual para construir mundos virtuales distribuidos en la Web en una aplicación de estudio, diseñando e implementando para ello, diferentes modelos de sincronización y actualización de los elementos que pueblan el mundo virtual, de manera que puedan ser comparados midiendo diferentes parámetros como el desempeño y la consistencia.

1.3.2 Objetivos específicos

- Implementar la arquitectura denominada “alma-cuerpo” para construir Mundos Virtuales Distribuidos (MVD) a través del Web usando tecnologías estándares y de libre distribución como Java RMI y VRML.

- Diseñar e implementar una aplicación que sirva de caso de estudio para probar la implementación y los modelos propuestos.

- Proponer, diseñar e implementar diferentes modelos de sincronización y actualización de los MVD (como extensiones a la arquitectura original) para mejorar el desempeño de la aplicación del caso de estudio, basándose en abstracciones computacionales de concurrencia y sincronización bien definidas y orientadas a objetos como hilos, monitores y callbacks, usando la plataforma Java 2 para su desarrollo.

- Tomar mediciones para cada uno de los modelos propuestos, basándose en parámetros como uso del procesador, tráfico en la red y grado de consistencia en los escenarios participantes.

- Realizar un esquema comparativo con los resultados obtenidos al medir los diferentes modelos, de manera que sirva de base para la

toma de decisiones acerca de cuándo y cómo usar cada modelo, dependiendo de los requerimientos.

1.4 Justificación

En los últimos años se ha visto crecer Internet a un ritmo sin precedentes, esto por las cualidades que ofrece; cada día los adelantos tecnológicos permiten disfrutar de más y mejores recursos en el campo de la información, los cuales van siendo más y más accesibles al usuario común, aumentando con esto el número de usuarios de Internet, que a través de un navegador están en la posibilidad de compartir información y múltiples recursos, todo esto desde lugares distantes.

Más allá de compartir, los Mundos Virtuales Distribuidos permiten pensar en *colaborar*, ya que las aplicaciones que se vislumbran para los MVD podrán tener un gran impacto sobre muchas áreas de la vida diaria, particularmente en lo que se refiere a colaboración, investigación y educación a distancia.

Entre las áreas de estudio que involucra el desarrollo de esta tesis se encuentran las siguientes:

- *Colaboración en Internet*: La arquitectura propuesta permite el diseño, desarrollo e implementación de MVD en Internet con ventajas sobre los enfoques existentes actualmente.

- *Servicios de red*: Se realizan propuestas para mejorar el ancho de banda y latencia de la red, la heterogeneidad de componentes y la interacción distribuida.

- *Cómputo distribuido y paralelo*: en cuanto al diseño e ingeniería de sistemas de múltiples hilos y asíncronos.

- *Gráficas por computadora*: despliegue y renderizado de gráficos, así como diseño de interfaces tridimensionales.

- *Desarrollo de bases de datos*: haciendo uso de un servidor centralizado de datos, disponibles para todos los usuarios del sistema distribuido o local.

En cuanto a las ventajas que representa la arquitectura propuesta, pueden mencionarse las siguientes:

- *Colaboración a distancia*: La construcción de aplicaciones de colaboración de Realidad Virtual (RV) es importante para diversas áreas en las cuales los participantes no pueden estar reunidos físicamente, como medicina, ingeniería, diseño industrial, educación.

- *Uso de estándares*: La utilización de tecnologías estándares permite implementar de manera rápida y fácil aplicaciones de RV requiriendo únicamente del navegador de Internet para interactuar con los demás usuarios, sin tener que aprender lenguajes o extensiones propietarias y no estándares, asegurándose la portabilidad y compatibilidad entre plataformas distintas.

- *Fácil implementación*: La arquitectura "alma-cuerpo" propone una metodología para el desarrollo de MVD basada en los conceptos de orientación a objetos, manteniendo un único estado de todo el Ambiente Virtual Distribuido (AVD) y replicando sólo los escenarios en las computadoras de los clientes.

- *Factibilidad económica*: Gracias al uso de tecnologías estándares y no propietarias como Java y VRML y a programas de distribución gratuita, como el servidor Web Apache, Netscape Communicator y visualizadores de VRML como Cortona o CosmoPlayer, es posible crear aplicaciones colaborativas de RV en Internet sin tener que pagar licencias de software.

Desde el punto de vista comercial, puede considerarse la creación de aplicaciones para comercio electrónico, con atención personalizada tal como sucede en las tiendas departamentales; también podría pensarse en la capacitación individual o de grupo, por ejemplo, proporcionando entrenamiento a un grupo de obreros sobre cómo armar determinada pieza de ingeniería, realizar un diseño con la participación simultánea de los usuarios, etc., todo esto entre otras actividades más, hasta donde la creatividad y el uso de la tecnología de inmersión y manipulación lo permitan.

Adicionalmente a los puntos mencionados, este trabajo de tesis es parte de una investigación existente del CIC [1], implementando la arquitectura propuesta en un proyecto de investigación realizado previamente.

1.5 Estructura del documento

La presente tesis está organizada de la siguiente forma:

En el Capítulo 1 se describe la introducción al presente trabajo, así como los objetivos de la realización del mismo. La justificación y una breve descripción de lo que se encontrará a lo largo de los demás capítulos.

En el capítulo 2 se describen los conceptos fundamentales y antecedentes necesarios para entender los alcances que enmarcan la presente investigación. Se hace una revisión de los trabajos relacionados con la construcción de Mundos Virtuales Distribuidos en la Web, estableciendo el estado del arte de la problemática actual de las arquitecturas utilizadas, así como de otras

tecnologías alternativas utilizadas para el despliegue de aplicaciones gráficas en el Web.

La mayor parte de esta investigación está cimentada en la propuesta del modelo alma-cuerpo, cuya implementación se detalla en el Capítulo 3 y se proponen diferentes esquemas para resolver problemas específicos de desempeño, consistencia y velocidad de procesamiento; se profundiza en detalles acerca de la implementación de las diferentes arquitecturas propuestas.

En el Capítulo 4 se describe la infraestructura tecnológica necesaria para el desarrollo del presente trabajo de tesis y se mencionan las características que destacaron para escogerlas como marco de trabajo. Igualmente se tratan los detalles acerca de la arquitectura y el diseño.

El capitulo 5 muestra la implementación de cada modelo propuesto en un caso de estudio, partiendo desde la definición de los casos de uso hasta la implementación de los mismos. Se exponen las pruebas y resultados obtenidos en cada una de las implementaciones permitiendo mostrar un esquema comparativo, ilustrando las ventajas y desventajas de cada enfoque.

En el capítulo 7 se presentan las conclusiones generales, los logros alcanzados y algunas propuestas de la cuales pueden derivarse otras líneas de investigación.

“Todas las verdades son fáciles de entender una vez que son descubiertas; el punto es descubrirlas”
Galileo Galilei

Este capítulo contiene los antecedentes y definiciones necesarias para la comprensión del trabajo de tesis, así como los proyectos relacionados.

Capítulo 2: Estado del Arte

2.1 Antecedentes

En 1965 se dijo que "*el último despliegue de cómputo lo realizarían imágenes, con apariencia real, sonido real y objetos con comportamiento real"* [2]. Estas fueron las bases para cimentar la línea de investigación hoy conocida como **Realidad Virtual** (RV), que es una combinación de diversas tecnologías e interfaces que permiten a uno o más usuarios interactuar en tiempo real con un entorno dinámico tridimensional generado por computadora, mejor conocido como "Mundo Virtual" (MV).

Dado que esta investigación se basa en la utilización de mundos virtuales, se comienza con hacer una caracterización de ellos.

2.2 Mundos Virtuales

Un Mundo Virtual es un sistema de cómputo para la simulación de objetos reales o imaginarios, consistente en un escenario tridimensional habitable por entidades u objetos donde el usuario tiene una participación.

Llos MV poseen ciertas características entre las que se pueden mencionar:

- *Tridimensionalidad*: Significa que los MV muestran al usuario un entorno creado usando las tres dimensiones del mundo real, ubicando los objetos por el espacio que ocupan.

- *Navegación*: Es la habilidad del usuario para moverse dentro del MV de acuerdo con las restricciones que los diseñadores hayan impuesto. Aunque por lo general el usuario no se da cuenta de dichas restricciones.

- *Interacción*: Permite que los usuarios manipulen el curso de las acciones dentro de los MV, de manera que el sistema responde a lo que el usuario acciona. La interacción *dinámica del* ambiente consiste en definir un conjunto de reglas de cómo, cuándo y dónde, los componentes del sistema interactuarán con el usuario para intercambiar información.

- *Inmersión*: Es la característica mediante la cual el usuario es "absorbido" fuera del mundo real y sólo percibe la "realidad" del MV. Se le conoce como Realidad Virtual Inmersiva, lo que no sucede en la denominada Realidad Virtual de Escritorio donde el usuario está consiente de que a través de su computadora puede interactuar con el ambiente virtual sin sentirse "inmerso" dentro del ambiente.

- *Estado*: El estado de un MV en un tiempo determinado se entiende por las características en ese momento de todos y cada uno de las entidades que componen el MV.

La *interacción* y el *estado* son las características claves en las que se enfoca esta investigación, pues la interacción define el momento de cambiar el estado de las entidades que pueblan el mundo virtual.

Los mundos virtuales tienen aplicaciones en el diseño, simulaciones (de vuelo, comportamiento espacial en sistemas planetarios, clima, etc.), educación, siendo ésta una de las áreas más importantes, ya que la RV aplicada al proceso de enseñanza-aprendizaje, mejora en gran medida cuando los estudiantes pueden comprender las cosas estudiándolas a detalle en cualquier área del conocimiento; el geoprocesamiento es otra área de aplicación importante, ya que se pueden identificar y analizar diferentes formas geológicas y geográficas, permitiendo a los investigadores visualizar y compartir opiniones acerca del análisis y comportamiento de la información representada.

Las aplicaciones que utilizan MV, dado su costo computacional, tienden a ser diseñadas, construidas y puestas en producción en máquinas cuyo precio es superior a los estándares del mercado debido a una necesidad extra de memoria y velocidad de procesamiento. En los años recientes el incremento en el uso de la Internet ha llevado a buscar alternativas para presentar MV con el menor costo posible en cuanto a su creación, navegación e interacción, por ello se crearon tecnologías de conectores (o plugins) para despliegue de MV.

Existen diferentes herramientas para crear MV, entre las más conocidas se encuentran 3D Studio y Autocad de Autodesk®, así como también las API (*Application Programming Interfaces*, Interfaces de Programación de Aplicaciones) para construir escenas tridimensionales (como Java 3D y OpenGL); además de lenguajes de marcado para su descripción (tales como OpenInventor de Silicon Graphics® y VRML (*Virtual Reality Modeling Language*, Lenguaje de Modelado de Realidad Virtual).

Sin embargo, con las características mencionadas anteriormente solo se tienen Mundos Virtuales que pueden ser visitados por diferentes usuarios, pero cada quien con una copia de la escena desplegada en su propia computadora, de manera que no pueden verse entre ellos ni ver los elementos que cada uno hubiera modificado, es decir, no pueden interactuar o mejor aún colaborar entre sí. La colaboración se da cuando los usuarios interactúan para lograr un objetivo en común.

2.3 Ambientes Virtuales Distribuidos

Un Ambiente Virtual Distribuido (AVD) es un sistema de software en el cuál múltiples usuarios interactúan unos con otros en tiempo-real, aún si los usuarios se encuentran localizados alrededor del mundo. También es conocido como Mundo Virtual Distribuido (MVD)[3]. Dicha interacción se vuelve colaboración cuando los usuarios la utilizan para lograr un objetivo en común.

Los usuarios acceden al sistema mediante sus computadoras, las cuales les proporcionan una interfaz para interactuar con el ambiente virtual. Por ejemplo, supóngase que un usuario entra a una galería virtual de arte y desea información de algún objeto; es posible entonces que el ambiente genere una representación del usuario con determinada forma, los movimientos que el usuario realice se verán reflejados en su representación en el ambiente; más allá de eso, podría encontrarse en su recorrido por el MV con otras formas, las cuales podrían ser representaciones de otros usuarios conectados al mismo tiempo, pero en diferentes localidades. Cada usuario vería el ambiente según su perspectiva de navegación, ya que generalmente los ambientes virtuales pueden simular lo que el usuario puede ver estableciendo ciertos límites acordes con las reglas de visión humanas [22] (véase la Figura 2.1).

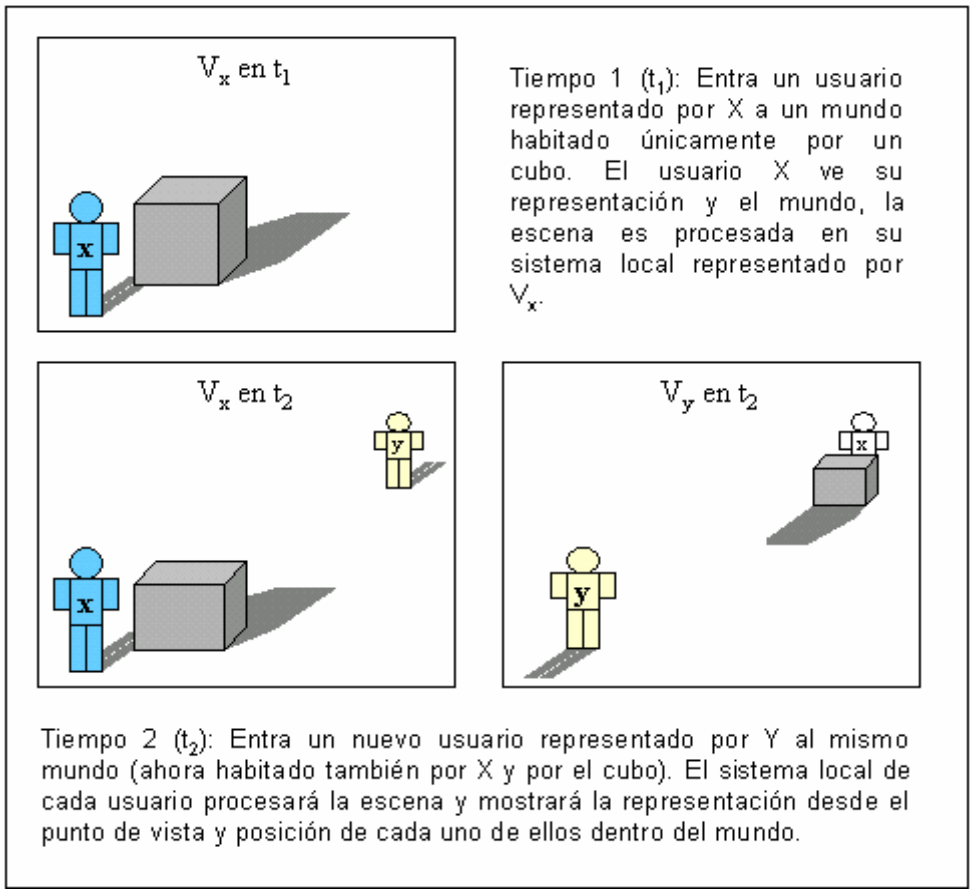


Figura 2.1 Representación de la escena de dos usuarios desde sus sistemas locales dentro un mundo virtual distribuido

Como puede verse, la utilización de este tipo de ambientes en el futuro podría abarcar desde realizar visitas virtuales a centros comerciales y salas de exhibición, así como asistir a conferencias y exposiciones profesionales, dar soporte a clientes, ocupando un lugar relevante el aprendizaje y educación a distancia.

Uno de los mayores retos del desarrollo de ambientes virtuales distribuidos es que se trata de una actividad interdisciplinaria, ya que se necesitan conocimientos de diferentes áreas, tales como:

- Protocolos de red
- Computación distribuida y paralela
- Gráficos por computadora
- Programación concurrente
- Bases de datos
- Interfaces tridimensionales

Actualmente existe un proyecto de Mundos Virtuales Distribuido denominado SecondLife [17] al cual se accede a través de una aplicación cliente que se descarga de Internet y que permite interactuar con usuarios y servicios en diferentes partes del mundo.

2.3.1 Características de los Ambientes Virtuales Distribuidos

Un Ambiente Virtual Distribuido cumple con cinco características, a saber:

- *Una percepción compartida de espacio*: Los usuarios tienen la ilusión de hallarse en el mismo lugar, por ejemplo en un mismo edificio o terreno.

- *Una percepción compartida de presencia*: Al estar en un lugar compartido cada usuario emplea un *avatar,* el cual tiene una representación gráfica en el ambiente como un modelo estructural del cuerpo humano, animal o alguna otra forma; cuando los usuarios dejan el ambiente los avatares desaparecen y eso deben percibirlo los demás usuarios en el sistema.

- *Una percepción compartida del tiempo*: Percibir los comportamientos de los demás usuarios “en el momento que ocurren”, es decir, tienen una noción consistente del tiempo.

- *Una forma de comunicación entre usuarios*: Movimientos, gestos, voz, o texto escrito.

- *Una forma de participar*: Los usuarios interactúan entre ellos y con los objetos del ambiente; pueden seleccionarlos, manipularlos, dárselos a otros usuarios e incluso destruirlos, de acuerdo a las reglas del AVD.

2.3.2 Componentes de los Ambientes Virtuales Distribuidos

Para cumplir con las características de percepción y participación, los sistemas de AVD deben componerse de cuatro elementos indispensables:

- *Infraestructura y motores de despliegue de gráficos*; donde los usuarios perciben el ambiente virtual: cascos, monitores con tarjetas aceleradoras de gráficos, estándares de despliegue de gráficos (como el API de OpenGL), visualizadores de estándares para el Web (VRML), procesadores, consolas de juegos, etc.

- *Dispositivos de control e interacción*, los cuales se utilizan para establecer comunicación con otros usuarios, manipular los elementos virtuales dentro del ambiente y navegar dentro del mismo; entre los principales se encuentran el teclado, ratón, guantes de RV, palancas de juego, etc.

- *Sistemas de procesamiento de información*, son los encargados de recibir la información generada por eventos que ocurren en los dispositivos de control de los participantes para determinar la localización del usuario y los demás objetos; determinan también cómo y cuándo notificar a los demás participantes de esos cambios.

- *Redes de comunicación de datos*, que serán utilizadas para sincronizar el estado compartido del AVD, intercambiando información. Para una mayor eficiencia en este tipo de ambientes, son necesarias redes con gran ancho de banda, de uso dedicado y con calidad de servicios.

2.3.3 Problemas al desarrollar Ambientes Virtuales Distribuidos

El desarrollo de un Ambiente Virtual Distribuido (AVD) resulta complejo, ya que son varios factores los que están involucrados en su diseño e implementación; los AVD son tanto sistemas distribuidos como aplicaciones gráficas interactivas.

Entre las tareas de los elementos que componen un AVD se encuentran la administración de la red, el manejo de pérdida de datos, tolerancia a fallas, transacciones, concurrencia, despliegue de gráficos, procesamiento de datos en tiempo real (que los usuarios introducen), entre otras. Además, estos elementos necesitan trabajar con otros servicios o aplicaciones, lo que hace

aún más complejo su desarrollo. Por ejemplo, el uso de Sistemas Manejadores de Bases de Datos para hacer persistente el estado del AVD, servicios de autenticación de usuarios, etc. Otra tarea compleja sucede al almacenar los eventos que ocurren en tiempo real, ya que el estado completo del AVD puede estar en diferentes máquinas del sistema.

Entre los principales factores que pueden afectar el desarrollo y ejecución de un AVD están [3][4]:

a) Ancho de banda y latencia de la red.

Los sistemas de AVD acceden constantemente a la red para intercambiar información relacionada con el estado actual de cada participante. Mientras mayor sea el nivel de detalle que los usuarios reciban acerca de las actividades de los demás usuarios, mayor información deberá ser enviada a través de la red, lo que provoca un aumento del tráfico.

En Internet el ancho de banda, además de ser generalmente bajo, no está garantizado, debido a la semántica de mejor esfuerzo que implementa el protocolo IP. Los retardos en la red son difíciles de manejar cuando hay muchos usuarios, por lo que hay que asegurar que la información presentada cumpla con un orden causal. Este es uno de los puntos más importantes en el cual la presente tesis se apoya para mejorar el tráfico en la red y por ende la consistencia de las escenas distribuidas.

b) Heterogeneidad.

El uso de sistemas heterogéneos hace que, debido a la naturaleza distribuida del ambiente, se tenga que lidiar con diferentes plataformas de hardware y software, como sistemas operativos, componentes de hardware y protocolos de red, lo que provoca los siguientes inconvenientes:

- A mayor heterogeneidad, menor desempeño, pues podría darse el caso de que diferentes usuarios conectados usen redes de diferentes capacidades de transmisión, por lo que algunos podrían recibir menos información que otros, degradando el desempeño. Una alternativa es diseñar el ambiente de manera que use un “mínimo común denominador” donde se garantice la misma capacidad de transmisión y recepción para todos.

- Que algunos usuarios tomarán ventaja sobre otros, pues tienen mejores equipos de despliegue de gráficos, de procesamiento, de red, etc.

- Los sistemas operativos difieran en los lenguajes y bibliotecas que proporcionan. Una solución a este problema es usar lenguajes independientes de la plataforma como VRML y Java, o arquitecturas

como CORBA, la cual consigue independencia de plataforma a través de protocolos estandarizados.

c) Interacción distribuida

Ésta proporciona la calidad en el AVD, ya que logra que el usuario tenga la ilusión de que el ambiente entero está en una sola computadora y que sus acciones tienen un impacto inmediato en el ambiente [22]; es por ello que cada computadora debe intentar presentar una vista consistente en el tiempo.

d) Tolerancia a fallas

Los AVD intrínsicamente deben lidiar con la posibilidad de que una o más de las computadoras conectadas puedan fallar.

El manejo de fallas cae en las siguientes categorías:

- *Paro total del sistema*: Cuando el AVD deja completamente de funcionar, por ejemplo, el uso de una arquitectura de servidor central para recibir y distribuir todos los datos, fallando éste.

- *Cierre del sistema*: Podría no impactar a los usuarios del ambiente, pero se impide que nuevos usuarios entren al sistema, por ejemplo si falla el servicio de autenticación.

- *Dificultad del sistema*: Degradan el desempeño del AVD; por ejemplo, si falla la computadora de un usuario X, los demás deben notar que su representación desaparece del sistema.

- *Continuidad del sistema*: Si ocurre alguna falla no debe tener efecto sobre el AVD; por ejemplo, si falla el servicio de autentificación.

e) Escalabilidad

Un sistema distribuido es escalable si trabaja con un gran número de clientes y objetos. Para conseguir esto, el trabajo debe ser distribuido de la misma forma a todos los clientes y no debe haber cuellos de botella en la estructura de la comunicación.

f) Despliegue y configuración

Si el software de una aplicación cliente de un AVD es grande y monolítico, inapropiado para descargarse, debe ser diseñado en torno a una biblioteca pequeña, como núcleo y componentes que puedan ser dinámicamente descargados dependiendo de las necesidades de ejecución.

Si el despliegue de la escena se realiza dentro de un navegador de Internet es necesario garantizar que:

- El ambiente sea descargable fácilmente.
- La implementación del AVD cumpla con las restricciones de seguridad impuestos por el entorno del navegador.
- El software se ejecute y despliegue correctamente a través de las diferentes plataformas de los navegadores.

Dados todos estos factores, en la presente tesis se proponen e implementan modelos para su optimización, como se ilustra en la Tabla 2.1:

Factor de optimización	¿Se trata en esta tesis?
Ancho de banda y latencia de la red	☑
Heterogeneidad	☑
Interacción distribuida	☑
Tolerancia a fallas	✖
Escalabilidad	✖[1]
Despliegue y configuración	☑

Tabla 2.1 Factores de optimización considerados en el desarrollo de la tesis

2.4 Objetos distribuidos

Desde el surgimiento de las redes de datos y comunicaciones, las aplicaciones comenzaron a comunicarse entre sí usando diferentes mecanismos, los cuales han ido evolucionando junto con los lenguajes de programación que los soportan.

La programación distribuida es importante pues permite reducir los costos de desarrollo de software, realizar un balanceo de cargas de los recursos e independencia entre plataformas, además de que se pueden aprovechar los recursos de ciertas computadoras para realizar tareas complejas que en las computadoras de los clientes tomarían mucho más tiempo.

La programación distribuida consiste básicamente en procesos de cómputo en diferentes espacios de memoria que se comunican entre sí por mecanismos tradicionales de comunicación, tales como sockets, RPC (*Remote Procedure Call*, Invocación a Procedimientos Remotos) y colas; enviándose mensajes ya sea de manera asíncrona o síncrona.

Llevar a cabo la abstracción de este proceso de comunicación hacia un mecanismo tan natural como lo es la encapsulación del comportamiento de un objeto, es la filosofía de las arquitecturas de objetos distribuidos.

Existen diferentes modelos propuestos que especifican arquitecturas de objetos distribuidos, siendo los más importantes:

[1] Estos temas se trataron en la tesis del M. en C. Chadwick Carreto Arellano, graduado por el Centro de Investigación en Computación, del IPN [22].

- Microsoft DCOM (*Distributed Object Component Model*).
- CORBA (*Common Object Request Broker Architecture*) de OMG (Object Management Group).
- Java RMI de Sun Microsystems.
- JEE (*Java Enterprise Edition*)
- Microsoft .NET

El uso de tecnologías de objetos distribuidos hace posible la programación de aplicaciones que conservan la semántica de invocación de métodos de objetos locales, permitiendo que el programador se olvide de los detalles de la comunicación, esto es, sin tener que escribir los protocolos para el manejo de la información transmitida.

Con los objetos remotos, el cliente invoca métodos de un objeto local. Los objetos son representados por sus referencias[2], las cuales pasan la solicitud de invocación a las implementaciones de los objetos sobre otros sistemas.

Por consiguiente, en el modelo de objetos remotos, los programadores tratan estrictamente con objetos: "las invocaciones a los métodos son siempre hechas a través de un objeto que lo representa". Idealmente, los programadores tratan cada objeto como si fueran locales. El espacio de direcciones actual y la localización del objeto es irrelevante. El programador trata con las invocaciones a objetos a través de un conjunto bien definido de interfaces y cada invocación al método, luce como una llamada a través de la representación del objeto local.

De cierta manera, los objetos remotos son a los lenguajes orientados a objetos, lo que los RPC son a los lenguajes estructurados y de procedimientos. La semántica local es extendida a los sistemas distribuidos lo más cercano posible.

En la Figura 2.2, el usuario percibe las invocaciones remotas como si los objetos se encontraran en la misma computadora (máquina del usuario), sin embargo los objetos podrían estar en diferentes máquinas (servidores 1 y 2).

[2] Esas referencias incluyen tanto la referencia misma, como el código necesario para comunicarse, esto gracias al principio de encapsulación que se logra con el uso de lenguajes orientados a objetos, es decir cada representante esta encapsulado con la referencia del objeto remoto al cual representa.

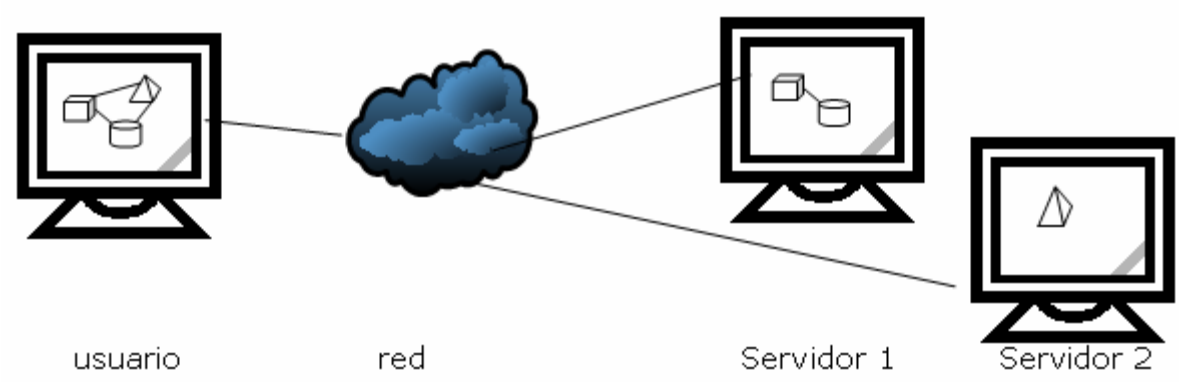


Figura 2.2. Vista de un usuario de un conjunto de objetos que se encuentran físicamente en diferentes computadoras

2.4.1 Principios de solicitud de sincronización en objetos distribuidos

El corazón de esta tesis maneja diferentes solicitudes de sincronización e invocación en Objetos Distribuidos. Es por ello, que para comprender los esquemas y los modelos propuestos en el Capitulo 3, es necesario estudiar las diferentes formas de solicitud y acceso a los métodos remotos de un objeto distribuido.

Existen cuatro modos para manejar la sincronización entre objetos distribuidos [16]:

a) Síncrono
b) Un solo sentido
c) Síncrono aplazado
d) Asíncrono

a) Modo síncrono

Las solicitudes estándares que CORBA, COM y Java/RMI soportan, son todas síncronas; esto significa, que el objeto cliente es bloqueado mientras el objeto servidor ejecuta la operación solicitada. El control sólo es retornado al cliente después de que el servidor ha completado la ejecución de la operación o el middleware[3] haya comunicado al cliente acerca de la ocurrencia de un error. (Véase Figura 2.3)

[3] Un middleware permite que los clientes envíen objetos y soliciten servicios en un sistema orientado a objetos.

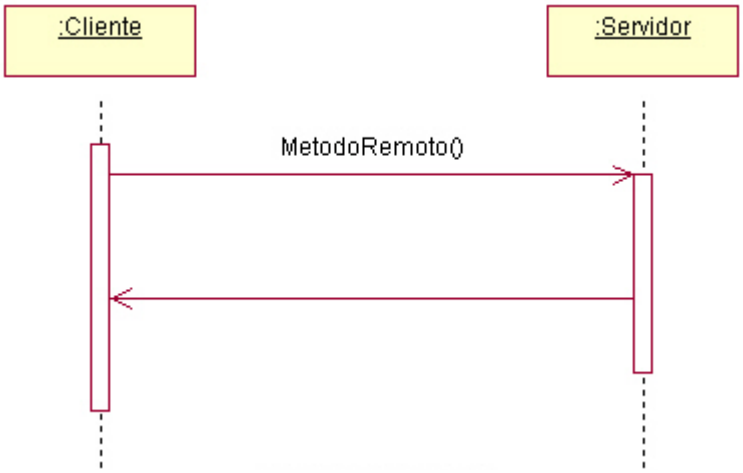


Figura 2.3 Modo Síncrono: Se invoca un método en espera de su terminación

Dependiendo del tiempo que se tarda en ejecutar las operaciones puede que no sea apropiado que el cliente se encuentre bloqueado, por ejemplo cuando varias operaciones independientes están destinadas a ser ejecutadas desde diferentes servidores puede ser apropiado ejecutarlas simultáneamente de manera que se tome ventaja de la distribución. Lo que no sucede con las solicitudes síncronas, los clientes en un simple hilo de ejecución sólo solicitan la ejecución de una operación a la vez. Es debido a este problema que se modelan diferentes opciones en el Capitulo 3.

b) Un solo sentido

Este modo regresa el control al cliente tan pronto como el middleware haya aceptado la solicitud, de manera que tanto el cliente como el método solicitado son ejecutados concurrentemente y no son sincronizados. La semántica del cliente no debe depender del resultado de la operación solicitada. (Véase Figura 2.4)

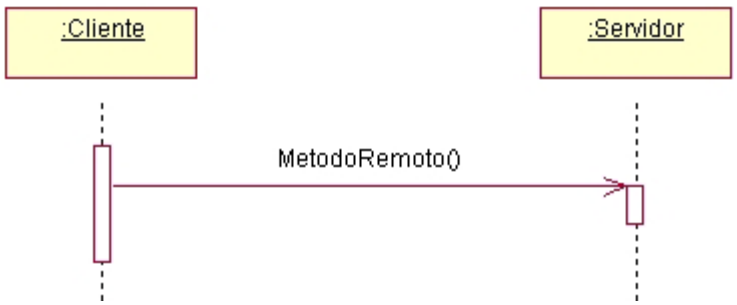


Figura 2.4 Un solo sentido, el cliente no espera

c) Síncrono aplazado

La sincronización aplazada puede ser usada en situaciones cuando se necesite que un resultado sea regresado al cliente. Regresa el control al cliente tan

pronto como el middleware de distribución (objetos de solicitudes utilizados para representar las solicitudes en curso) haya aceptado la solicitud. El objeto cliente no es bloqueado entre las solicitudes y puede ejecutar otras instrucciones. (Véase Figura 2.5)

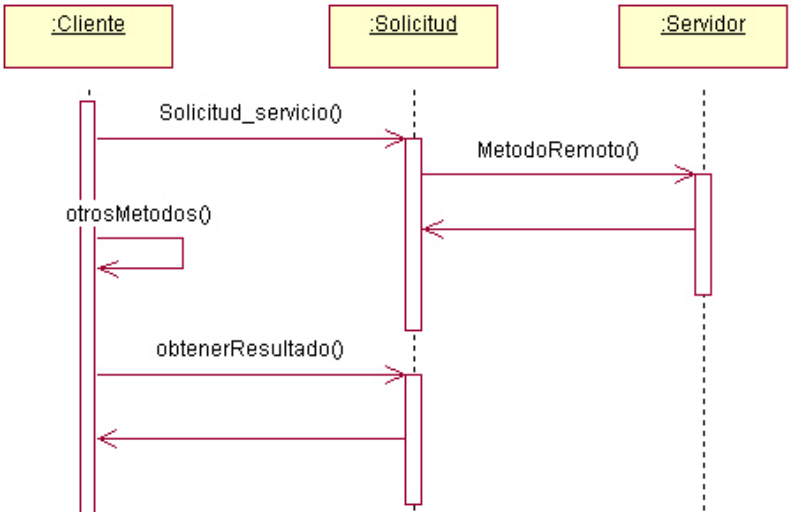


Figura 2.5 Síncrono aplazado, el cliente consulta el resultado más tarde

Una ligera desventaja del tipo síncrono aplazado, es que los clientes están a cargo de la sincronización con el servidor cuando obtienen el resultado, el cual en particular, puede no estar aún disponible cuando el cliente invoca la operación para obtenerlo. Entonces pueden pasar dos cosas: el cliente es bloqueado o la operación necesita ser invocada de nuevo más tarde (técnica conocida como escrutinio).

d) Modo asíncrono

Es cuando un cliente usa una solicitud asíncrona, recupera el control tan pronto como el objeto servidor ha aceptado la solicitud. La operación es ejecutada por el servidor y cuando finaliza, el servidor hace explícitamente una llamada a una operación del cliente (invoca un método del cliente –callback-) para transferir el resultado de la operación. De esta manera se evita la carga de procesamiento y tráfico de red que genera el polling para obtener la respuesta. Para que el callback o llamada al cliente pueda ser realizada, el cliente debe pasar su referencia al servidor. (Véase Figura 2.6).

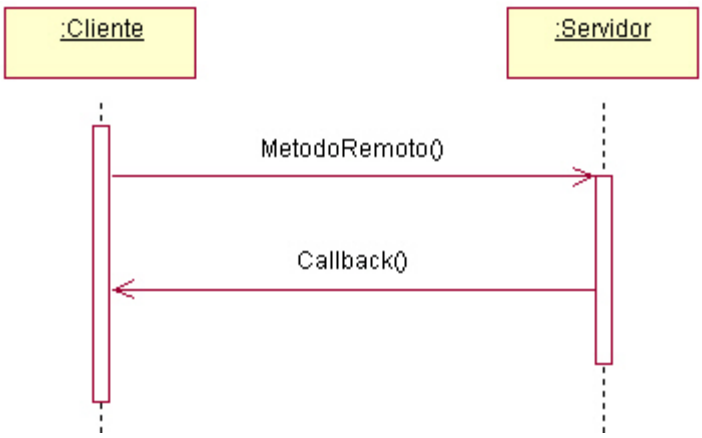


Figura 2.6 Asíncrono, el servidor proporciona el retorno a los clientes

Las tecnologías actuales de objetos distribuidos soportan las solicitudes de sincronización síncronas, sin embargo, usando computación multi-hilo junto con el modo síncrono se pueden implementar los modos restantes.

2.5 VRML y su interfaz con Java

VRML es un lenguaje para codificar escenas y objetos en tres dimensiones animados e interactivos como parte de la Web. La motivación para su creación fue conectar MV sobre Internet y permitir que varios usuarios interactúen simultáneamente con los objetos del mundo, debiendo ser una plataforma independiente y extensible y que trabajara con conexiones de bajo ancho de banda, metas parcialmente alcanzadas hasta ahora.

2.4.1 Breve historia de VRML

VRML 1.0

- Surge en 1994 por la necesidad de estandarizar un lenguaje para definir escenas tridimensionales con hipervínculos.
- La versión 1.1 de VRML estaba basada en el lenguaje Open Inventor de Sillicon Graphics (SGI) y el diseño a cargo de Mark Pesce.
- En 1995 fueron liberados muchos visualizadores[4], como WebSpace por SGI y Live 3D por Netscape.
- Los mundos en VMRL 1.0 eran estáticos, sólo se podía navegar en ellos y la interacción se limitaba a simples “clicks” en hipervínculos.

VRML 2.0

- Presentada en agosto de 1996 en el congreso más importante de gráficos por computadora, SIGGRAPH’96.
- Basada en el lenguaje Moving Worlds de SGI.
- Permite objetos en movimiento e interactivos, para lo que se implementaron conceptos como tiempo, sensores y eventos.
- Permite la inclusión de programas escritos en Java, JavaScript.
- A finales de 1997 empezó a ser un estándar de la ISO, llamándose VRML97.
- En 1996, 35 compañías de Internet formaron el Consorcio VRML para coordinar el estándar y en 1998 fue renombrado como Consorcio Web3D abarcando otras tecnologías 3D en Internet.

En el Apéndice A, se revisa más a detalle la tecnología VRML y sus mecanismos de integración con aplicaciones Java.

[4] Son los visualizadores de las escenas VRML que se conectan al navegador como plugins.

2.6 Modelo alma-cuerpo

El modelo alma-cuerpo, como se detalla en [1], describe la forma de interacción entre los elementos de un MV. A grandes rasgos, la arquitectura se define como un AVD poblado por un conjunto de "sujetos", los cuales están conformados por dos elementos: "alma" y "cuerpo". El cuerpo es la representación visual (geometría) del sujeto, mientras que el alma encapsula su estado y comportamiento. De esta forma, existe sólo una copia del estado global del AVD, representado por las almas de todos los sujetos que lo pueblan.

El modelo Alma-Cuerpo surge como parte de una investigación en el CIC; en el Capítulo 3 del presente trabajo se describe más ampliamente y se discute a fondo su implementación como parte de este trabajo de tesis.

2.7 Navegadores, Applets y seguridad

La capa cliente de los modelos propuestos es básicamente el cliente Web, el cual es el navegador de Internet; en él, se ejecutan tanto el visor de VRML para desplegar las escenas, como un *applet* de Java que se comunicará con la escena para actualizarla a través del API de EAI.

Dada la naturaleza de los *applets* como programas de Java descargados desde Internet a la máquina local y ejecutados en la JVM (*Java Virtual Machine*, Máquina Virtual Java) integrada en el navegador, este último tiene un mecanismo para prevenir que lo *applets* ejecuten instrucciones para acceder a los recursos de la computadora donde se ejecuta; dicho mecanismo es denominado "caja de arena", la cual se encarga de vigilar que la ejecución del *applet* no viole la integridad del sistema.

Sin embargo, en una de las arquitecturas propuestas es necesario que el *applet* funja como un objeto distribuido y por ende, reciba invocaciones remotas, por lo que para que el navegador permita el acceso al *applet* es necesario utilizar el mecanismo de firma de *applets.* Se firma digitalmente con un certificado digital el cual identifica el código que se va a ejecutar, permitiendo al usuario autorizar o denegar la ejecución al identificar al applet.

El proceso de firma varía dependiendo del navegador y del certificado; en el Apéndice C se describe el proceso a utilizar en esta tesis.

2.8 Trabajos previos relacionados

En los últimos años han surgido diferentes propuestas y metodologías para construir aplicaciones de RV distribuida en Internet utilizando para ello los mecanismos y tecnologías de comunicación existentes.

Algunas propuestas se basaron en hacer ligeras modificaciones al servidor HTTPD, usando CGI (*Common Gateway Interface*) y como consecuencia, cambios a los visualizadores de VRML para lograr la interacción en un esquema cliente-servidor [5]. Para reducir los problemas de este modelo (como cuellos de botella) surgieron propuestas que utilizaban protocolos *Multicast* además del servidor HTTPD extendido [5][6][7].

También se han propuesto protocolos como DIS (*Distributed Interactive Simulation*, Simulación Interactiva Distribuida) creado por el departamento de defensa de los Estados Unidos para entrenamiento militar virtual, el cual no es escalable y requiere de una red dedicada [8], DWTP (*Distributed Worlds Transfer and Communication Protocol*, Protocolo de Transferencia y Comunicación de Mundos Distribuidos) que es un protocolo de la capa de aplicación independiente y heterogénea [7], Mu3D (*Multi-User 3D Protocol*) el cual es un protocolo punto a punto que únicamente envía actualizaciones garantizando el orden causal de los eventos [9]. VRTP (*Virtual Reality Transfer Protocol,* Protocolo de Transferencia de Realidad Virtual) que aún se encuentra en investigación, es heterogéneo y se basa tanto en comunicación multicast y punto a punto [10].

La evolución de VRML1.0 a VRML2.0 tuvo muchos cambios significativos, quedando así, un estándar definido para crear MV con comportamiento e interacción, sin embargo no incluye soporte para múltiples usuarios, por lo que algunas propuestas incluyen extender el lenguaje VRML, en forma de extensiones que declaran nodos escritos al estilo orientado a objetos en la definición de la escena, para lograr la transmisión de mensajes hacia un servidor a través de un simple protocolo que distribuye los mensajes a todos los participantes [11].

Otras extensiones pretenden que la conversión de mundos VRML existentes a mundos de múltiples usuarios sea simple; algunas proponen nuevos tipos de nodos, como VSPLUS que permite compartir eventos con otros visualizadores vía nodos intermedios llamados nodos red (*net nodes*) [12][13]. Los nodos propagan eventos compartidos a todas las instancias de la escena, donde la coordinación de todas las instancias de un nodo red es tarea de un "administrador de consistencias". Otro proyecto llamado VASE Multicast V-R propone, además de la adición de un nodo red, añadir rutas compartidas y bloqueadas, así cuando un evento es transmitido sobre una ruta compartida, se propaga a todas las instancias del MV. Un evento sobre una ruta bloqueada primero intentará obtener un candado de un servidor central (Gatekeeper) y sólo si lo obtiene, el evento pasará a lo largo de la ruta y será propagado a

todas las instancias [14]. Dicho comportamiento se puede apreciar en la Figura 2.5:

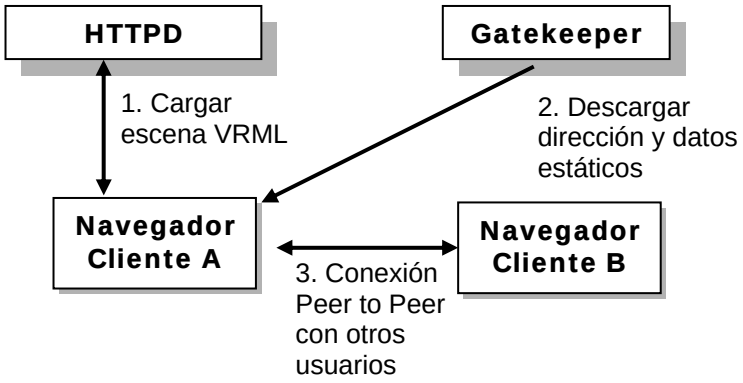


Figura 2.7: Interacción entre Clientes y el Gatekeeper

Así mismo el consorcio de VRML ha trabajado en una extensión de VRML llamada "Living Worlds", donde se propone añadir comportamiento e interacción entre usuarios al estándar VRML97. En Living Worlds existen "zonas" que son partes limitadas de una escena la cual contiene objetos compartidos. Cada zona tiene su propia tecnología multi-usuario (MUTech), la cual proporciona administración de consistencia y comunicación entre las instancias de objetos compartidos; Living Worlds no depende de una arquitectura específica. La especificación se ha denominado X3D y el esquema distribuido que maneja es el estándar *dis [8].*

El uso de los sistemas mencionados anteriormente requiere de mucho aprendizaje por parte de los desarrolladores de MV. En ciertos casos es necesario el uso de nuevos nodos, por lo que el código de la escena de VRML se vuelve difícil de entender.

VRML tiene una Interfaz de Autoría Externa (EAI); la cual permite comunicar una escena en VRML con aplicaciones externas, como los *applets*. Gracias a este mecanismo es posible dotar a los escenarios de características multiusuario y de colaboración, usando para ello un lenguaje de programación soportado por el API de VRML como Java y usando los mecanismos de comunicación proporcionados por el lenguaje. Recientemente han surgido otras arquitecturas que usan la EAI para construir MVD donde se propone el uso de infraestructuras basadas en agentes, IPC (*Inter-Process Communication*, Comunicación entre procesos) y tecnologías de objetos distribuidos como CORBA [15][12].

“No tienes un alma. Eres alma. Tienes cuerpo”
C. S. Lewis

En el presente capítulo se describe la parte medular de este trabajo: las extensiones al modelo denominado Alma-Cuerpo para construir mundos virtuales distribuidos.

Capítulo 3: Esquemas de implementación del modelo Alma-Cuerpo

Partiendo de la propuesta de utilizar el modelo Alma-Cuerpo [1] para la construcción de Mundos Virtuales Distribuidos (MVD); el modelo Alma-Cuerpo, está basado en la representación de un objeto en tres dimensiones (3D) dentro de un mundo virtual al cual se le llama sujeto y que se compone de su Alma y de su Cuerpo. La implementación de dicho modelo es propuesto con un objeto remoto (o distribuido) representando el alma y por una representación del cuerpo, la cual básicamente es la descripción en cualquier lenguaje de modelado de tres dimensiones como VRML, Java 3D.

Se utilizan esas tecnologías para resolver la problemática para la que fueron diseñadas, los objetos distribuidos para dar soporte multiusuario en un MVD y lenguajes de modelado de escenas 3D para el soporte de Realidad Virtual. Los objetos remotos pueden ser implementados con las tecnologías modernas que los definen, RMI, CORBA, etc.

La justificación de la descomposición del sujeto en las dos partes se apoya en una regla básica del diseño de software, la *separación de responsabilidades* ya que las actividades de comunicación del estado de los elementos del Mundo Virtual (MV) y el despliegue de la escena 3D son suficientemente independientes.

Gracias a esto es relativamente fácil desarrollar implementaciones con diferentes patrones de comunicación, políticas de actualización de las escenas, así como el reparto de la carga de trabajo entre participantes y servidor; todo lo anterior permite realizar comparaciones objetivas entre los costos y beneficios de utilizar estas alternativas de diseño.

Y es con esta idea que surge el presente trabajo, ya que aprovechando las características del modelo alma-cuerpo, se extienden diferentes modelos para hacer optimizaciones en la comunicación, consistencia y desempeño del mundo virtual distribuido.

A continuación se describe el modelo núcleo del presente trabajo, explicando de manera abstracta el comportamiento original propuesto por sus autores [1].

3.1 Modelo Alma-Cuerpo

El modelo Alma-Cuerpo define una arquitectura en la cual un mundo virtual distribuido es poblado por un conjunto de “sujetos”. Cada sujeto en el ambiente está constituido por dos partes: “alma” y “cuerpo”.

El “cuerpo” es la representación visual del sujeto, es decir, la forma o geometría que lo describe. El “alma” se encarga de encapsular el estado cambiante de las propiedades del cuerpo (como su posición, color, etc.) y definir el comportamiento de cada “cuerpo” en el ambiente (como poder desplazarse de un lugar a otro, cambiar de color al surgir algún evento, etc.).

La arquitectura propone tener réplicas de cada cuerpo en los diferentes ambientes que cada usuario controla, pero asociado a cada cuerpo existe una única alma que lo representa, por lo que se forma una relación biunívoca entre cada cuerpo con su alma.

Debido a que las almas son implementadas como objetos remotos, existe en esta arquitectura una única copia del estado global del mundo virtual distribuido, el cual es representado por las almas de los sujetos que lo pueblan. Gracias a ello se evita la implementación de algoritmos para replicar el estado de los elementos del Mundo Virtual Distribuido.

Este modelo se divide en 2 niveles de abstracción dentro de un Mundo Virtual Distribuido: Nivel Físico y Nivel Metafísico.

Tal como lo muestra la Figura 3.1, los objetos que pueden ser percibidos por el usuario se encuentran dentro del nivel denominado *físico* (como analogía del mundo físico, donde las cosas son tangibles). Los objetos dentro del nivel físico son construidos de acuerdo a la geometría del mundo virtual y de acuerdo a los mecanismos para capturar los eventos que son generados por el usuario al interactuar con el mundo virtual distribuido.

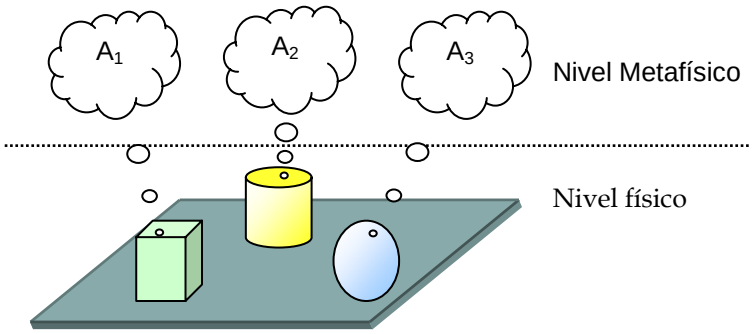


Figura 3.1 – Modelo"Alma-Cuerpo

El mundo físico es replicado en cada lugar en que un usuario existe como parte del MVD. Igualmente, en este nivel se encuentran los "representantes" que conectan a la copia del MVD con su estado global, es decir, las almas en el nivel "metafísico".

El nivel metafísico se compone de características imperceptibles, las cuales son vistas en el mundo real como comportamiento. Este nivel contiene el estado global del sistema y esta representado por el conjunto de las almas (A_1, A_2, A_3) de todos los sujetos que pueblan el MVD y que son susceptibles de cambiar de estado debido a la interacción con los demás sujetos. Gracias a este modelo, a pesar que el estado del sistema es único, puede ser distribuido geográficamente.

Todas las copias locales del MVD incluyen únicamente el nivel físico, mientras que el nivel metafísico es único, lo cual elimina los problemas relacionados a las posibles inconsistencias por múltiples copias del MVD, que siempre aparecen en los esquemas en que el estado del MVD es replicado. (Véase Figura 3.2)

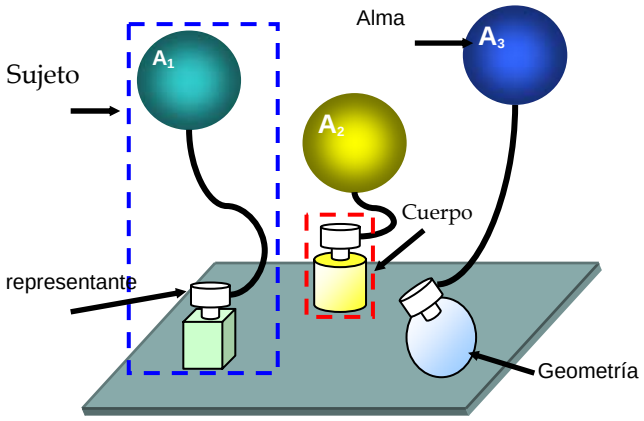


Figura 3.2. Elementos del Modelo Alma-Cuerpo

Es importante mencionar que en este capítulo se habla de los modelos teóricos de implementación cuya tecnología para logralos se verán en el siguiente capítulo.

Para entender cada esquema de implementación es necesario definir una representación básica de los elementos del modelo como clases de implementación, para poder entender de mejor manera las representaciones secuenciales de invocaciones a los métodos de modificación del estado del Mundo Virtual Distribuido. En la Figura 3.3 se muestra el mapeo entre los elementos del modelo y las clases de implementación básicas.

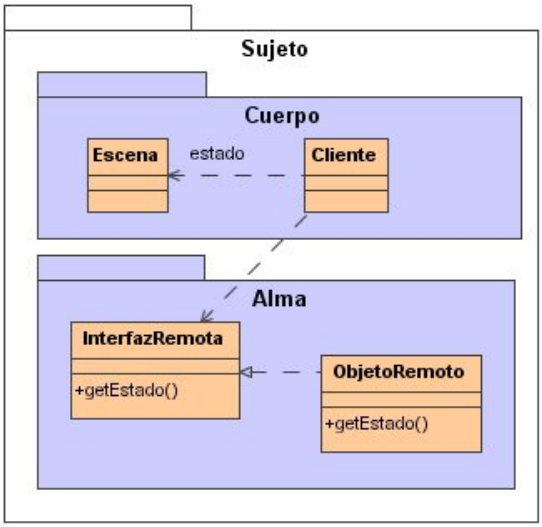


Figura 3.3 Representación de clases de los elementos del modelo alma-cuerpo

La descripción de las clases del modelo es la siguiente:

Escena: Es la clase de acceso a la geometría del sujeto y representa a la escena en 3D y proporciona el API para modificarla. Ante cada modificación de la escena el usuario podrá apreciar el cambio de la misma.

Cliente: Es la parte fundamental del cuerpo. Es la clase con acceso al representante (*stub)* del objeto remoto. Esta clase se encarga de obtener el estado encapsulado en el alma (objeto remoto) y acceder a la escena para modificarla.

InterfazRemota: Es la definición del comportamiento del sujeto, es decir, la definición de los métodos relacionados con la modificación del estado. El cliente accede al objeto remoto a través del uso de esta interfaz.

ObjetoRemoto: Es la implementación remota del comportamiento de los métodos definidos por la InterfazRemota; estos son los métodos para obtener, modificar el estado y la lógica de procesamiento para el cambio del mismo.

El modelo básico de Alma-Cuerpo propuesto es sujeto de ciertas optimizaciones de acuerdo al tipo de representación que se desea modelar.

Para ello, surge la necesidad de hacer diferentes modificaciones al modelo en pro de lograr en algunos casos un balance para obtener consistencia y eficiencia de recursos computacionales, las cuales son el tema fundamental de este trabajo y que a continuación se definen.

3.2 Implementaciones del modelo alma-cuerpo

Se presentan tres esquemas de implementación del modelo Alma-Cuerpo. En primer lugar se parte del esquema original, al cual llamaremos de escrutinio o *polling,* además se definirán diferentes optimizaciones a este esquema.

En el segundo esquema, se utiliza retro-alimentación para que sea el alma quien informe al cuerpo cuando ha cambiado de estado; a esta implementación se le llama retroalimentación o *callback* y también se definen optimizaciones.

En el tercer esquema se plantea (en casos en los que el modelo lo permita por la naturaleza del comportamiento) trasladar parte del comportamiento de los sujetos a las máquinas de los clientes; a este esquema se le conoce como *almas locales*.

3.2.1 Esquema 1: Escrutinio (*polling*)

Es el modelo básico que define la implementación alma-cuerpo, en el que los clientes solicitan constantemente el estado del sujeto al objeto remoto a través de la ejecución de uno de sus métodos, es decir, es una serie de iteraciones, en un intervalo definido de tiempo *t* de acuerdo a la complejidad del comportamiento. Con esta información se actualiza la apariencia del sujeto en el escenario virtual, con lo que cada uno de los usuarios puede ver el resultado de la actualización en el alma del sujeto.

3.2.1.1 Modelo básico: Clientes con un único hilo de actualización en cada tiempo definido

En este esquema el cliente obtiene la referencia al objeto remoto, con la cual se solicita el requerimiento del estado cada intervalo de tiempo denominado TIEMPO_ACTUALIZACION. Ya con el estado, el cliente actualiza el cuerpo (la escena virtual con la apariencia del sujeto). De esta forma cada usuario observa el estado global consistente del Mundo Virtual Distribuido. Este esquema se puede modelar con la secuencia mostrada en la Figura 3.4.

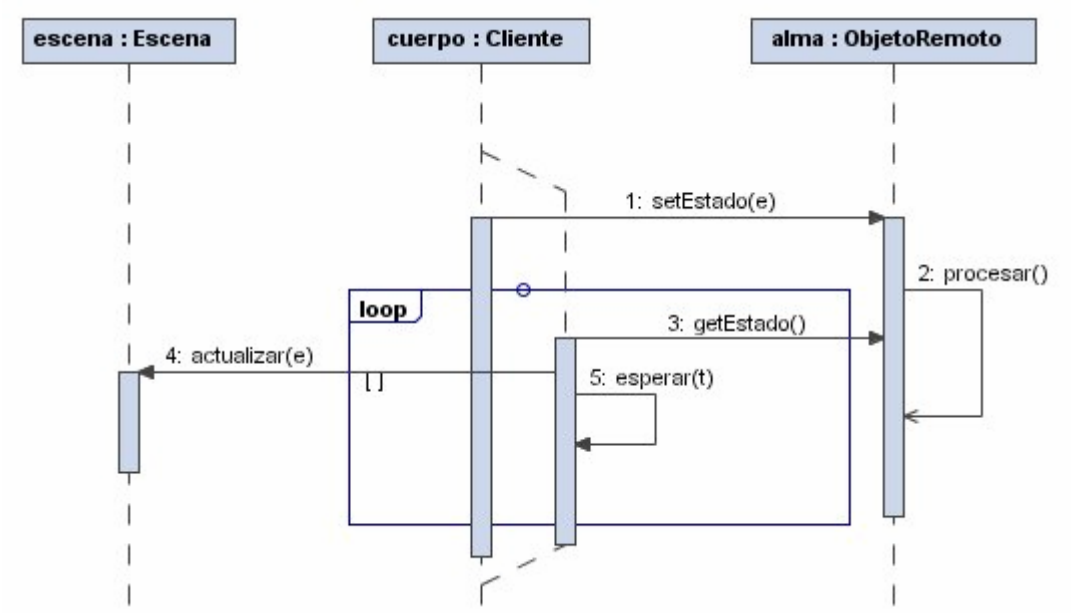


Figura 3.4 Modelo básico de escrutinio (polling)

En el diagrama de secuencias de la figura 3.4 se puede apreciar cuando el usuario solicita o modifica el estado (1), se lleva a cabo un procesamiento (2), o comportamiento el cual cambia el estado centralizado. Así mismo, en el cliente existe un hilo con un ciclo de actualización constante que repite los pasos de recuperar el estado, actualizar la escena y descansar (secuencias 3,4,5).

Aunque en este modelo se observan consistencias de actualización cada tiempo *t* se tiene un constante acceso a la red solicitando el estado, aunque por ser de forma secuencial produce un efecto de latencia debido a la semántica sincrónica de los métodos remotos, es decir, sus métodos son bloqueantes.

La consistencia se presenta cada determinado finito de tiempo, el cual podría variar ya que es un único hilo de actualización por lo que se depende directamente de la planificación del procesador.

3.2.1.2 Modelo básico extendido 1.1: Clientes con múltiples hilos de actualización en cada tiempo definido

Una mejora en la consistencia del estado en la escena virtual, es lanzar en el cliente múltiples hilos de actualización, como se aprecia en el diagrama de la Figura 3.5.

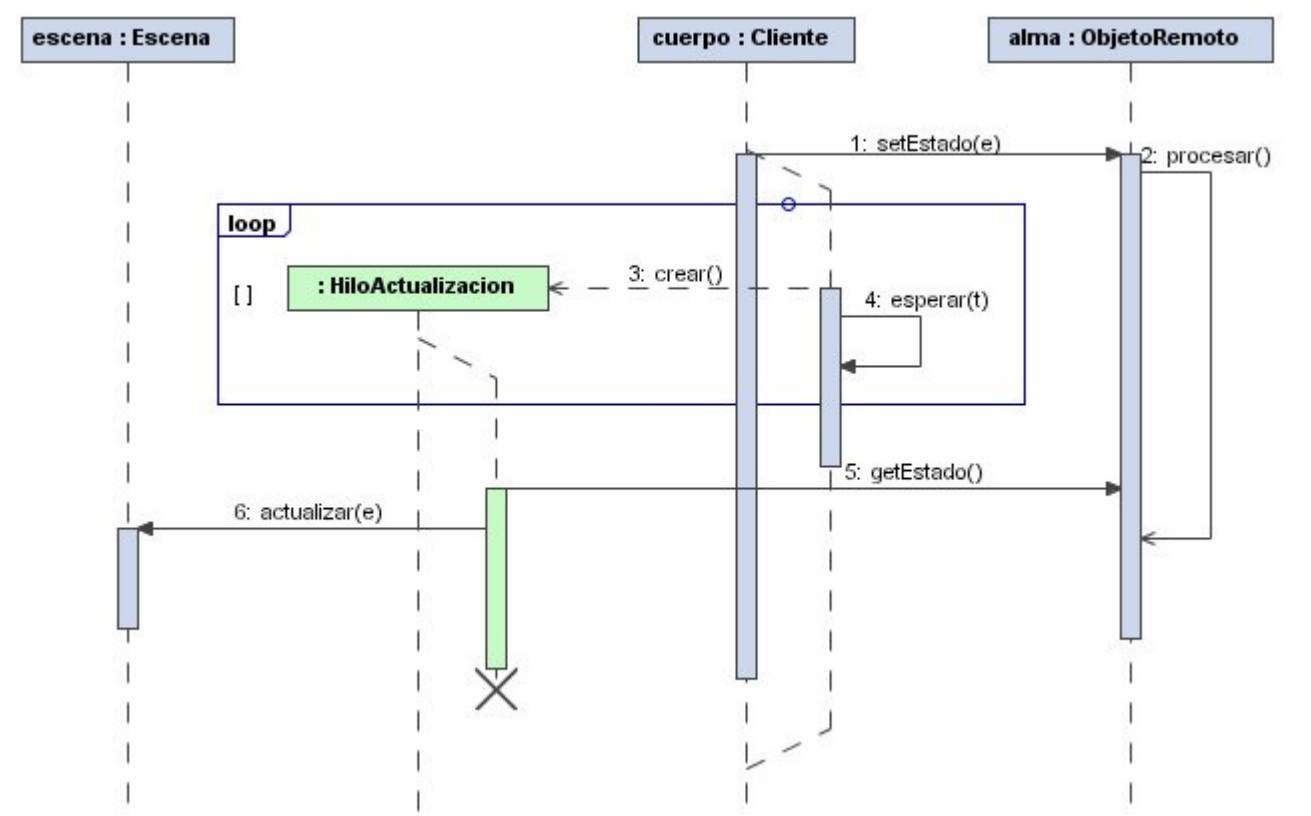


Figura 3.5 Escrutinio (polling) con múltiples hilos de actualización

En esta extensión se tiene mayor concurrencia en la solicitud al estado del objeto ya que el cliente, delega la actualización en múltiples hilos de ejecución especializados en realizar esta actividad. En este diagrama se puede observar como el ciclo lanza la creación de los hilos (3), lo que implica un elevado consumo de memoria así como del procesador en la máquina cliente debido al ciclo de crear-planificar-destruir los hilos de actualización. Sin embargo, se elimina la latencia de actualización ya que la espera del hilo cliente es solo para la creación de otro hilo y no para la invocación remota de un método, lo que da mayor consistencia y fluidez en la actualización de la escena virtual. El uso del ancho de banda se ve incrementado considerablemente por cada hilo creado, ya que debido a este tipo de procesamiento las invocaciones aunque son bloqueantes, son concurrentes y no secuenciales (5,6).

3.2.1.3 Modelo básico extendido 1.2: Clientes con hilos predefinidos de actualización en cada tiempo definido

Para evitar el crecimiento desmedido de hilos de actualización, se puede optimizar pre-creando el número de hilos que se estarán utilizando para este propósito. Lo que hará que se decremente el uso de memoria e instrucciones del procesador ya que se eliminará el proceso de creación-destrucción de hilos,

dejando únicamente el proceso de planificación, como se puede apreciar en la Figura 3.6.

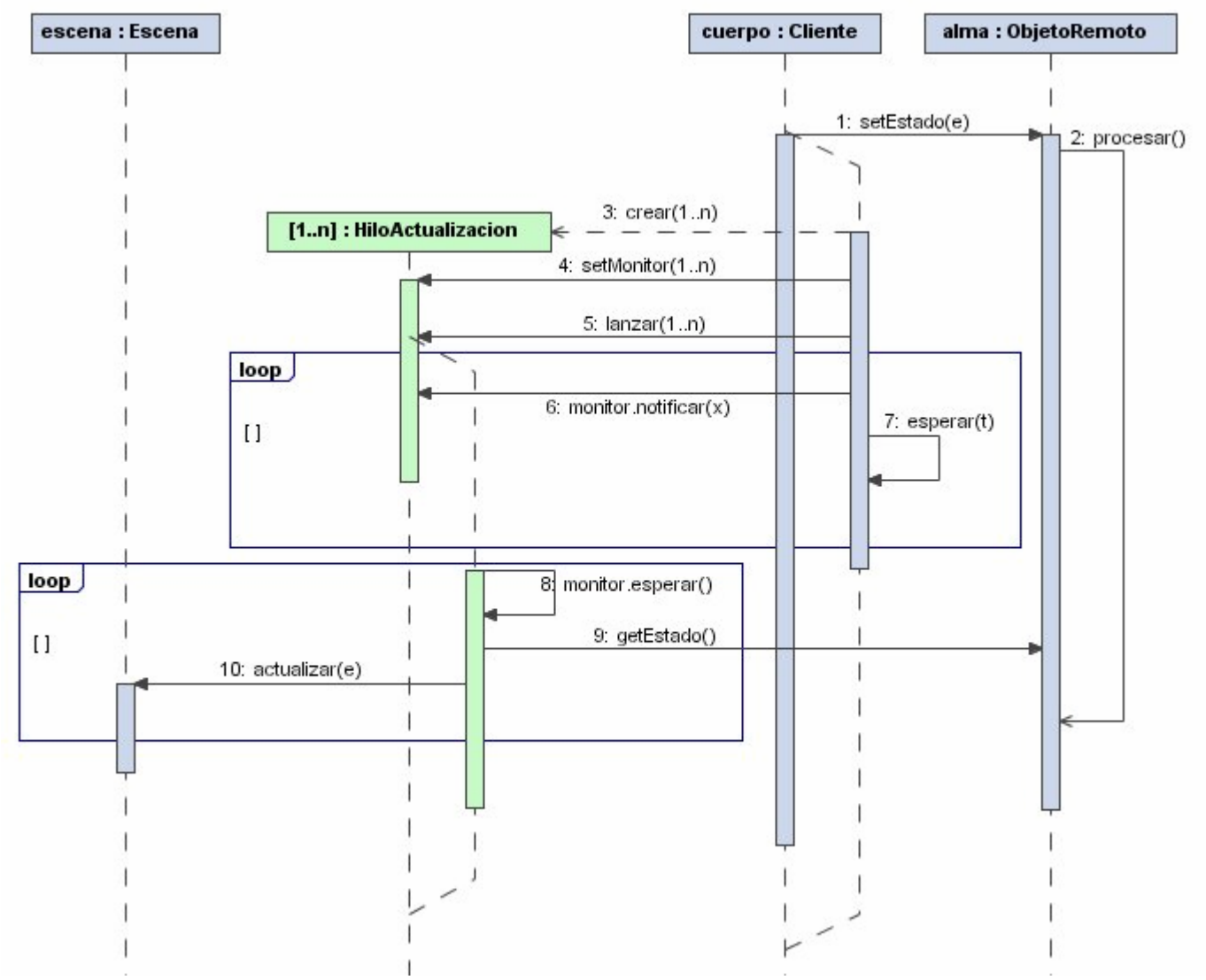


Figura 3.6 Escrutinio (polling) con hilos de actualización predefinidos

Con estas modificaciones el desempeño mejora considerablemente, gracias al uso de las directivas de concurrencia denominadas “monitores” que permiten manipular hilos de procesamiento.

Este esquema consiste en crear un conjunto finito y definido de hilos actualizadores de estado (3, 4, 5) y dormirlos (8), de manera que el cliente pueda irlos despertando de manera aleatoria (6) y con ese proceso se vaya actualizando la escena virtual (10).

Se mejora considerablemente el uso de memoria y de procesamiento del lado del cliente. También el ancho de banda se ve acotado a un número que puede ser medido y fácilmente restringible.

Ya se ha optimizado de manera considerable la parte cliente, en la extensión siguiente se conserva esta última secuencia y se optimiza la parte del procesamiento.

3.2.1.4 Modelo básico extendido 1.3: Clientes con hilos predefinidos de actualización en el cliente con optimización de cálculo en el objeto remoto

Este esquema mejora la parte de procesamiento del lado servidor como se muestra en el diagrama de secuencias de la Figura 3.7.

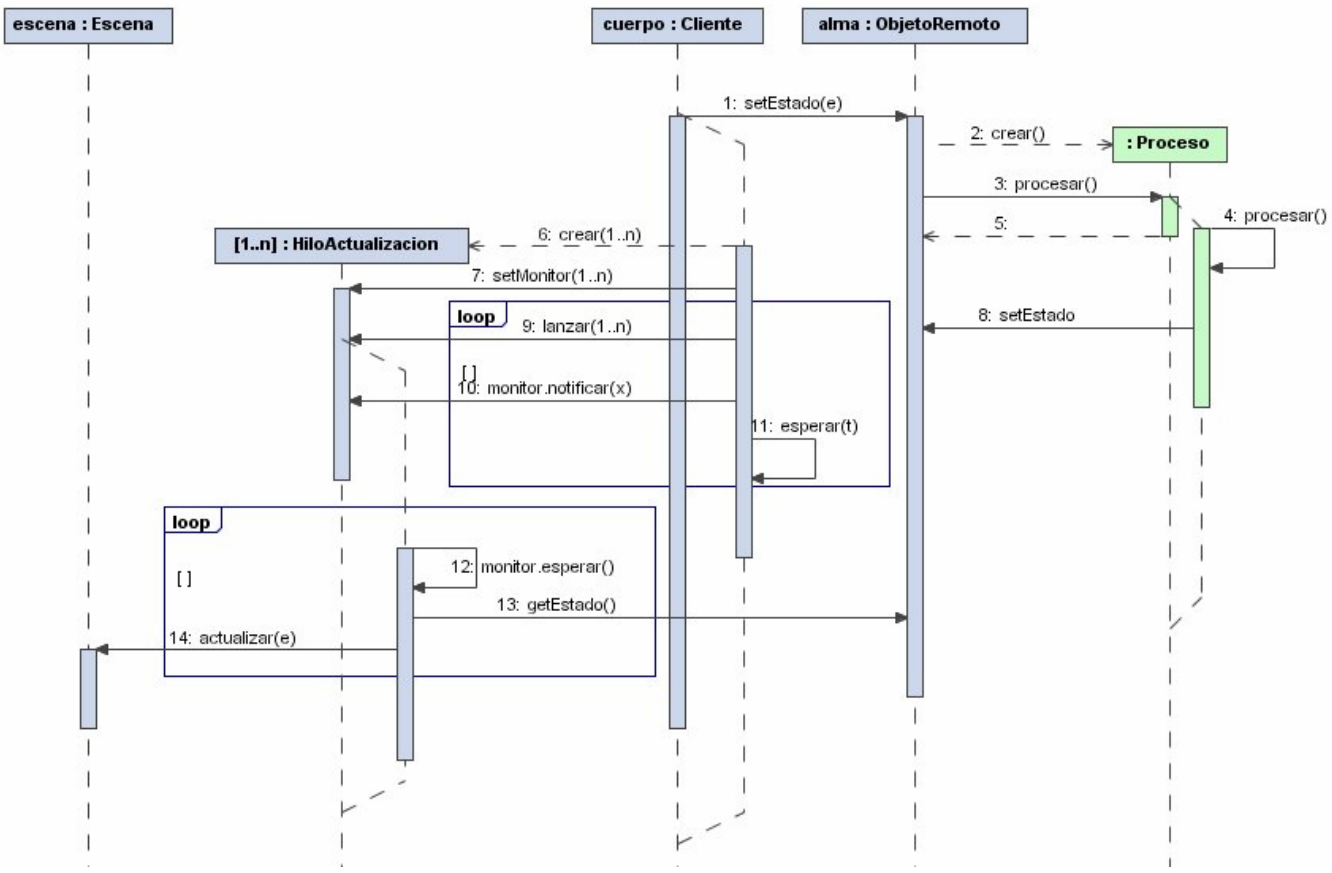


Figura 3.7 Escrutinio (polling) con hilos de actualización predefinidos con optimización en el objeto remoto.

En este esquema se optimiza el método de comportamiento del objeto remoto (3), para devolver el control a la invocación bloqueante al cliente la más pronto posible (5). Esto debido a que el cliente cuenta ya con un mecanismo

concurrente de actualizaciones no necesita que los métodos sean sincronizados.

Este modelo crea un objeto exclusivamente para llevar a cabo el procesamiento del comportamiento del objeto remoto (2), creando un hilo de ejecución (4) que modificará el estado del sujeto cuando le sea solicitado.

3.2.1.5 Conclusiones del esquema

Este esquema es aparentemente ineficiente, más aún si se tiene en mente que en cada sistema de colaboración con objetos distribuidos existe un compromiso entre la reacción (velocidad para percibir los efectos de nuestras propias acciones), el acceso (capacidad de interactuar con los elementos del Mundo Virtual Distribuido) y la consistencia (los usuarios perciben las mismas vistas de los mismos objetos al mismo tiempo) y también que el objetivo del modelo Alma-Cuerpo es preservar la consistencia de las vistas de los MVD.

Esta ineficiencia aparente usando este esquema (en reacción y acceso) es necesaria para mantener la consistencia de las vistas de los MVD. Sin embargo, como podrá observarse, se pueden realizar algunos cambios para mejorar el desempeño.

3.2.2 Esquema 2: Invocaciones a los clientes (Retroalimentación o callbacks)

El esquema anterior intenta llevar parte del procesamiento a los clientes del MVD, pero no se ocupa de optimizar la cantidad de mensajes enviados por la red necesarios para conocer el estado de los objetos. En este esquema se usa retroalimentación del alma hacia el cuerpo, es decir, del objeto remoto al cliente mediante su referencia.

En los esquemas previos, el cuerpo solicita una actualización del estado del sujeto al alma cada cierto tiempo (quizá, tan rápido como sea posible), sin importar si éste ha cambiado o no. Como se ha mencionado, esto se hace para asegurar la consistencia. Sin embargo, el número de mensajes necesarios transmitidos por la red para la actualización puede ser tan grande que podría estar haciendo cuellos de botella y congestionando el ancho de banda de la red.

Por ejemplo, imagínese un MVD pequeño con 5 sujetos y 5 portales; supóngase que el estado de los sujetos consiste en su color. Si cada uno de los 25 cuerpos (5 en cada uno de los 5 portales) solicita una actualización cada 0.5 segundos (que es mucho tiempo), entonces se estará hablando que en un segundo se generan 50 solicitudes del estado de los sujetos.

En este caso, el estado del sujeto consiste en el color del sujeto, que generalmente se maneja con tres números de punto flotante, es decir 24 bytes por cada solicitud. En total, cada segundo se envían 1200 bytes por los diferentes canales de la red. A esto hay que agregar que, como es notorio, para transmitir esa cantidad de información se requiere tener una serie de protocolos que a su vez agregan mayor carga de bytes a la red.

En los Mundos Virtuales Distribuidos grandes, el número de sujetos puede alcanzar el orden de miles e incluso millones. Por lo anterior, se requieren esquemas para que la actualización del estado de los sujetos en el MVD sea más eficiente. Uno de estos esquemas es, como mencionamos, utilizar *callbacks* desde el alma hacia el cuerpo.

3.2.2.1 Modelo 2.1: Simples Callbacks desde el alma al cuerpo

En este modelo, el alma es responsable de la consistencia del MVD. Debe mantener una referencia de todos los clientes y avisarles acerca de la actualización de su estado cuando este haya cambiado. (Véase Figura 3.8).

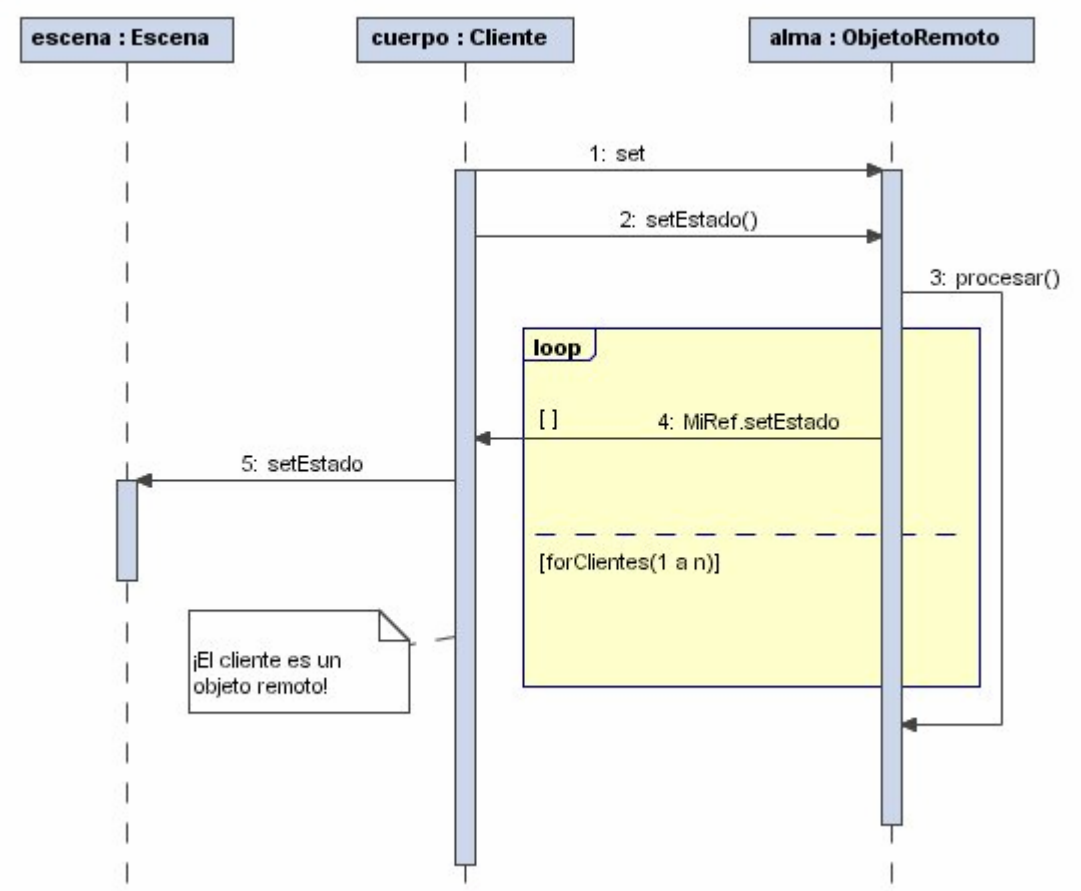


Figura 3.8 Simple Callbacks

Este es el modelo más simple del esquema de callbacks. Como se muestra en el diagrama de secuencias de la figura 3.8, una de las primeras cosas que tiene que hacer el cliente es proporcionar su referencia al objeto remoto (1).

Cuando alguno de los clientes realiza una invocación que cambia el estado (2) en el objeto remoto (alma) y se lleva a cabo algún tipo de procesamiento que involucre cambiar constantemente el estado del sujeto (3), el objeto remoto comenzará a actualizar el estado de todos sus cliente registrados a través de invocaciones a métodos de actualización en el cuerpo (4).

Es importante recalcar que para que un objeto remoto pueda invocar de manera remota métodos de sus clientes, es necesario que los clientes también sean objetos remotos. Es la única manera en la que se pueden llevar a cabo invocaciones hacia los clientes (*callbacks*) con las tecnologías modernas de objetos distribuidos.

3.2.2.2 Modelo 2.2: Callbacks desde el alma al cuerpo con pool de hilos de actualización

Este modelo extendido optimiza la invocación de los métodos de cambio de estado del cliente con el uso de un pool de hilos de actualización pre-creado como se muestra en el diagrama de secuencias de la Figura 3.9.

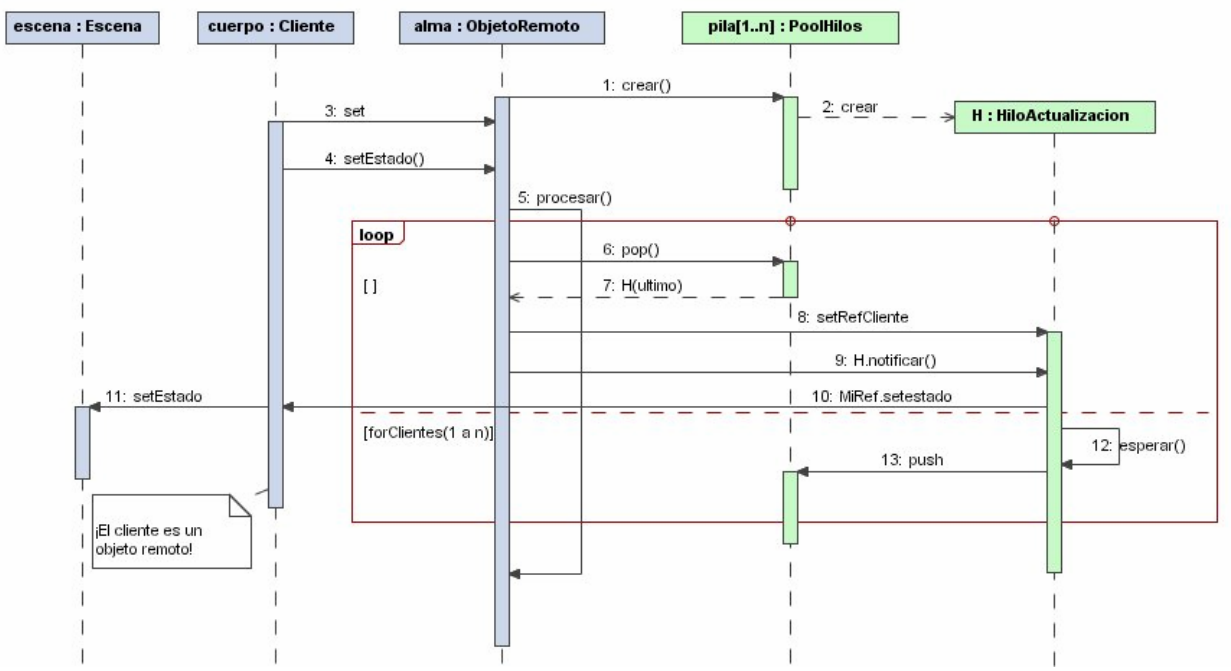


Figura 3.9 Callbacks con pool de hilos de actualización

En este modelo se crea un conjunto de hilos de Actualización (H) en una pila (1). Cuando se encuentra procesando un método de cambio de estado del sujeto, se le solicita a la pila un hilo (6) para usarlo específicamente con uno de los clientes registrados (8). Ese hilo será el responsable de actualizar el estado del cliente que le ha sido asignado (10). Después de la actualización el hilo reposa (12) y es auto-retornado a la pila para que pueda ser usado posteriormente (13).

Al utilizar un mecanismo de concurrencia como la pila de hilos, se optimiza la consistencia de actualización ya que en el modelo simple se tenía una actualización secuencial al ir iterando sobre cada cliente de manera secuencial.

3.2.2.3 Conclusiones del esquema

El esquema de *callbacks* o retroalimentación es más eficiente en términos de reacción, consistencia y acceso al medio. Ya que solo se utiliza el ancho de banda cuando se detectan cambios en el estado de los sujetos.

Sin embargo, manteniendo el objetivo de la consistencia, se tiene un uso recurrente de los recursos de red por cada cambio en el estado del sujeto.

3.2.3 Esquema 3: Almas Locales

En el esquema original, se tiene que preguntar cada vez por el estado de los objetos, incluso cuando éstos no se hayan modificado y en el esquema de retroalimentación (*callbacks*) se tiene que actualizar cada cambio que sufra el estado del sujeto de manera remota.

Además, pueden tenerse objetos cuyo comportamiento no sea complejo o no es tan importante la consistencia. Por ejemplo, algún objeto que se encuentre siempre girando o que su orientación no sea tan importante, como en el caso de animaciones con las bolas de billar en las cuales no importa como gire, ya que eso es sólo un efecto visual. En este caso, puede tenerse almas parcial o totalmente locales.

En este tipo de almas el comportamiento y el estado del sujeto se encuentran almacenados en el *proxy*. Esto quiere decir que existirá una instancia del alma para cada uno de los clientes del MVD. Sin embargo, se pueden tener almas parcialmente locales, es decir, objetos que realizan cálculos como los que hemos mencionado, pero tienen un aspecto de sincronización. Dicho aspecto puede incluir, entre otras cosas, una marca de tiempo, una posición, un orientación, en color, etc., tal y como lo define el estándar dis-java-vrml [18].

3.2.3.1 Modelo 3.0: Almas completamente locales

En la Figura 3.10 se muestra el esquema con alma local. Como se puede apreciar, el cuerpo accede al alma local como si lo hiciera con el alma remota, ya que el alma local encapsula estado y comportamiento.

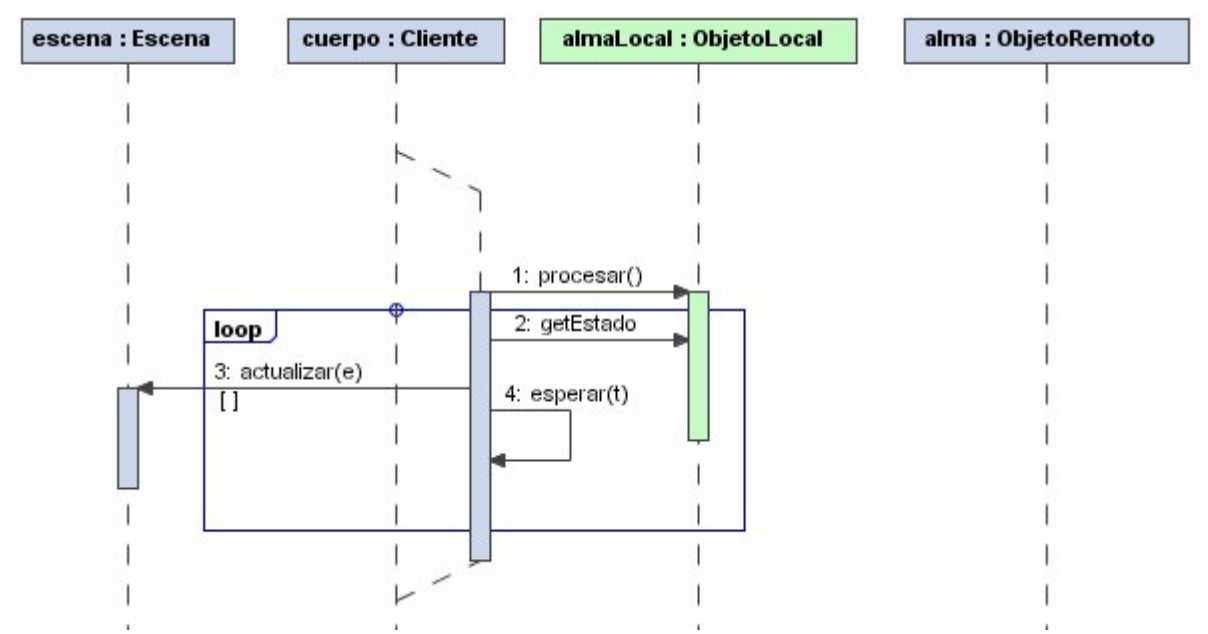


Figura 3.10 Alma completamente local

Para que este modelo funcione, cada cliente o cuerpo deberá estar procesando la misma información al mismo tiempo (presumiblemente) lo que serviría para actualizar elementos en el mundo que sean considerados como "despreciables". Esto puede provocar una total inconsistencia en el ambiente ya que cada máquina cliente tiene diferentes componentes de hardware y software.

Podría considerarse que esta definición se sale del planteamiento del modelo Alma-Cuerpo original. Por esta razón se muestra a continuación un par de optimizaciones a este modelo, las cuales consisten en mantener la consistencia del sujeto en los portales de los usuarios.

3.2.3.2 Modelo 3.1: Almas semi-locales con sincronización con polling

En la Figura 3.11 se muestra el caso en el cual las almas no son completamente locales y que existe un elemento de sincronización, en otras palabras el alma esta dividida en una parte local y en una parte remota y que la comunicación no se da entre el cuerpo y el objeto remoto, sino entre el cuerpo y la parte local del alma (1,2); ésta a su vez se encarga de sincronizarse con su parte remota (6).

Se puede observar como el alma local, usa la referencia remota para comunicarse con el alma (objeto remoto) (5).

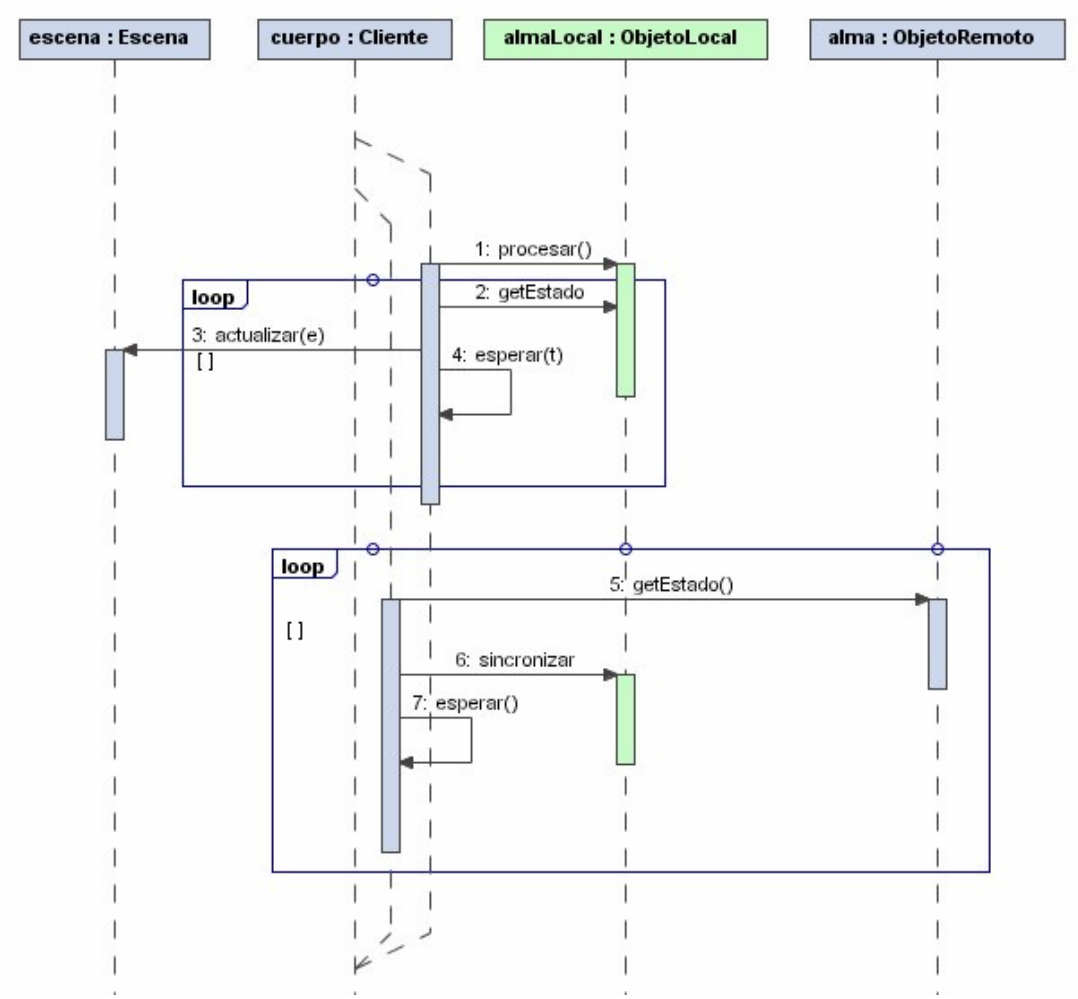


Figura 3.11 Alma semi-local sincronizada con polling

Este modelo tiene la misma desventaja que el polling ya que siempre pregunta por el estado aunque el sujeto no haya cambiado, esto debido, a su filosofía de mantener la consistencia.

3.2.3.3 Modelo 3.2: Almas-semi locales con callbacks

También se puede manejar el esquema de callbacks con las almas locales. La manera de hacer esto es modelar el comportamiento de manera local (Véase figura 3.12).

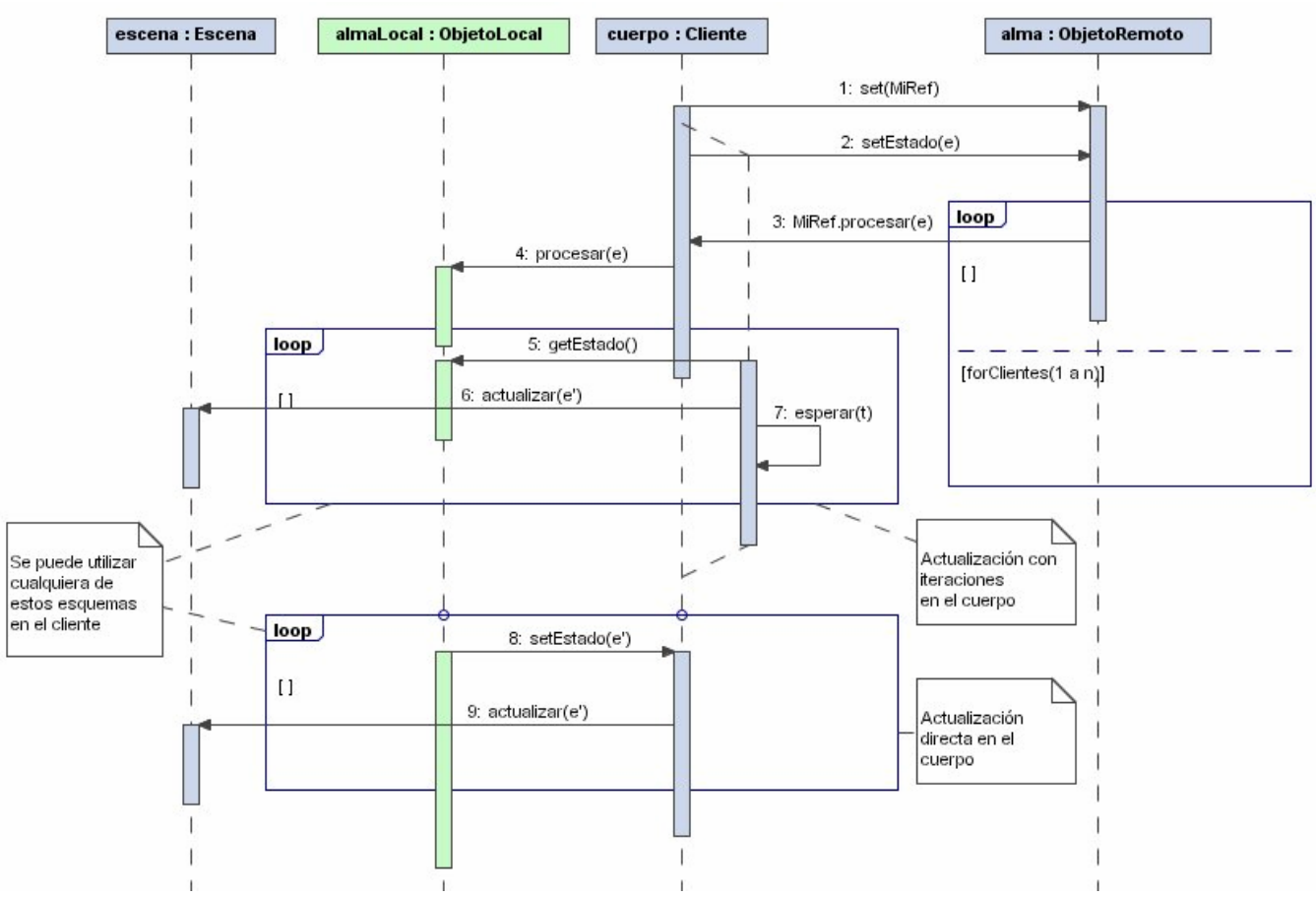


Figura 3.12 Alma semi-local con callback

En este esquema el cuerpo se comunica de manera inicial con su alma (remota) y como en los modelos de retroalimentación (*calbacks*), le pasa su referencia (1). Es importante mencionar que cuando ocurre un evento de solicitud de cambio de estado "e" desde un cliente (2), el objeto remoto pasa ese estado "*e"* al cliente con el objetivo que el procesamiento lo lleve a cabo el alma local (4). A partir de ese punto los sujetos se actualizan con un estado nuevo "*e'* " generado por las almas locales.

El esquema del diagrama de secuencias muestra 2 tipos de actualización de la escena que pueden ser usados dependiendo de los resultados que se deseen obtener.

El primero consiste en lanzar un hilo que le pregunte al alma local constantemente por el estado "e' " para realizar las actualizaciones de la escena, lo que recuerda el esquema de escrutinio básico.

En la parte inferior del diagrama se muestra un segundo tipo de actualización en la que el alma local actualiza el estado directamente, lo que puede considerarse como retroalimentación, ya que implica que el alma local tenga la referencia del cliente (cuerpo) y pueda invocar sus métodos de actualización de estado.

3.2.3.4 Conclusiones del esquema

El modelo de almas locales con retroalimentación se considera una buena opción para mejorar el desempeño de los MVD. Es fácil darse cuenta que con todas las optimizaciones realizadas a cada uno de los modelos mostrados, se pueden formar nuevas soluciones mezclando los conceptos de cada modelo.

Muchas veces el usar uno u otro modelo dependerá del tipo de sujeto que se este modelando. A lo mejor algún tipo de sujeto irrelevante para el usuario del mundo virtual distribuido podría tener un procesamiento del tipo de almas locales. O almas locales sincronizadas con ventanas grandes de tiempo.

“Las ciencias aplicadas no existen, sólo las aplicaciones de la ciencia”
Louis Pasteur

En este capítulo se presenta la arquitectura global, así como los elementos y recursos que se necesitan para implementar el modelo alma-cuerpo y cada uno de sus esquemas, presentando los motivos por los cuales se escogieron las diferentes tecnologías utilizadas.

Capítulo 4: Análisis y diseño de la arquitectura propuesta

Es necesario definir los esquemas de arquitectura física y lógica en la que los esquemas serán implementados y probados, para poder evaluar los resultados de cada característica presentada en el capítulo anterior. También es necesario definir el diseño lógico de cómo van a interactuar las tecnologías escogidas para llevar a cabo esta implementación.

4.1 Arquitectura física

Definir la arquitectura física es una cuestión que varía de acuerdo a la complejidad y tamaño de cómputo que requiera la simulación del Mundo Virtual Distribuido (MVD). Una distribución de equipos podría variar desde uno hasta el orden de las decenas de servidores, por lo que habrá que garantizar los mecanismos de crecimiento horizontal en cada una de las capas responsables de atender la simulación del MVD.

Sin embargo, para efectos del presente trabajo de tesis, se requiere de una infraestructura mínima para realizar las pruebas de los modelos presentados en el Capitulo 3. Dicha infraestructura se puede organizar como se muestra en la Figura 4.1.

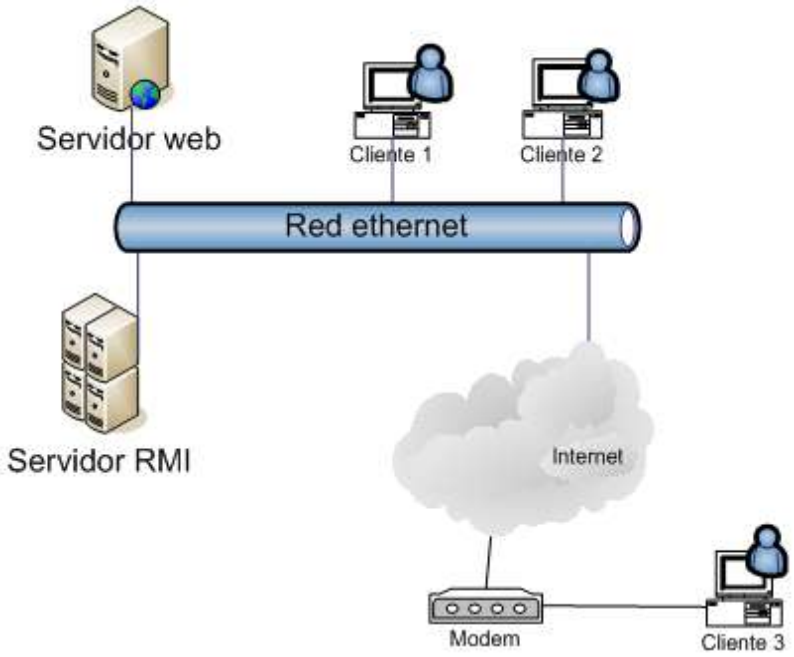


Figura 4.1 Infraestructura física

Clientes: Es necesario contar con al menos tres máquinas para los usuarios del MVD, dos en una red local a donde se encuentra el ambiente virtual y uno más accediendo desde Internet, ya sea accediendo por el dispositivo modem o por otra red interna. Los clientes deben tener instalado un navegador Web (como Netscape) y el visualizador de VRML para desplegar la escena.

Servidor WEB: Es un servidor http que se encarga de entregar la página de inicio del ambiente virtual distribuido. Este servidor proporciona los archivos estáticos HTML, VRML, imágenes, applets, etc.

Servidor RMI: Es un servidor con una Maquina Virtual Java donde se encuentran registrados y en ejecución los Objetos con Métodos Remotos y que pueden ser localizados a través de su servicio de nombres.

4.2 Plataformas tecnológicas escogidas

Para poder continuar con la definición lógica de la arquitectura, es necesario definir la plataforma tecnológica con la que vamos a trabajar en la implementación de los modelos descritos. Como se ha mencionado a lo largo de esta, el modelo Alma-Cuerpo es el corazón de la misma, por lo que hay que poner nombre y apellido a las tecnologías asociadas a cada elemento del modelo.

Para la implementación de los cuerpos hemos escogido la tecnología **VRML** [19] ya que se desea continuar con la arquitectura definida en el modelo original Alma-Cuerpo.

Las Almas serán implementadas usando la tecnología de **Java** [20] denominada **RMI** (*Remote Method Invocation*, Invocación de Métodos Remotos), dicha tecnología también está propuesta en el modelo Alma-Cuerpo y explicada brevemente en este capítulo.

Una de las características principales para la selección de estas tecnologías es que además de ser software libre, son compatibles con Java. Es decir, VRML tiene 2 tipos de integración con el lenguaje Java como se explica en el apéndice A dedicado a VRML (el nodo *Script* y la Interfaz de Autoría Externa EAI).

Aunque el modelo Alma-Cuerpo original define una implementación de referencia usando el nodo *Script* debido a la simplicidad del esquema ahí propuesto, en este trabajo es necesario contar con un mecanismo para actualizar la escena virtual más eficiente y de mayor flexibilidad. Es por ello, que va a trabajarse con la EAI de VRML. En el Apéndice A se profundiza el uso de la EAI.

Hasta este punto ya se tienen definidas las tecnologías para el alma y la definición de la escena virtual tridimensional del cuerpo, sin embargo, falta el elemento representante e intermediario entre ambas tecnologías, es decir, un componente que sea capaz de acceder al alma y además utilizar el API que define la EAI de VRML para actualizar la escena.

Dada la característica tecnológica de VRML, es decir, que la escena se despliega usando un visualizador para el navegador Web, entonces, se necesita un código que se ejecute en el navegador, este código "cliente" por añadidura debe estar escrito en lenguaje Java y la tecnología definida para este propósito es el estándar de **Java Applets**.

El uso de la EAI de VRML dentro de los applets de Java tiene como ventajas:

1. La capacidad de los applets para descargar código y comunicarse con otras aplicaciones por medio de la plataforma Java
2. Desarrollar controles visibles, como botones, que interactúen con el visualizador de VRML y con el navegador web
3. Soporte de seguridad, como certificados y firmas digitales
4. La capacidad para usar toda la gama de tecnologías de la plataforma Java (*Swing, CORBA, RMI,* Multi-Hilos)

La única desventaja del soporte de la EAI dentro de la tecnología de Java Applets es que no se puede usar la tecnología Java PlugIn de los navegadores Web pues carece de soporte para la tecnología LiveConnect y la implementación del visualizador de VRML depende de dicha tecnología.

Esta tesis no es un tutorial de RMI o de VRML, ni mucho menos de la construcción de Java Applets, sin embargo será necesario que el lector profundice en estas tecnologías en caso de querer realizar sus propias pruebas con lo que se proponga en este trabajo.

4.2.1 Un vistazo en la arquitectura de Java RMI

La especificación de Java RMI fue desarrollada por Sun Microsystems. RMI le da la capacidad al objeto cliente que reside en una máquina virtual de Java (JVM por sus siglas en inglés) de invocar un método desde un objeto servidor remoto en una JVM diferente. Java RMI debe ser considerado como un **middleware** orientado a objetos y se puede considerar a las invocaciones a métodos remotos en Java como solicitudes de objetos entre los objetos Java que están distribuidos a través de las diferentes JVM.

La secuencia de invocación inicia de la siguiente manera:

- Los clientes comienzan las invocaciones a los métodos remotos realizando una llamada local a un código representante (stub).
- Los *stubs* son específicos de las interfaces remotas e incluyen a todos los métodos remotos que están disponibles desde el objeto remoto del lado del servidor.
- Los clientes obtienen referencias de objetos remotos por medio del servicio de nombres el cual es implementado en el registro (registry).
- Los objetos servidor registran sus referencias a objeto con el registro de manera que los clientes pueden localizarlos.

- Los objetos servidor pueden ser activados explícitamente por algún administrador o implícitamente cuando se hace una invocación de un método remoto a ese objeto. En el último caso debe ser posible activar al objeto, el cual debe usar interfaces de activación para registrarse a sí mismo. Durante la activación las interfaces de activación llaman al constructor del objeto y permiten al objeto restaurar su estado. Una vez que un objeto es activado el control es pasado al esqueleto (*skeleton)* el cual invoca al método remoto deseado.

Los elementos de esta secuencia se pueden apreciar en la Figura 4.2.

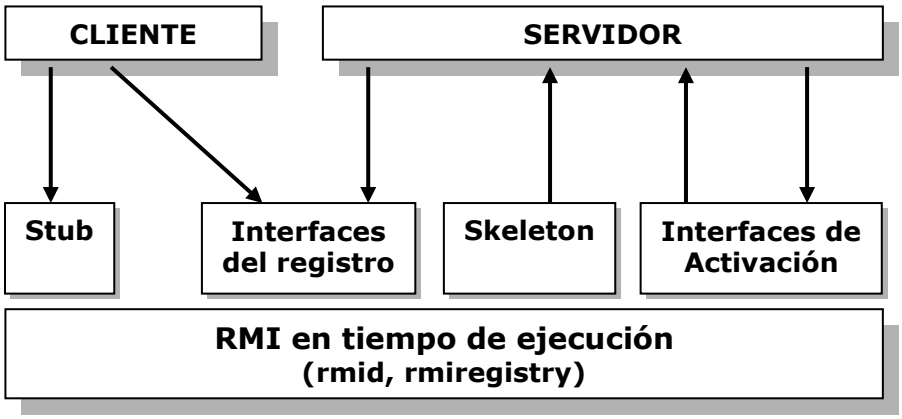


Figura 4.2 Arquitectura de las invocaciones a métodos remotos

Los representantes y los esqueletos son específicos del tipo de un objeto servidor remoto que implementa una invocación de método remoto. Ambos deben llevar a cabo el aplanado y desaplanado (en el lenguaje Java se utiliza el mecanismo de serialización de objetos para llevarlo a cabo) de los parámetros de invocación y por lo tanto implementan la capa de presentación. La especificación Java/RMI requiere que **rmic**, que es un compilador que produce los códigos representantes y esqueletos, se encuentre disponible. Este compilador no trabaja sobre definiciones de interfaz, sino que los genera a partir de la clase que implementa una interfaz.

4.3 Arquitectura Lógica

Una vez definida la plataforma tecnológica se tienen que definir las capas lógicas de implementación de los diferentes elementos que serán necesarios para construir y evaluar cada esquema propuesto en el capítulo 3.

En la Figura 4.3 se muestran los elementos definidos como plataforma tecnológica, agrupados por la ubicación que ocupa cada tecnología y mostrando la forma en que están conectados.

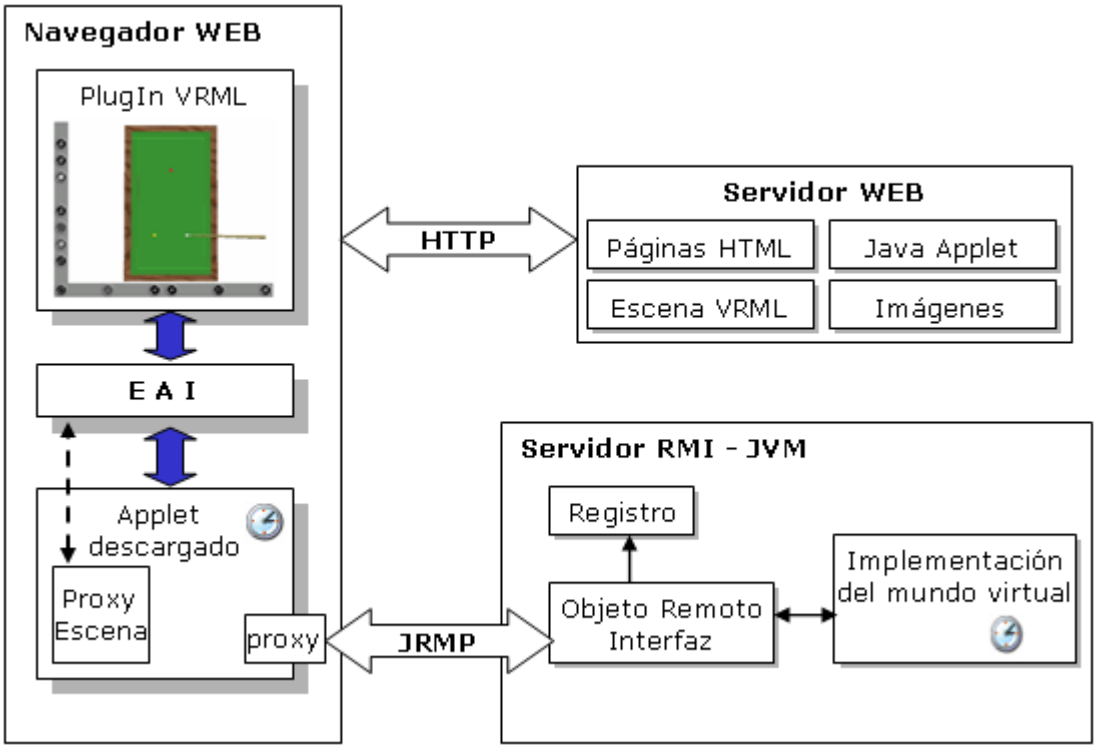


Figura 4.3 Arquitectura lógica

Se comienza a describir cada componente, estableciendo la relación con su correspondiente elemento en la arquitectura física.

Navegador Web: Son los clientes en la arquitectura física. Cada navegador tiene instalado el visor de VRML para poder desplegar las escenas tridimensionales descargadas en el formato de VRML (archivos con extensión *.wrl). El navegador también ejecuta el Java Applet descargado del servidor Web y este hace uso del API de la EAI de VRML para acceder a modificar y consultar el estado de los elementos de la escena VRML.

El Applet descargado conoce la referencia al objeto remoto a través del servidor RMI, las invocaciones de los métodos remotos se logran gracias al protocolo JRMP (*Java Remote Method Protocol*). Es necesario que en el paquete del applets existan las clases representantes del objeto remoto así como la definición de su Interfaz, para conocer la lista de métodos que pueden ser invocados de manera remota.

El elemento denominado ProxyEscena dentro del Java Applet es una clase Java encargada de encapsular los accesos a la escena gracias al API de la EAI.

Se recomienda para las pruebas de esta tesis se instale como navegador Web el Netscape Communicator 4, ya que la implementación de la JVM que trae por defecto cumple con la implementación de Sun de RMI.

Servidor Web: Proporciona la página HTML de inicio a la que los clientes se conectan cuando quieren desplegar la escena virtual. La página HTML contiene en su código la referencia al archivo VRML y al Java Applet. Cuando el HTML esta siendo desplegado en el cliente y encuentra estas referencias, de inmediato le solicita ambos elementos al servidor web.

El servidor Web es encargado de despachar al cliente los elementos estáticos como imágenes, archivos VRML, HTML. El protocolo de comunicación del cliente con el servidor Web es conocido como protocolo **http** y se basa en solicitud-respuesta, además que no tiene estado y no recuerda el estado entre petición y petición.

Servidor RMI: Es el servidor donde se encuentra instalada la Máquina Virtual Java que contiene las instancias de lo objetos remotos y también el servicio de registro de nombres para localizarlos y obtener sus referencias.

Cada Objeto Remoto representa al Alma y tiene una Interfaz que define los métodos que pueden ser invocados por el cliente.

Es en esta capa donde se encuentra implementado el comportamiento de los sujetos, así como los mecanismos para iniciar el procesamiento de alguna habilidad de los sujetos, y los mecanismos accesotes y mutadores del estado de los mismos.

Secuencia de actividades dentro de la arquitectura lógica

1. El usuario solicita la página HTML de inicio al servidor Web a través de un URL.
2. El servidor Web despacha la página HTML.
3. El navegador Web del usuario despliega la página HTML.
4. La página va al servidor Web por los elementos que la componen, es decir, la escena VRML (*.wrl), el applet (*.jar), imágenes (*.gif, *.jpg).
5. El navegador web despliega la escena VRML usando el plugin de VRML.
6. El applet ejecuta su método de inicialización.
7. El applet busca en el servidor RMI la referencia del objeto remoto y la obtiene.
8. El applet despliega su interfaz gráfica en el navegador Web para que el usuario comience a utilizarla.
9. Cuando el usuario realiza una acción sobre el applet, esta acción afecta la escena VRML y si es requerido accede al objeto remoto a través de su referencia para modificar el estado.
10. Depende del esquema implementado (Véase Capitulo 3) es como los clientes obtendrán el nuevo estado del sujeto para actualizar la escena.

4.4 Arquitectura lógica de implementación para los diferentes esquemas

Para hacer el mapeo entre la arquitectura lógica definida anteriormente y el modelo alma-cuerpo, es necesario identificar cada una de sus partes relacionándola con los elementos de las plataformas tecnológicas escogidas y mencionadas previamente. Se identifican los tres elementos fundamentales del modelo:

1) CUERPO, que es básicamente el escenario VRML, el cual contiene la descripción de la geometría que representa al sujeto.

2) ALMA, el objeto remoto, es decir un objeto Java RMI, el cual implementa el estado y las habilidades del sujeto.

3) PROXY, un objeto local (que en realidad forma parte del cuerpo) el cual se encarga de "enlazar" el escenario VRML (el cuerpo) con el objeto remoto (el alma).
Estos tres elementos y sus relaciones se muestran en la Figura 4.4.

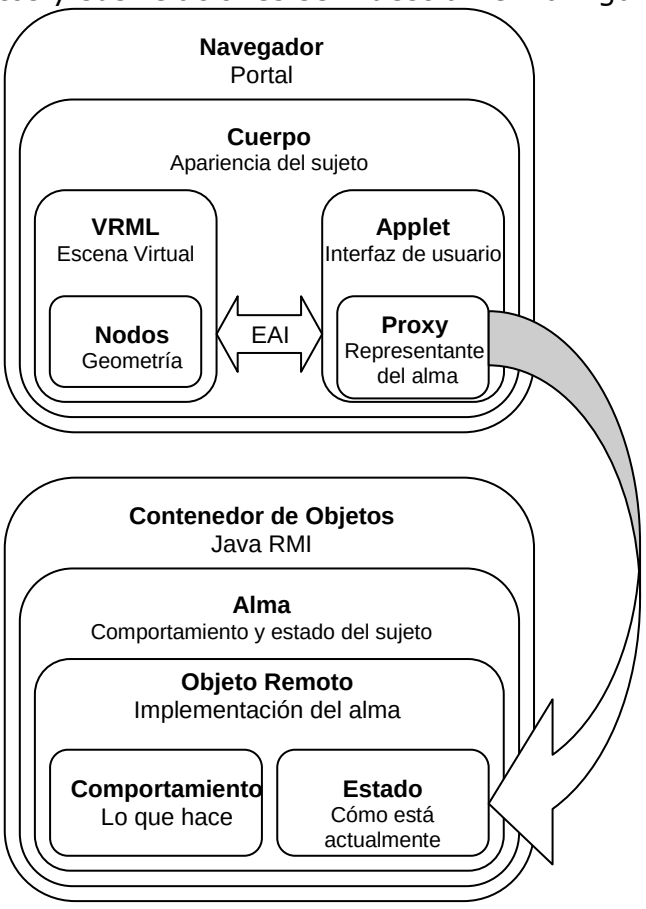


Figura 4.4: Elementos del modelo alma-cuerpo y sus relaciones en la arquitectura lógica

La implementación básica de estos elementos está definida por el conjunto de clases que representan cada elemento y que usan las tecnologías seleccionadas.

El diagrama de paquetes de la Figura 4.5 muestra la estructura de clases, en color blanco, que deberán de implementarse.

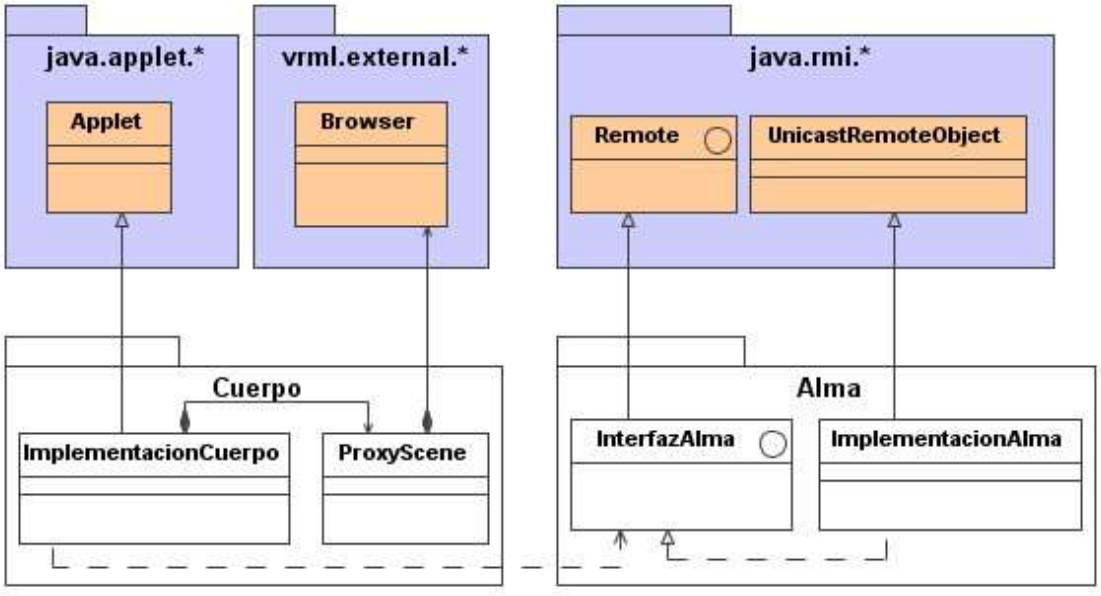


Figura 4.5 Diagrama de paquetes de implementación del modelo Alma-Cuerpo genérico

En la capa superior se muestran los paquetes de las tecnologías mencionadas en este capítulo que se utilizarán para implementar el modelo de referencia de cada esquema propuesto en el presente trabajo de tesis.

A continuación se describen los elementos de dicho diagrama:

Paquete java.applet.*
Contiene las clases necesarias para definir un applet de Java. La clase Applet contiene todos lo métodos que habrán que implementarse por el Cuerpo para que el navegador web pueda desplegarlo como Applet de Java.

Paquete vrml.external.*
Es el paquete con el conjunto de clases que forman el API de la EAI de VRML. A través de este paquete se obtiene la referencia a la escena VRML y las demás clases y métodos para cambiar los elementos que pueblan el mundo virtual desplegado en el plugin de VRML. Contiene entre muchas más, una clase llamada *Browser*, que es la referencia al plugin de VRML del navegador web.

Elementos del Cuerpo

ImplementacionCuerpo:
Es la implementación del Applet. Contiene la implementación que se encarga de mantener la referencia al Alma y la referencia al representante de la Escena para actualizar el estado de los elementos virtuales. Es el cliente de las invocaciones de los métodos remotos del Alma.

ProxyScene:
Es la clase que contene la referencia al Browser VRML a través del paquete vrml.external.* Se utiliza esta estrategia para desacoplar el Applet de acceder directamente a las clases y métodos de la EAI de VRML.

Elementos del Alma

Interfaz alma:
Es la Interfaz Remota que define los métodos que podrán ser invocados desde el cliente de manera remota. Esta interfaz estará disponible al cuerpo (applet) para que a través de ella pueda realizar las invocaciones.

ImplementacionAlma:
Es el Objeto Remoto. Es la implementación de los métodos definidos en la interfaz remota, además de implementar los mecanismos para registrarse en el servicio de nombres de RMI para que pueda ser localizado y accedido.
También implementa los métodos privados que definen algún comportamiento específico. Los clientes solamente utilizan los métodos definidos en la Interfaz Remota.

4.4.1 Arquitectura de implementación del esquema 1: Escrutinio (polling)

En este esquema, la manera de actualizar la escena VRML es iterando sobre el objeto remoto, es decir, el alma. Para ello el cliente utiliza cualquiera de los modelos definidos en el capitulo 3 sobre este esquema. En la Figura 4.6, se aprecia la dirección de la flecha del Proxy hacia el alma en un único sentido:

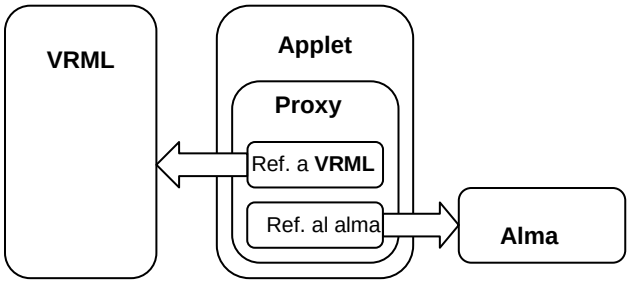


Figura 4.6: Elementos del modelo alma-cuerpo y sus relaciones con el esquema de escrutinio

Es importante recalcar que el escrutinio es siempre utilizado, aún cuando no existan cambios en el estado de los sujetos.

En el diagrama de la Figura 4.7 se muestra la implementación de este esquema con las tecnologías descritas. Es importante mencionar que se esta modelando el esquema óptimo de concurrencia de actualización de estado.

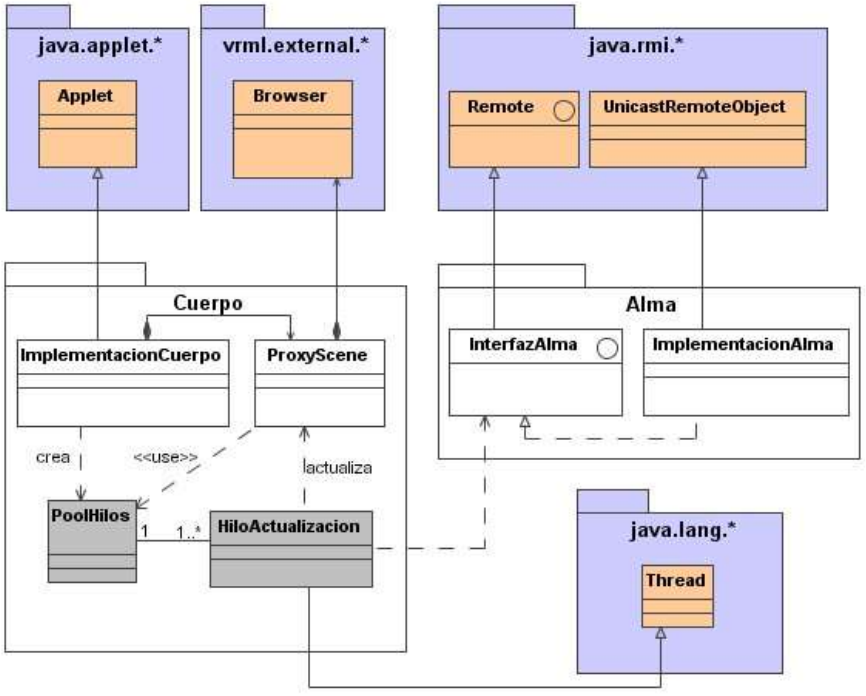


Figura 4.7: Diagrama de paquetes que implementa el esquema de polling

4.4.2 Arquitectura de implementación del esquema 2: Retroalimentación (callback)

Por la naturaleza del esquema anterior, se utiliza un esquema ineficiente en la transmisión de mensajes sobre la red para preguntar el estado de los sujetos. La optimización más importante sobre este esquema es que solo se actualicen los clientes (cuerpos) cuando el estado haya cambiado. Computacionalmente hablando, esto es posible si la implementación del alma "ve" la implementación del cuerpo, es decir, si el objeto remoto tiene la referencia de cada cliente y puede invocar los métodos de cada uno de ellos para actualizar su estado. Este esquema es sobre demanda.

En la Figura 4.8 se muestran los elementos que componen dicho esquema; se puede apreciar que el alma mantiene una referencia hacia el Proxy de cada cliente, con lo cual podrá realizar las actualizaciones correspondientes cuando el estado del sujeto cambie.

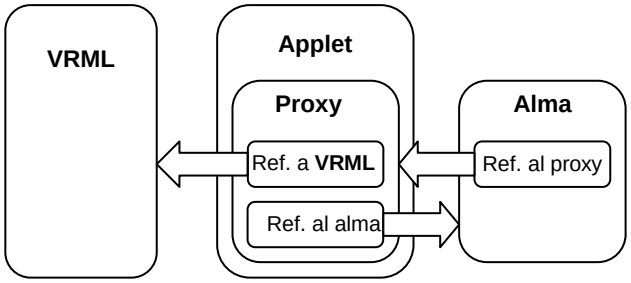


Figura 4.8 Elementos del modelo alma-cuerpo y sus relaciones con el esquema de retroalimentación

En este caso, el alma es quien se encarga de mantener consistentes los cuerpos que conforman el Mundo Virtual Distribuido ya que debe mantener una referencia de todos los proxies y avisarles acerca de la actualización de su estado, invocando los métodos que los cuerpos (clientes) definan para ello.

Es importante mencionar que para que este esquema pueda llevarse a cabo los clientes también deben ser objetos remotos, para que sus métodos puedan ser invocados de manera transparente por las Almas.

El diagrama de clases genérico correspondiente a este esquema se muestra en la Figura 4.9.

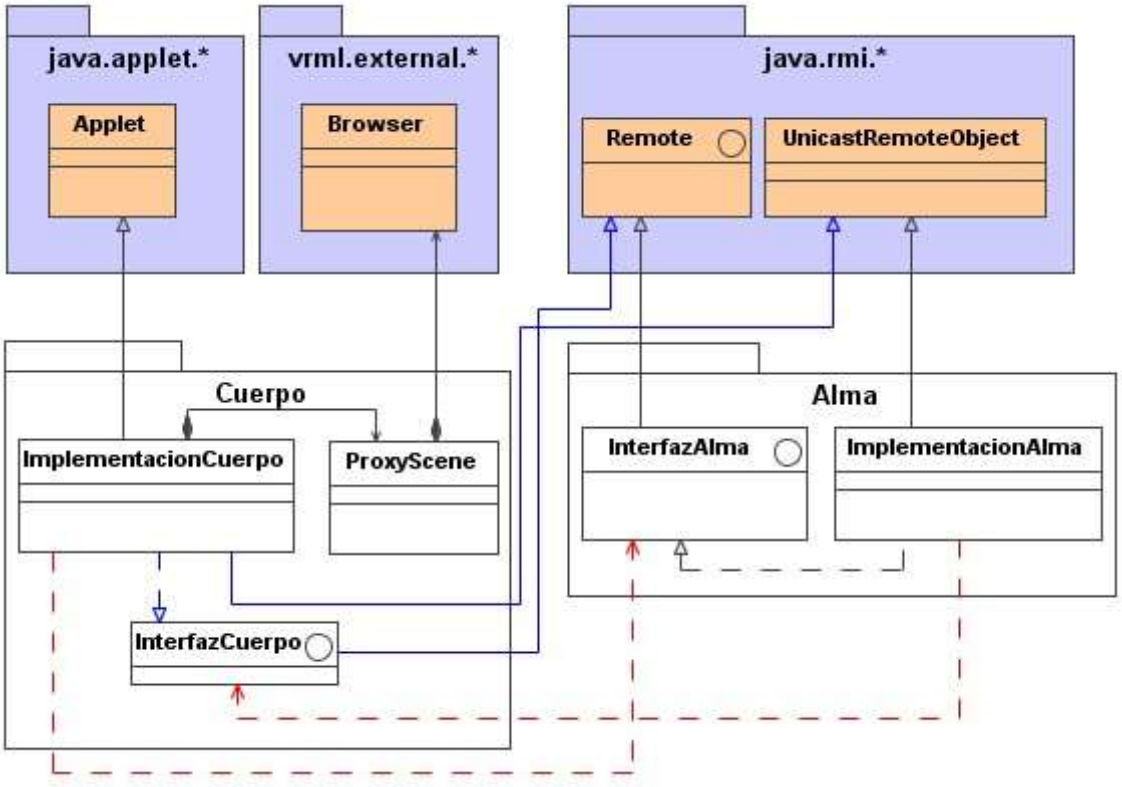


Figura 4.9 Diagrama de paquetes de clases genéricas del modelo Alma-Cuerpo para callbacks

En dicha figura podemos observar como el cuerpo también tiene una Interfaz Remota y cumple con las especificaciones de Java RMI para ser así mismo un objeto remoto.

Recuerdese que para optimizar las invocaciones de acuerdo a los principios de invocación se utilizan mecanismos de concurrencia tanto en el Alma, como en el Cuerpo.

Existen algunas consideraciones de seguridad en este esquema debido a que al ser el cliente un Java Applet y ejecutarse en un contexto protegido del navegador Web en la máquina del cliente (en un entrono conocido como caja de área o *sandbox*) es necesario proveer al applet de un mecanismo para que otros objetos en otras computadoras puedan invocarlos. Para lograr esto es necesaria la autorización del cliente mediante el uso de certificados digitales, preguntándole al usuario si permite las llamadas entrantes de invocaciones a su computadora. Para lograr esto es necesario firmar el applet de forma digital. El Apéndice C muestra el procedimiento para llevarlo a cabo.

4.4.3 Arquitectura de implementación del esquema 3: Almas locales

Este esquema define una optimización para reducir el número de mensajes enviados a los clientes (en el caso de retroalimentación) o el número de mensajes solicitados por los clientes (en el caso de escrutinio) para actualizar su estado.

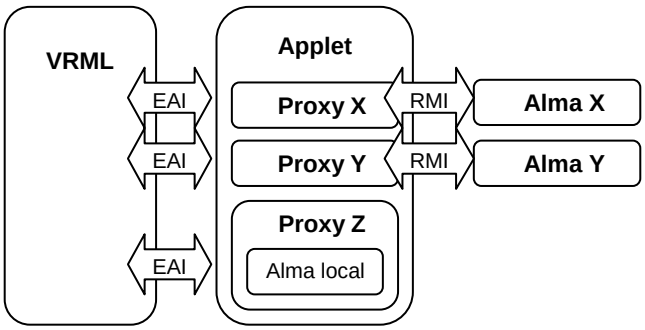
Para ello se ha definido que se puede tener almas total o parcialmente locales, con el objetivo de tener un procesamiento que cambie el estado local de los sujetos. Obviamente esto solo puede llevarse a cabo en comportamientos que pueden ser modelados de manera exacta y calculando los estados asegurando que en cada cliente (cuerpo) se obtenga el mismo resultado.

En el tipo de almas totalmente locales, el comportamiento y el estado del sujeto se encuentran almacenados en el proxy. Esto quiere decir que existirá una instancia del alma para cada uno de los clientes del Mundo Virtual Distribuido.

Sin embargo, pueden tenerse almas parcialmente locales, es decir, objetos que realizan cálculos como los mencionados, pero tienen un aspecto de sincronización ya sea por polling o por callbacks.

Dicho aspecto puede incluir, entre otras cosas, una marca de tiempo, una posición, orientación, color, etc., que será usado por un método o función que generará los nuevos estados de los cuerpos.

En las Figuras 4.10 y 4.11 se muestran los elementos que conforman este esquema:



Este esquema muestra como funcionaria en el caso de que el alma fuera completamente local. Se puede apreciar, que los objetos que implementan las almas locales forman parte del Proxy y no tienen un correspondiente objeto remoto.

Figura 4.10 Elementos del Modelo Alma-Cuerpo y sus relaciones con el esquema de almas locales

En la Figura 4.11 se muestra el caso en el cual las almas no son completamente locales y que existe un elemento de sincronización. El alma esta dividida en una parte local y en una parte remota, y que la comunicación no se da entre el proxy y el objeto remoto, sino entre el proxy y la parte local del alma; ésta a su vez se encarga de sincronizarse con su parte remota.

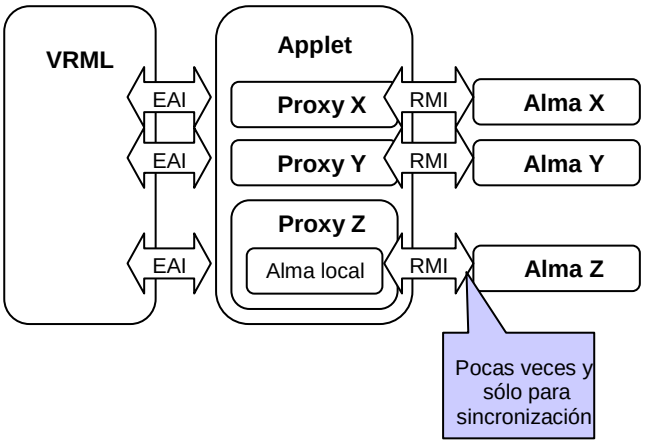


Figura 4.11 Elementos del modelo alma-cuerpo y sus relaciones con el esquema de almas semi-locales

En el diagrama de clases de la Figura 4.12 se puede apreciar la relación entre clases de implementación de este último modelo. Dónde la comunicación con el objeto remoto es para cuestiones de sincronización de estados. Sin embargo, el procesamiento se lleva a cabo en el *alma local* y es quien es realmente usada por el cuerpo para actualizar la escena.

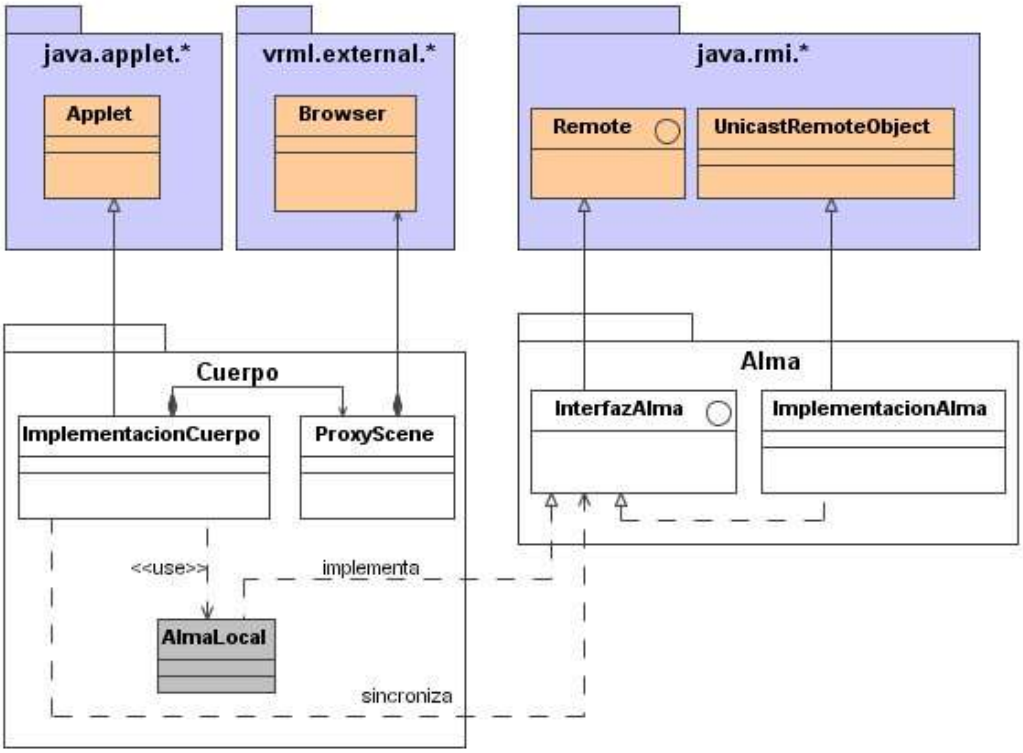


Figura 4.12 Diagrama de clases de implementación del esquema de almas semi-locales

4.5 Conclusiones

La arquitectura que se propone tiene la característica que es flexible, en el sentido que es posible intercambiar tecnologías usadas para el desarrollo de los aspectos tanto remotos como de despliegue. Es posible usar Objetos CORBA en vez de Objetos RMI y también usar algún escenario de 3D como Java 3D o VRML. Más aún, gracias al nivel de estandarización de estas tecnologías es posible integrar soluciones heterogéneas que incluyan más de una tecnología.

Es importante no olvidar las secciones críticas en la modificación de los estados, sobre todo en las actividades de concurrencia de los objetos remotos.

El uso de este tipo de arquitecturas esta abierta para la implementación de una gran cantidad de patrones de diseño usados en Ingeniería de Software.

En este capítulo se describe el análisis y diseño del caso de estudio para llevar cabo la implementación de cada uno de los modelos de sincronización propuestos al modelo alma-cuerpo. Se muestran los resultados de las pruebas realizadas a cada caso y un esquema comparativo de los mismos.

Capítulo 5: Caso de Estudio

5.1 Definición del caso de estudio

Para el caso de estudio sobre el cual se trabajarían los modelos se revisaron diferentes propuestas de mundos virtuales para demostrar cada modelo, que iban desde elementos que modelan comportamientos naturales hasta posibles entornos de conversación en 3D. Era importante definir un caso de estudio que se apegara a las diferentes condiciones que podrían existir en un Mundo Virtual más complejo, es decir, que contara con elementos con los que un usuario que habita el mundo virtual pueda interactuar, con leyes físicas que pudieran modelarse matemáticamente y con elementos cuyo cambio de estado pueda percibirse de manera visual.

Se escogió utilizar el conocido juego de billar llamado *carambola* debido a que cumple con las características básicas para la explicación de cada modelo propuesto. Además que posee elementos físicos sencillos de manipular y modelar por computadora, y que decir de la física implicada para modelar el movimiento, rebotes, velocidad y fricción de las bolas.

Los códigos fuentes relacionados con las implementaciones del caso de prueba para cada esquema se encuentran en un CD-ROM complementario a esta tesis.

5.1.1 Billares en la web

Haciendo un análisis de otros juegos de billar en Internet se encontró que la mayoría de ellos muestran la vista superior de la mesa de billar (véase Figura 5.1) y llevan a cabo el procesamiento del movimiento de las bolas cambiando de coordenadas las bolas y pintando las diferentes posiciones de las bolas al girar.

http://www.cyberjuegos.com

http://www.candystand.com

Figura 5.1 Billares en la web

Se observó que los protocolos de sincronización de dichos juegos hacían que el estado de los diferentes clientes se actualizara de manera parcial, esto es, se presentaban algunos de los siguientes casos de inconsistencia:

a) El cliente que esperaba su turno no visualizaba el movimiento del taco del rival.
b) Las jugadas ya habían terminado en un cliente mientras que en los demás comenzaban a realizarse.

De lo anterior se puede concluir que dichas demostraciones no constituyen un mundo virtual por definición, ya que no se puede apreciar la tridimencionalidad en dichos ejemplos ni que los clientes pudieran manipular la escena. Además que sus protocolos tampoco son en tiempo real debido a la gran inconsistencia de las diferentes escenas.

5.1.2 Carambola

Se utilizó la herramienta de modelado 3DStudio de Autodesk para modelar las mesas y el taco, de manera individual, una vez modelado se exportaron los archivos a VRML, es decir, archivos .WRL.

Los archivos escena .WRL que describen las características físicas de cada elemento: taco, bolas y mesa, se describen en los anexos de esta tesis.

Los elementos que poblaran el Mundo Virtual son (como se muestra en la Figura 5.2):

- Mesa de Billar (con archivo de textura de madera)
- Taco
- Bola roja
- Bola blanca
- Bola amarilla

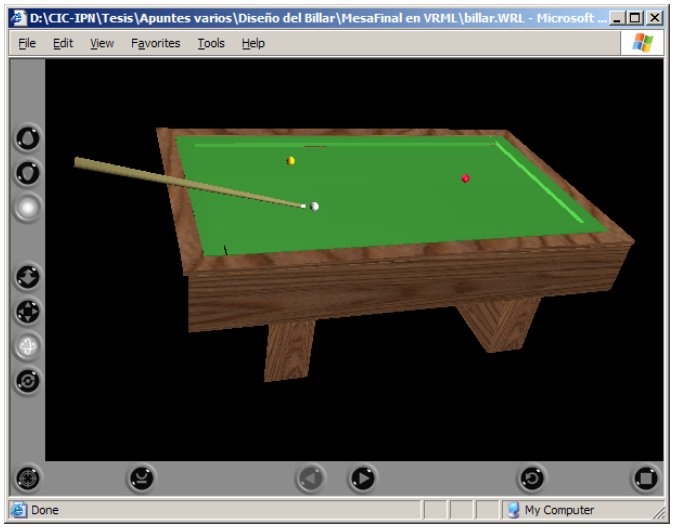


Figura 5.2 Mundo Virtual del Caso de Prueba

Una vez definido el caso de estudio sobre el cual se trabajará, el siguiente paso es identificar los elementos sujetos a sincronización de estado, es decir, los elementos que utilizando los diferentes modelos y la manipulación por parte del los usuarios, actualizarán su estado en la computadora de cada usuario. Estos elementos, dado el presente caso de estudio, son únicamente las bolas y el taco.

5.2 Casos de uso

Dadas las condiciones de infraestructura y arquitecturas definidas en el capitulo anterior se decidió probar los modelos propuestos con 3 usuarios participando al mismo tiempo.

En el caso de uso mostrado en la Figura 5.3 se ilustran las actividades que los usuarios realizarán en el ambiente virtual definido como caso de estudio.

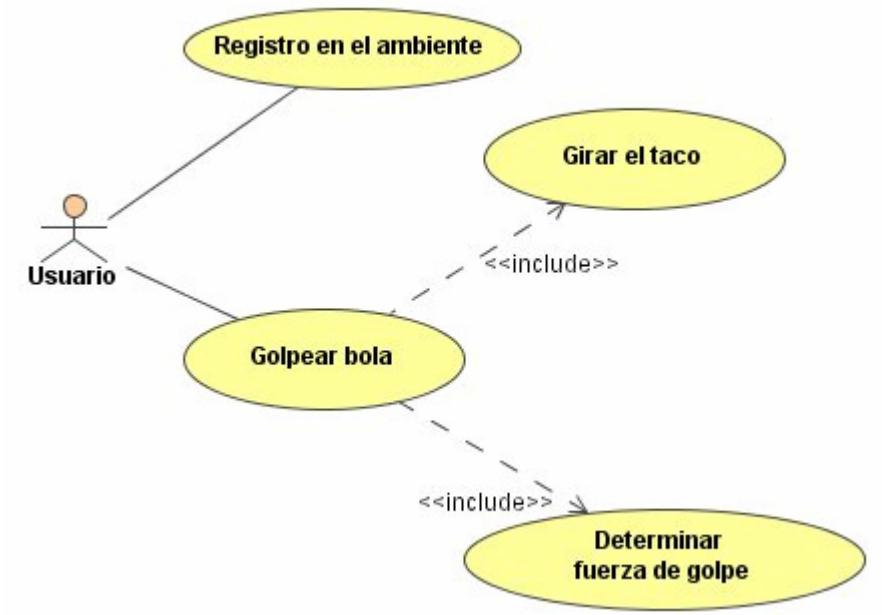


Figura 5.3 Casos de uso del usuario del Mundo Virtual Distribuido

La descripción de los casos de uso, es la siguiente:

Registro en el ambiente: El ambiente necesita que los usuarios que vayan accediendo al sitio Web proporcionen su nombre como mecanismos de identificación. Esto le sirve al ambiente para asignar los turnos de los usuarios registrados.

El ambiente necesita de manera obligatoria tres usuarios para comenzar su ejecución y cuando sea el turno de un usuario se activarán los controles visuales que puede manipular para poder golpear la bola que le es asignada.

Golpear la bola: El usuario “golpea” la bola con el taco puesto en la dirección y con la magnitud de fuerza deseada.

Girar el taco: El usuario necesita un mecanismo para poder mover el taco (girarlo alrededor de su bola asignada) y así determinar la dirección en la cual se moverá la bola que golpeará.

Determinar fuerza del golpe: El usuario podrá manipular la magnitud de fuerza con la golpeará su correspondiente bola. Para ello, el ambiente presentará un mecanismo para que el usuario pueda proporcionar esa información.

5.2.1 Flujo de eventos de los casos de uso principales del Mundo Virtual Distribuido

Registro en el ambiente

Flujo principal:

Acciones del usuario	*Acciones del sistema*
Acceder a la aplicación vía Web	El servidor Web (http) despacha la página de inicio y de registro
Coloca su nombre	El ambiente registra el nombre en el servidor y le asigna una bola (turno)
Espera por los demás participantes	La aplicación devuelve el control a los clientes (de forma diferente dependiendo del esquema) hasta que se hayan registrado 3 participantes. Los demás participantes se consideran observadores. El sistema posiciona el taco sobre la bola en turno y la aplicación cliente desbloquea los controles de movimiento solo del usuario en turno.
Selecciona velocidad y fuerza del golpe y la rotación del taco.	

Golpear bola

Flujo principal:

Acciones del usuario	*Acciones del sistema*
El usuario selecciona fuerza del golpe	La aplicación cliente del usuario en turno cambia el valor de la fuerza de acuerdo al indicado en el control visual de la interfaz gráfica. El valor

	de la fuerza es enviado al servidor invocando un método y pasándolo por valor como argumento.
El usuario selecciona un valor de rotación del taco	La aplicación cliente del usuario en turno cambia el estado de rotación del taco y envía esos datos al servidor. Cada cliente (dependiendo del esquema) actualizará su Mundo Virtual con la nueva rotación del taco.
Después de haber seleccionado los valores el usuario puede oprimir el botón para golpear la bola.	La aplicación cliente (dependiendo del esquema utilizado) actualizará el Mundo Virtual con el movimiento de las bolas.

5.2.2 Diseño de la Interfaz de usuario

El diseño de la Interfaz se divide en tres partes: el diseño de la página HTML, el diseño de los elementos visuales que habitarán el Mundo Virtual y el diseño de la interfaz de colaboración.

El diseño de la página incluye los elementos del Mundo Virtual Distribuido, el despliegue de la escena con el visor de VRML y el Java Applet como Interfaz de colaboración como lo muestra la Figura 5.4:

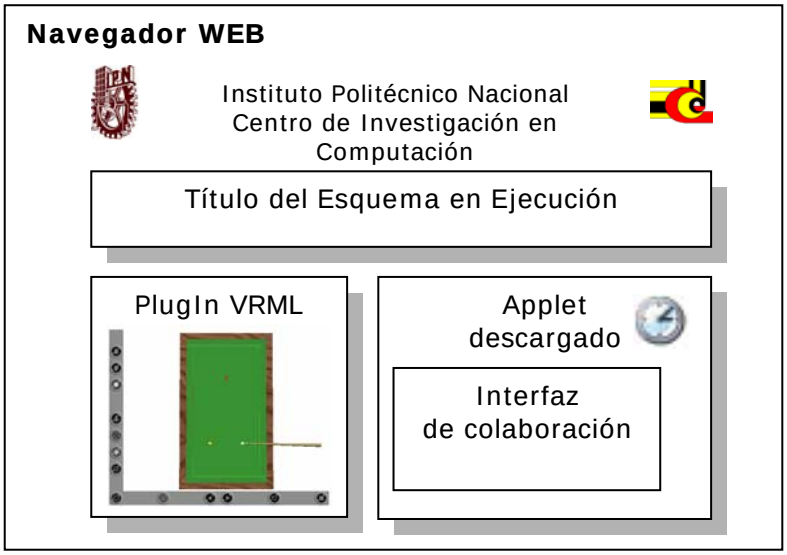


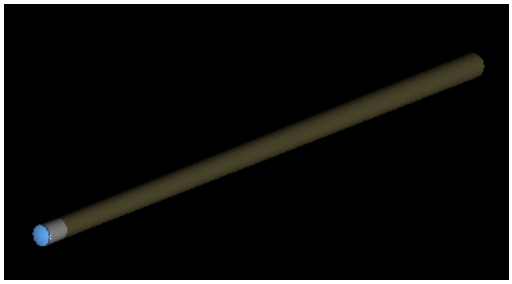
Figura 5.4 Diseño de la interfaz Web HTML

El visualizador de VRML (*plugin*) recomendado es *Cortona VRML* o *Cosmo Placer* debido a su estabilidad y compatibilidad con las APIs de VRML.

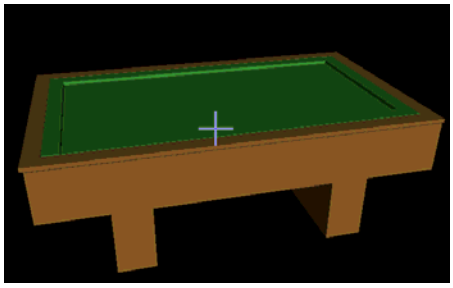
5.2.3 Diseño de los elementos del Mundo Virtual

Se tienen que definir y diseñar los elementos tridimensionales que poblarán el Mundo Virtual del caso de prueba. Algunos elementos no requieren actualización ya que por su naturaleza no sufren ningún cambio de manera directa por el usuario o por su comportamiento físico. Por ejemplo, la mesa de billar es un elemento estático que no sufre modificaciones durante la ejecución del Mundo Virtual Distribuido (Véase Figura 5.5).

El taco es un elemento que actualizará su posición respecto a la bola en juego y que el usuario indicará el ángulo de rotación adecuado para golpearla.



Elemento TACO: Requiere actualización



Elemento MESA: No requiere actualización

Figura 5.5 Diseño de elementos del MVD

Finalmente, los elementos que el usuario no modifica en posición de manera directa, pero si al ejecutar el golpe con el taco, son las bolas de billar. Ya que la posición de cada una en el plano físico de la mesa de billar se verá modificado durante el procesamiento de simulación de movimiento y colisiones entre la mesa y las demás bolas.

En la Figura 5.6 se muestran todos los elementos integrados en el mundo virtual. La actividad de identificación de “sujetos”, “cuerpos” y “almas” depende de la definición de los casos de uso y de la identificación del comportamiento de los elementos.

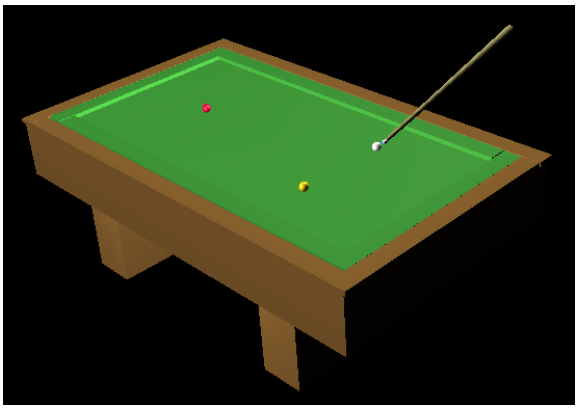


Figura 5.6 Mundo Virtual con todos los elementos del caso de estudio

Es importante mencionar que el visualizador de la escena VRML proporciona las herramientas para navegar en el mundo, respecto a la vista del usuario, es decir, acercarse, alejarse, rotar la escena, mostrar diferentes vistas, etc., como si se tratara del lente de una cámara que se mueve a las órdenes del usuario.

Una vez determinados los elementos y analizado cuáles sufrirán actualizaciones por el usuario, se define a continuación la interfaz de colaboración.

5.2.4 Diseño de la Interfaz de usuario colaborativa

La interfaz de usuario colaborativa es la herramienta visual de cada cliente para ejecutar acciones sobre el Mundo Virtual Distribuido. Para el caso de estudio los elementos de interacción con el ambiente son: el registro, el movimiento del taco, el establecimiento de la fuerza del golpe y finalmente golpear la bola. Después de esa última acción el ambiente cambiará el estado únicamente de las bolas hasta que se active la interfaz para quien tenga el siguiente turno.

En la Figura 5.7 se muestra el diseño del Java Applet de colaboración que usa componentes del paquete AWT (*Abstract Windows Toolkit*, Herramientas de Ventanas Abstractas)

Figura 5.7 Diseño de la Interfaz de usuario colaborativa (Java Applet)

Existe un elemento visual donde se desplegarán los mensajes del ambiente a manera de bitácora para monitorear la actividad de los objetos remotos y de las acciones de los otros usuarios en el Mundo Virtual Distribuido, dicho elemento se denomina Mensajes del Ambiente.

A continuación se realizarán las implementaciones de cada esquema propuesto anteriormente con este caso de estudio, lo cual servirá para hacer un análisis comparativo del desempeño de cada esquema y considerar su uso ante diferentes situaciones dependiendo del comportamiento y utilidad de cada elemento del mundo.

5.3 Implementaciones

Para comprobar los esquemas propuestos se llevaron a cabo cuatro implementaciones del caso de estudio, se escogió el modelo básico para la primera implementación de referencia y a partir de ahí, se tomaron las mejores optimizaciones de cada esquema (descritos en el Capítulo 3) para llevar a cabo la implementación de referencia de cada uno y de esta manera poder evaluar sus características de desempeño y uso.

A continuación se ilustran las partes medulares de cada implementación, mostrando los diagramas de clases y piezas de código, relacionadas con la forma de actualizar los elementos del mundo virtual. Se omitirán detalles relacionados con la implantación de objetos remotos y obtención de referencias remotas, por tratarse de temas de Java RMI.

En caso de que se requiera mayor detalle en el código fuente, refiérase al CD-ROM adjunto.

5.3.1 Esquema 1: Escrutinio - Modelo Básico

En la Figura 5.8 se muestra el diagrama de clases de la implementación del modelo básico de escrutinio que se propone en el modelo Alma-Cuerpo.

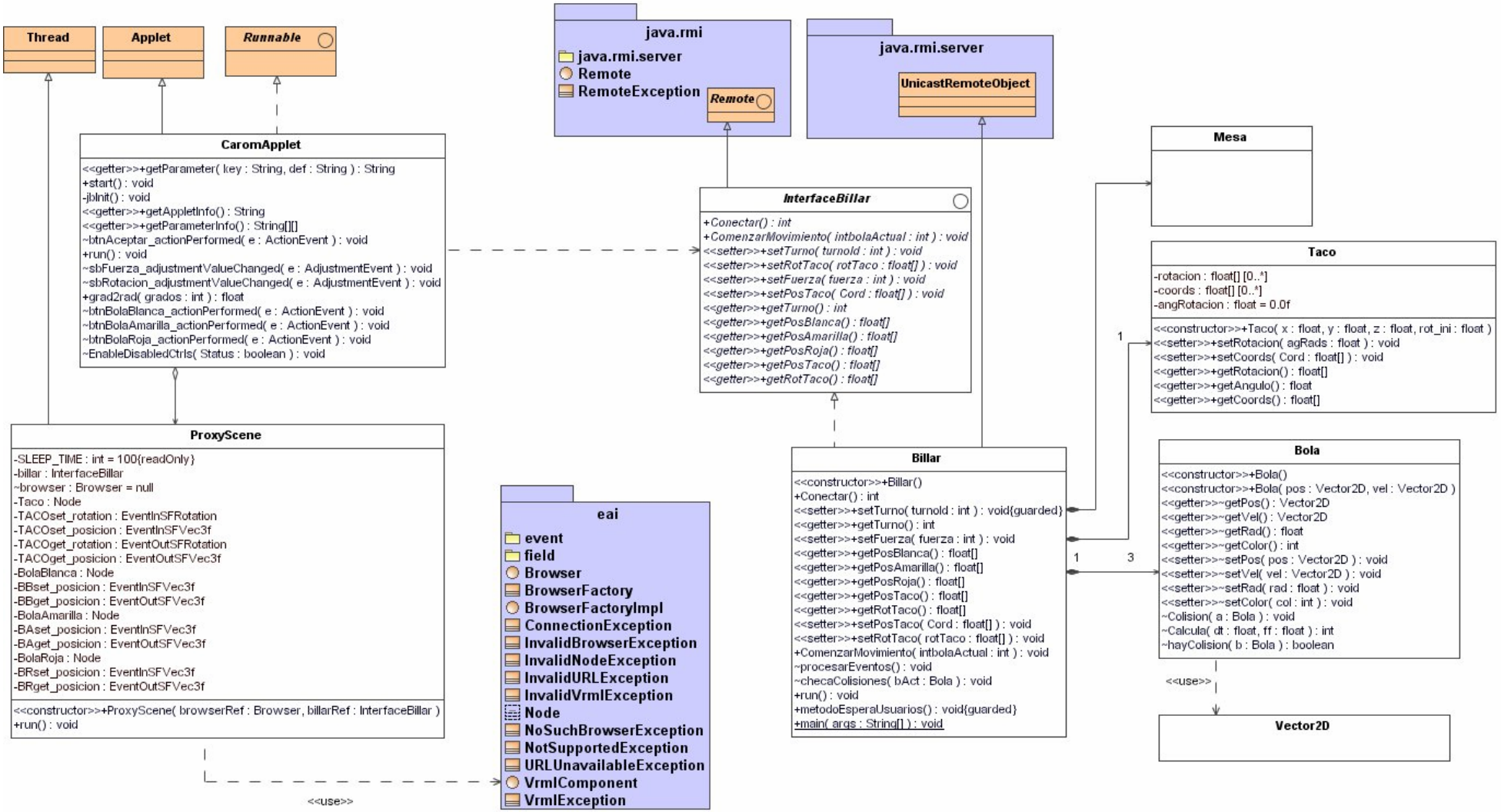


Figura 5.8 Diagrama de clases de la implementación del modelo básico de escrutinio.

A continuación se explica la función de cada clase relacionándola con el modelo Alma-Cuerpo y de esa manera entender su implementación dentro del mismo.

En el cliente (cuerpo):

CaromApplet: Es el programa cliente, que forma parte del Cuerpo y mantiene la relación con el alma (objeto remoto) y con la escena VRML a través de la clase ProxyScene.

ProxyScene: Es la representación de la escena VRML, se encapsulan ahí los nodos para modificarla y obtener información de la misma. Es la que contiene la secuencia de actualización de la escena al invocar los métodos remotos del objeto remoto a través de la interfaz remota de Java RMI *InterfazBillar*, haciendo uso del API de la EAI de VRML para actualizar los elementos en el visualizador.

InterfazBillar: Interfaz que utiliza el cliente para mantener la referencia remota al objeto e invocar los métodos de actualización de estado.

En el servidor (alma):

InterfazBillar: Define todos los métodos remotos que serán invocados por los clientes para actualizar los mundos virtuales de cada uno y que serán implementados por el Objeto Remoto *Billar*. (Véase Figura 5.9)

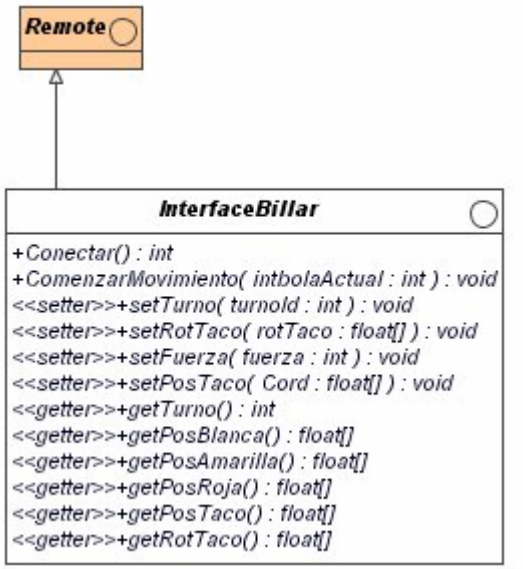


Figura 5.9 Diagrama de clase de la interfaz remota

Billar: Define al alma y contiene la implementación de los métodos definidos en la interfaz remota y de otro conjunto de métodos de procesamiento y comportamiento del alma. Para efectos del caso de estudio, en esta clase se lleva a cabo el procesamiento del cálculo de las posiciones de las bolas cuando

una ha sido golpeada por el Taco. Uno de los métodos se denomina ComenzarMovimiento() y se ejecuta cuando el Java Applet invoca ese método remoto ante la acción del usuario al que le corresponde golpear la bola.

Vector2D: Clase utilitaria para modelar los vectores de movimiento de las bolas de billar.

Bola: Forma parte del billar, existen 3 instancias de esta clase, una por cada bola del juego. Esta clase utiliza la clase Vector2D para calcular su propio movimiento.

Taco: Representa al taco y mantiene información de su posición y rotación.

Mesa: Define las dimensiones de la mesa para poder realizar los cálculos de los demás elementos.

Fragmentos de código fuente de actualización de estados

El bloque de código más importante de este modelo es el de la clase ProxyScene, ya que actualiza los cuerpos en el cliente accediendo al alma (invocando los métodos remotos) de acuerdo al modelo básico propuesto en el esquema 1 del capitulo 3.

La clase ProxyScene sirve de representante de la escena de VRML, ya que contiene como atributos la definición de cada nodo de cada elemento que va a actualizar, esto lo logra utilizando el API de la EAI (Véase línea 1, 9, 15 de la Figura 5.10).

```
1  import vrml.external.*;
2            ...
3  import java.lang.Thread;
4                ...
5  public class ProxyScene extends Thread {
6    private final int SLEEP_TIME = 100;
7    private InterfaceBillar billar;
8    // Variables de la escena
9    Browser browser = null;
10   ...
11
12     public ProxyScene(Browser browserRef, InterfaceBillar billarRef) {
13         browser = browserRef;
14         billar = billarRef;
15         BolaBlanca = browser.getNode("BolaBlanca");
16         ...
17     }
```

Figura 5.10 Bloque de código de la clase ProxyScene

El constructor de la clase ProxyScene (Figura 5.10) recibe la referencia a la escena VRML y la referencia al objeto remoto (alma) para que pueda invocar los métodos de actualización de estados.

La Figura 5.11 ilustra el bloque de código con el algoritmo de actualización del mundo virtual, el cual consiste en un ciclo de actualizaciones (línea 20) en un mismo hilo de ejecución (línea 19) cada determinado tiempo SLEEP_TIME (línea 28). Durante cada ciclo se llevan a cabo las invocaciones al objeto remoto billar y las actualizaciones a la escena VRML (líneas 22 a 26). Esta secuencia define la actualización de elementos, por lo que cada invocación de actualización debe esperar a que termine la anterior debido a su naturaleza síncrona.

```
18
19   public void run(){
20         while(true) {
21                       try {
22                         BBset_posicion.setValue(billar.getPosBlanca());
23                         BAset_posicion.setValue(billar.getPosAmarilla());
24                         BRset_posicion.setValue(billar.getPosRoja());
25                         TACOset_posicion.setValue(billar.getPosTaco());
26                         TACOset_rotation.setValue(billar.getRotTaco());
27                           System.out.println("Update escene...");
28                           Thread.sleep(SLEEP_TIME);
29                       } catch(Exception e) {
30                           System.out.println("Error:" + e);
31                         break;
32                    }
33         }
34   }
35 }
```

Figura 5.11 Bloque de código de actualización del modelo básico

5.3.2 Esquema 1: Escrutinio - Modelo Optimizado

A continuación se explican los detalles de implementación del modelo optimizado de escrutinio. Como el modelo anterior presentaba demasiada latencia en las invocaciones secuenciales de acceso al objeto remoto, se utilizó un mecanismo de concurrencia denominado “repositorio de objetos concurrentes”.

En el diagrama de clases de la Figura 5.12 define la implementación del modelo optimizado del esquema 1 propuesto en el capitulo 3. Sólo se explicarán las clases nuevas (las de la parte inferior de la Figura 5.12) y las diferencias respecto al modelo anterior.

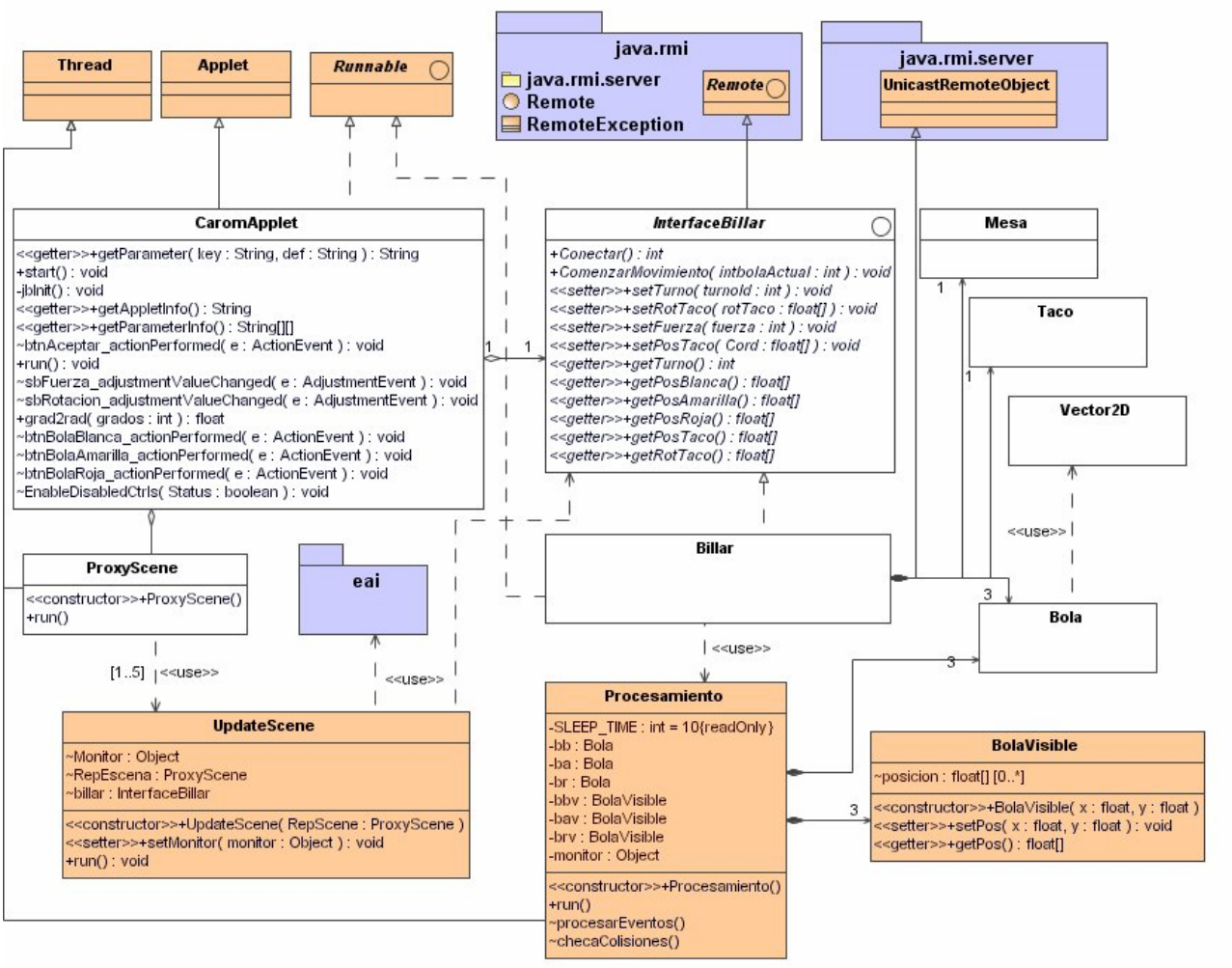


Figura 5.12 Diagrama de clases de la implementación del modelo optimizado de escrutinio.

En el cliente (cuerpo):

ProxyScene: Se encarga de crear un conjunto de objetos finitos cuya función será actualizar el objeto remoto de manera concurrente. Estos objetos son del tipo UpdateScene y ellos son los que contiene la referencia al objeto remoto (alma) y al visualizador de VRML. La función ahora es controlar la actividad de cada objeto de actualización.

UpdateScene: Definen hilos de ejecución que actualizan el estado de los cuerpos en el visualizador de VRML a través del API de EAI. Se crea un número de objetos finitos de este tipo y el planificador de concurrencia de la Máquina Virtual Java los planifica para activarse de acuerdo al algoritmo que se define en la clase ProxyScene.

En el servidor (alma):

Procesamiento: En esta clase se delega la responsabilidad de llevar a cabo el cálculo de los movimientos de las bolas de billar, como un hilo de ejecución lanzado por el objeto remoto (cada que el usuario en turno, golpee la bola, este método es activado por el Java Applet). Esto hace que el objeto remoto se encuentre disponible para aceptar invocaciones y realizar otros procesos.

BolaVisible: Esta clase sirve como mecanismo de seguridad al acceso de los valores de la posición de cada bola, ya que estas bolas son las que dan la información de su estado a los clientes, mientras que la clase Bola es donde se va almacenando la información de cada bola durante el procesamiento.

Fragmentos de código fuente de actualización de estados

A diferencia del modelo anterior, la clase ProxyScene se encarga ahora de gestionar los objetos que tienen la capacidad de actualizar los estados de los elementos del mundo virtual, los cuales son un conjunto de elementos finitos con un hilo concurrente por cada uno en un repositorio.

La Figura 5.13 muestra la ceración de dicho repositorio, además ilustra el algoritmo para administrar el ciclo de vida de la concurrencia de cada uno, con un mecanismo aleatorio de “dormir y despertar” los hilos de actualización (línea 23 a 27). Como se puede apreciar en la línea 24 de la Figura 5.13, se utiliza el mecanismo de Monitores del planificador de hilos de la Máquina Virtual Java, para despertar a los hilos de actualización.

```
3  public class ProxyScene extends Thread {
4    ...
5
6    public void run(){
7            ...
8          Object [] monitores = new Object[5];
9          for (int j=0; j<monitores.length; j++) monitores[j]=new Object();
10
11         UpdateScene updateScene1 = new UpdateScene(this);
12                 updateScene1.setMonitor(monitores[0]);
13                                 ...
14         UpdateScene updateScene5 = new UpdateScene(this);
15                 updateScene5.setMonitor(monitores[4]);
16
17         updateScene1.start();
18             ...
19         updateScene5.start();
20
21         while(true) {
22                     try {
23                         synchronized(monitores[i]){
24                             monitores[i].notifyAll();
25                         }
26                         Thread.sleep(SLEEP_TIME);
27                         i = (++i) % 5;
28                     } catch(Exception e) {
29                       System.out.println("Error:" + e);
30                       break;
31                   }
32         }
33   }
```

Figura 5.13 Código de creación del repositorio de objetos de actualización

La clase UpdateScene mantiene las referencias al objeto remoto RMI y al visualizador de VRML mediante las clases de la EAI (líneas 44 a 46 de la Figura 5.14).

Cada objeto de la clase UpdateScene se encarga de llevar a cabo lo que en el modelo básico realizaba la clase ProxyScene. Esto significa que en este modelo existen mas llamadas concurrentes de actualización. Es decir, existe un número finito de objetos que están invocando de manera concurrente los métodos remotos del objeto remoto –alma- (líneas 55 a 58 de la figura 5.14).

Después de llevar a cabo las actualizaciones, empieza de nuevo el ciclo y de inmediato el hilo en ejecución se detiene (línea 53 de la Figura 5.14) y queda en espera que la clase ProxyScene lo haga continuar su ejecución línea 24 de la Figura 5.13).

```
38 class UpdateScene extends Thread {
39
40   Object Monitor;
41
42    ProxyScene RepEscena;
43    InterfaceBillar billar;
44    public UpdateScene (ProxyScene RepScene) {
45       RepEscena = RepScene;
46       billar= RepEscena.billar;
47    }
48
49    public void run() {
50              while(true){
51                   try {
52                      synchronized(Monitor) {
53                         Monitor.wait();
54                      }
55                      RepEscena.BBset_posicion.setValue(billar.getPosBlanca());
56
57                      RepEscena.TACOset_posicion.setValue(billar.getPosTaco());
58                      RepEscena.TACOset_rotation.setValue(billar.getRotTaco());
59                   } catch(Exception e) {
60                      System.out.println("Error:" + e);
61                   }
62             }
63    }
64
65 }
```

Figura 5.14 Código de actualización de los elementos del mundo virtual.

En este modelo se añadieron optimizaciones del lado del objeto remoto, ya que el alma, además de gestionar el turno de los clientes, tenía que llevar a cabo el procesamiento del movimiento de las bolas de billar. Por lo que se delegaron esas tareas en una nueva clase, que también es concurrente y que implementa un mecanismo de protección de acceso a los valores de la bolas (en el modelo básico, el mismo valor expuesto a la invocación remota de métodos, era el que estaba en uso y cambio constante durante el procesamiento). De las líneas 10 a la 13 de la Figura 5.15 se ilustra la creación, lanzamiento y espera del procesamiento del movimiento de las bolas de billar, cuando el usuario en turno golpea la bola.

```
1  import java.rmi.*;
2  import java.rmi.server.*;
3  import java.lang.Thread;
4
5  public class Billar extends UnicastRemoteObject implements InterfaceBillar , Runn
6   ...
7       ...
8         public void ComenzarMovimiento(int intbolaActual) throws RemoteException(
9                 ...
10              procesarEventos = new Procesamiento(bb,ba,br,bbv,bav,brv,monitor);
11              try{
12                procesarEventos.start();
13                procesarEventos.join();
14                System.out.println("ComenzarMovimiento: despues Join");
15              } catch(Exception e){
```

Figura 5.15 Código del uso del objeto Procesamiento.

5.3.3 Esquema 2: Retroalimentación (callbacks)

En el esquema anterior aunque incrementa la consistencia de actualización, también incrementa considerablemente el uso del procesador, debido a las invocaciones constantes al alma aunque su estado no haya cambiado. Es por ello, que se pensó en la manera de reducir el uso de procesador y acceso a la red mediante actualizaciones generadas únicamente por el servidor a cada cambio de estado, retroalimentando a cada cliente dicho estado modificado.

En la Figura 5.16 se muestra el diagrama de clases correspondiente a la implementación de este esquema y se puede apreciar que el Java Applet que forma parte del cuerpo, expone implementaciones de métodos remotos para que pueda ser invocado por el objeto remoto –alma- cuando se requiera.

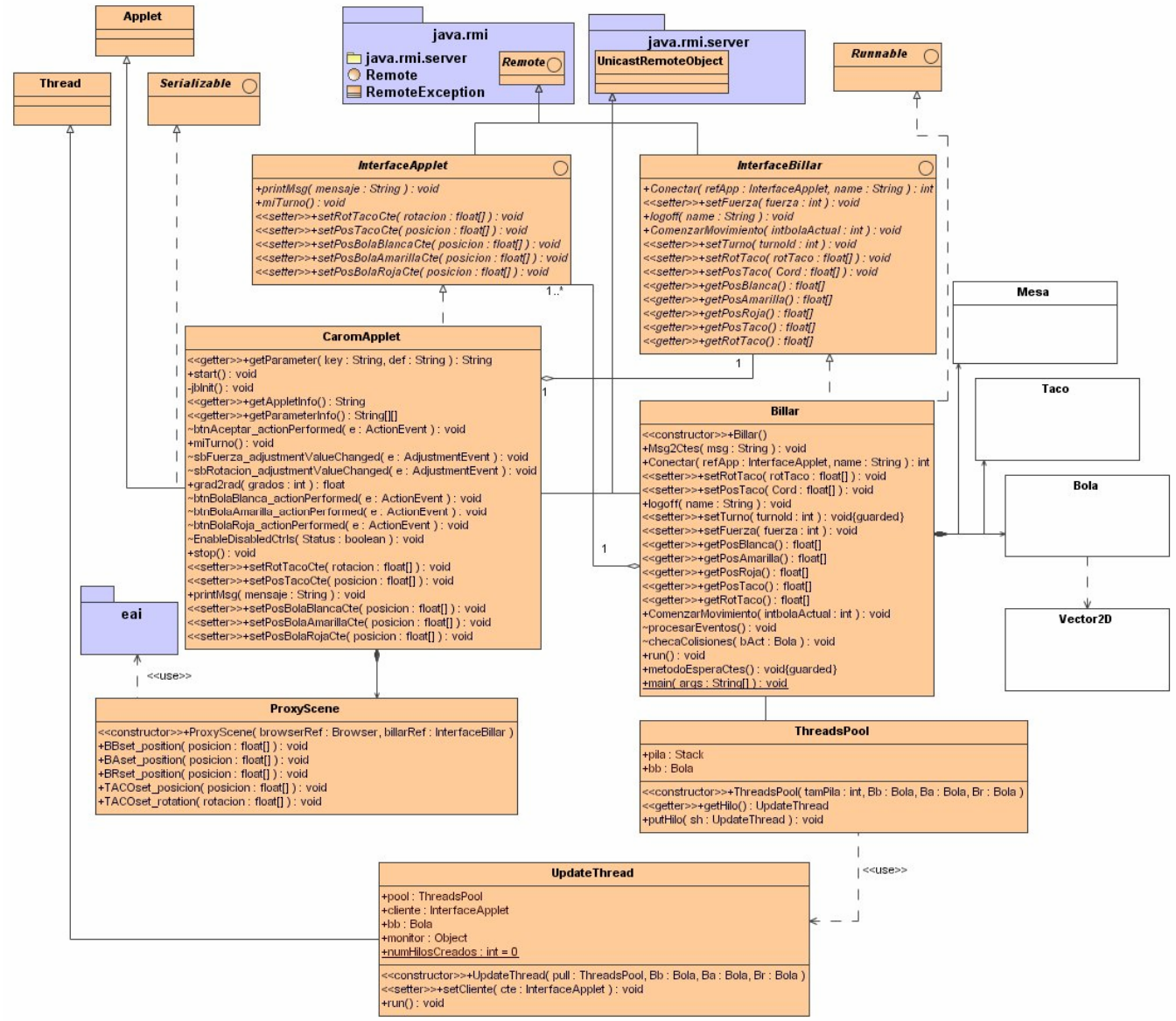


Figura 5.16 Diagrama de clases de la implementación del esquema de retroalimentación.

A continuación se explican las clases nuevas y las modificaciones en el comportamiento.

En el cliente (cuerpo):

InterfazApplet: Es la interfaz remota del Java Applet, es decir, del cliente. Expone el conjunto de métodos que el servidor podrá invocar para modificar el estado de los elementos del cliente.

CaromApplet: Además de continuar con la funcionalidad anterior, ahora implementa los métodos remotos que serán invocados por el servidor. Esta clase deberá firmarse digitalmente para que el Navegador de Internet pueda permitir las invocaciones entrantes en la máquina cliente (Véase Apéndice C). Los métodos de actualización se exponen como fachadas ya que realmente usan a la clase ProxyScene para actualizar la escena (patrón Fachada [21]).

ProxyScene: Anteriormente esta clase mantenía la referencia al alma haciendo iteraciones sobre sus métodos remotos para actualizar el estado. En esta implementación cambia de manera radical, implementando métodos tipo "set" o de actualización simple de la escena por cada nodo, haciendo uso de la EAI de VRML para lograrlo. Dichos métodos son invocados por el Java Applet en la implementación de sus métodos remotos.

En el servidor (alma):

InterfazBillar: Expone nuevos métodos que registran las referencias de los clientes que servirán para invocar los métodos de los mismos en cada actualización.

Billar: Mantiene la lista de los clientes, que ellos mismos pasan a esta clase al invocar el método de registro. Escucha por los eventos de actualización cuando los clientes mueven el taco o golpean la bola, en este último, optimiza el procesamiento del cálculo implementando un esquema de repositorio de hilos.

ThreadPool: Es un repositorio de hilos Java instanciados, básicamente la implementación de una estructura de datos tipo Pila. Se encarga de guardar instancias de objetos de actualización de estados del tipo UpdateThread. Esto con la finalidad de minimizar el tiempo de creación /destrucción de hilos.

UpdateThread: Implementa un hilo de ejecución instanciado y listo para ser activado. Recibe la referencia del cliente sobre el cual invocará sus métodos remotos de actualización. Posterior a la actualización se desactivará y retornará al repositorio listo para ser utilizado nuevamente por el ciclo de actualización de clientes de la clase *Billar*.

Fragmentos de código fuente de actualización de estados

Los siguientes fragmentos de código muestran los procedimientos para llevar a cabo la retroalimentación, códigos desde el registro, actualización cliente a cliente y actualización de la escena.

Para llevar a cabo la retroalimentación, el objeto remoto tiene que conocer la referencia de sus clientes, en la Figura 5.17 hay un fragmento de código del cliente (cuerpo) que invoca un método remoto llamado Conectar() (línea 5), pasándole su propia referencia para que pueda ser invocado desde el servidor posteriormente.

```
2  ...
3       try {
4               textArea.append("Espere por los otros participantes...");
5               miID = billar.Conectar(this, nombreUsr);
6          } catch(RemoteException errConx){
7               System.out.println("Error:"+errConx);
8          }
9
10         switch(miID) {
11            case 1 : textArea.append("\nTu bola es la BLANCA.");
12                     MiBoton = btnBolaBlanca;
13                     EnableDisabledCtrls(true);
14                     break;
15                     ...
16
17            default: textArea.append("\n Solo eres OBSERVADOR.");
18                     break;
19         }
```

Figura 5.17 Registro del objeto cliente ante el objeto remoto

El objeto remoto mantiene en una lista la referencia de los clientes para poder invocar sus métodos remotos posteriormente. (Línea 10 de la Figura 5.18).

```
1      ...
2
3       public InterfaceApplet JugadorBA,JugadorBA,JugadorBA;
4       private Hashtable appletList;
5
6    ...
7           public int Conectar(InterfaceApplet refApp, String name) throws RemoteException {
8                 int myID = ID++;
9                           ...
10                  appletList.put(name,refApp);
11                        switch(myID) {
12                         case 1: JugadorBB=refApp; break;
13                         case 2: JugadorBA=refApp; break;
14                         case 3: JugadorBR=refApp; break;
15                        }
16                ...
17              Msg2Ctes("\n" + name + " entra al ambiente con el ID:"+myID+"\n");
18                return myID;
19        }
```

Figura 5.18 Método de registro en la clase Billar

Debido a que el mecanismo de actualizar los clientes es invocar sus métodos remotos en orden secuencial (línea 7 de la Figura 5.19), se utilizó un repositorio de hilos de actualización con el objetivo de reducir el tiempo de invocación entre cada cliente de la lista, delegando la tarea de invocación a hilos de ejecución, los cuales atienden a un cliente. En las líneas 11 y 12 de la Figura 5.19 se observa el mecanismo de obtención del hilo y la asignación del cliente sobre el cual se invocarán sus métodos de actualización.

```
6                 Enumeration enumer = appletList.keys();
7                 while(enumer.hasMoreElements()) {
8                   String llave = (String)enumer.nextElement();
9                   InterfaceApplet cte = (InterfaceApplet) appletList.get(llave);
10                     //usando el repositorio de hilos
11                      UpdateThread ut = pool.getHilo();
12                      ut.setCliente(cte);
13                }
14                try {
15                      Thread.sleep(SLEEP_TIME);
16                } catch(Exception e) {
17                      System.out.println("error");
18                }
```

Figura 5.19 Recuperación de hilos de actualización del repositorio

Cada hilo es responsable de invocar los métodos de actualización de un cliente en específico. (Línea 26 del bloque de código mostrado en la Figura 5.20).

```
18  public void run() {
19    while(true){
20      synchronized (monitor) {
21            if (cliente == null) try { monitor.wait(); } catch(Exception e){}
22      }
23      try {
24
25           float [] f = new float[]{bb.getPos().getX(),0,bb.getPos().getY()};
26           cliente.setPosBolaBlancaCte(f);
27           ....
28
29      } catch(Exception e) {
30           System.out.println("Srv error ROTaco:"+e.getMessage());
31      }
32      cliente=null;
33      pool.putHilo(this);
34    }
35  }
```

Figura 5.20 Hilo de ejecución responsable de la actualización de un cliente en la clase UpdateThread

Cuando el hilo finaliza de actualizar un cliente, retorna al repositorio de hilos para estar disponible y poder ser utilizado para la actualización de otro cliente. (Líneas 32 y 33 de la figura 5.20)

Los métodos de actualización del cliente que se invocan desde el objeto remoto sirven como fachadas ya que delegan la actualización de la escena a los métodos de la clase ProxyScene. La actualización de la escena es directa, mediante la invocación de un método específico con valores específicos, con

información proveniente directamente del alma, es por ello que se le llama retroalimentación.

En el bloque de código de la Figura 5.21, la clase ProxyScene, utiliza el API de EAI de VRML para actualizar la escena. A diferencia de los modelos anteriores ya no se utiliza escrutinio, hilos o ciclos para llevar a cabo la actualización.

```
4  public class ProxyScene {
5    // REPRESENTACIONES DE GEOMETRIAS DE LA ESCENA
6    private Node ...
7   ....
8
9      public ProxyScene(Browser browserRef, InterfaceBillar billarRef) {
10         browser = browserRef;
11         billar = billarRef;
12         ....
13         BolaRoja = browser.getNode("BolaRoja");
14         BRget_posicion = (EventOutSFVec3f)BolaRoja.getEventOut("translation");
15         ....
16     }
17
18   public void BBset_position(float [] posicion) {
19           BBset_posicion.setValue(posicion);
20   }
21   public void BAset_position(float [] posicion) {
22           BAset_posicion.setValue(posicion);
23   }
24   public void BRset_position(float [] posicion) {
25           BRset_posicion.setValue(posicion);
26   }
27
28   public void TACOset_posicion(float [] posicion){
29           TACOset_posicion.setValue(posicion);
30   }
31
32   public void TACOset_rotation(float [] rotacion){
33           TACOset_rotation.setValue(rotacion);
34   }
35 }
```

Figura 5.21 Actualización de la escena en la clase ProxyScene

5.3.4 Esquema 3: Almas Locales

Finalmente, la implementación de un esquema óptimo en cuanto a recursos de red, es la que modela el comportamiento local de un sujeto. Para ello se decidió tener un esquema hibrido, ya que se utiliza tanto escrutinio como retroalimentación.

Para los propósitos de este caso de estudio, el procesamiento del movimiento de a las bolas se realiza en la parte cliente, con mecanismos de espera hasta que todos los clientes terminen basados en retroalimentación y mecanismos de actualización de la escena basados en escrutinio.

Esta implementación produce inconsistencias en el estado de los sujetos de la escena, debido a que el tiempo de procesamiento en cada cliente varia

dependiendo de las características tecnológicas de las computadoras de los clientes, así como de la velocidad de conexión, cantidad de memoria, etc.

La retroalimentación se utiliza para que el servidor les comunique a los clientes (sujetos) que comiencen el procesamiento, pasándoles la pieza de información (fuerza, dirección del taco, bola en turno) para que ellos lleven a cabo el cálculo de manera local.

El cálculo se realiza en una clase local (como la que lo hacia anteriormente en el servidor) y la actualización de la escena se lleva a cabo realizando el escrutinio de los atributos de esta clase.

En la Figura 5.22 se tiene el diagrama de clases de este esquema, donde se pueden apreciar las clases nuevas en este modelo: BillarLocal, AvisarClientes que serán explicadas brevemente más adelante.

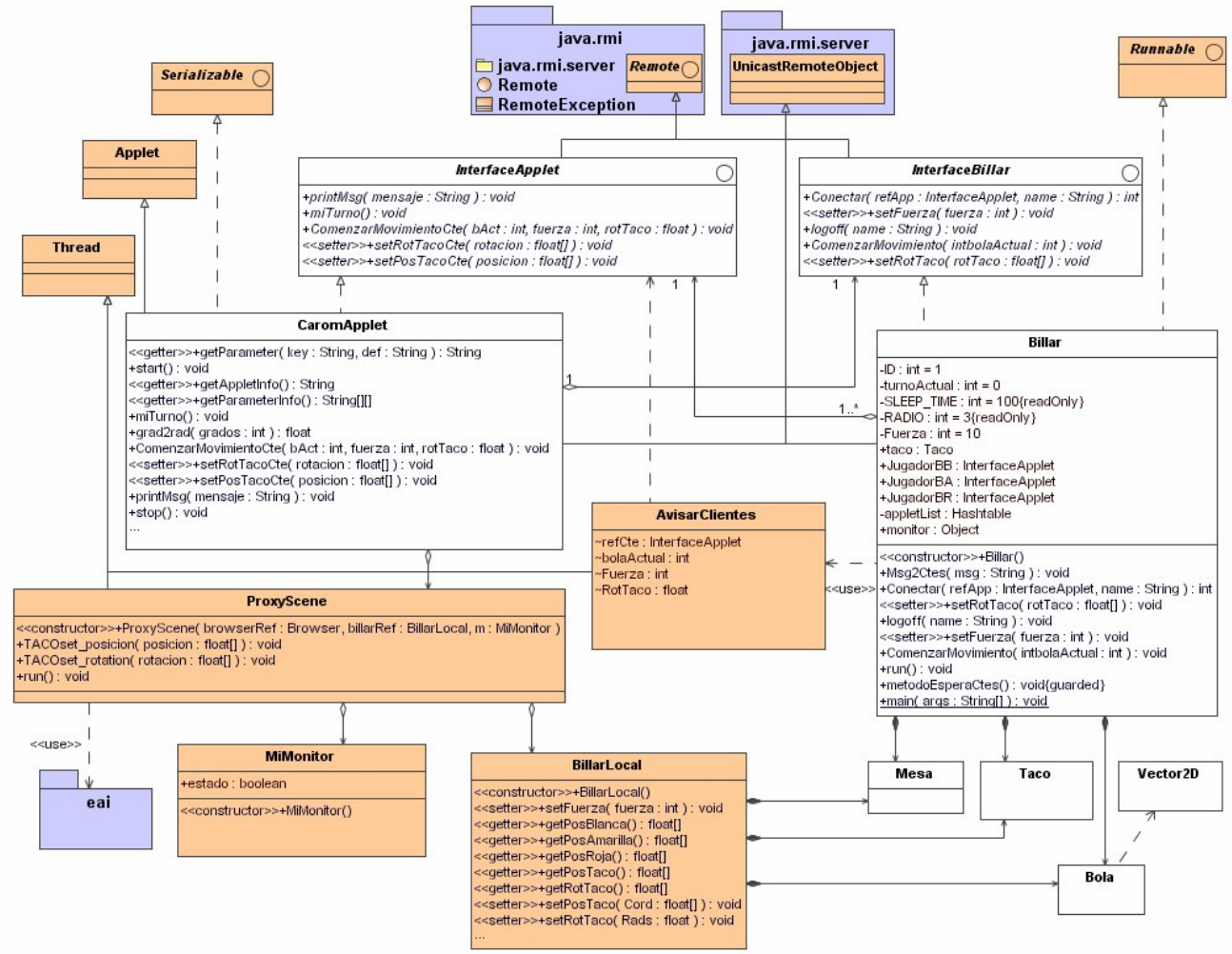


Figura 5.22 Diagrama de clases del esquema de Almas Locales

A continuación se explican las clases nuevas y las modificaciones en el comportamiento de las clases de este modelo tanto en la parte del cuerpo como en el alma.

En el cliente (cuerpo):

InterfazApplet: Reduce el número de métodos remotos creados para el esquema anterior de retroalimentación, exponiendo ahora el método *ComenzarMovimientoCte().*

Billar: Envía y recibe notificaciones del alma (clase *Billar*), implementa el método *ComenzarMovimientoCte()*, el cual recibe como parámetro la información del turno, fuerza y ángulo de golpeo del taco y hace uso de la clase *BillarLocal*, para que cada cliente pueda llevar a cabo la simulación del movimiento de las bolas.

BillarLocal: Se encarga de llevar a cabo el cálculo del movimiento de las bolas, con la información que recibe. Se realiza exactamente de la misma manera en la que se realizaba en los anteriores modelos del lado del servidor. Esta clase NO se encarga de actualizar la escena.

ProxyScene: Se encarga de realizar la actualización de la escena mediante el uso de escrutinio sobre el objeto local (*BillarLocal*), es decir, no usa invocaciones a métodos remotos como en el esquema 1.

Taco, Mesa, Bola, Vector2D: Las clases que modelan el cuerpo del sujeto y que son utilizadas para el procesamiento, ahora son utilizadas del lado del cliente, exactamente de la misma forma en que se utilizaban del lado del servidor.

MiMonitor: Es una clase utilitaria para modelar un estado tipo bandera que indica los momentos de actualizar la escena.

En el servidor (alma):

AvisarClientes: Clase responsable de notificar a los clientes registrados que es momento de realizar el procesamiento y les pasa la pieza de información necesaria para tal efecto. La notificación se realiza creando hilos de ejecución concurrentes para disminuir el tiempo de notificación entre cada cliente.

InterfazBillar: Se eliminan todos los métodos de obtención y actualización del estado, ya que el estado se encuentra propagado del lado del cliente.

Billar: La implementación del cálculo y procesamiento de actualización de estados que se realizaba anteriormente en esta clase, se delega al cliente haciendo uso de la clase *AvisarClientes*. Ahora la clase Billar funge como coordinadora de los participantes, ya que orquesta los momentos cuando los clientes deben comenzar a calcular los movimientos y esperar por su turno, esto mediante el uso de retroalimentación (callbacks).

A continuación se ilustran los fragmentos de código fuente de mayor representación en este esquema.

Fragmentos de código fuente de actualización de estados

En el código de la Figura 5.23, cuando el usuario en turno golpea la bola, el Applet (clase CaromApplet) lanza la información de este evento al objeto remoto (Línea 326, 334 y 343).

```
323  void btnBolaBlanca_actionPerformed(ActionEvent e) {
324      try {
325           EnableDisabledCtrls(false);
326              billar.ComenzarMovimiento(billar.intBOLA_BLANCA);
327      } catch (RemoteException ErrStartMovs){
328              System.out.println("Error:" + ErrStartMovs);
329      }
330   }
331    void btnBolaAmarilla_actionPerformed(ActionEvent e) {
332      try {
333           EnableDisabledCtrls(false);
334              billar.ComenzarMovimiento(billar.intBOLA_AMARILLA);
335      } catch (RemoteException ErrStartMovs){
336              System.out.println("Error:" + ErrStartMovs);
337      }
338   }
339
340   void btnBolaRoja_actionPerformed(ActionEvent e) {
341     try {
342         EnableDisabledCtrls(false);
343           billar.ComenzarMovimiento(billar.intBOLA_ROJA);
344     } catch (RemoteException ErrStartMovs){
345           System.out.println("Error:" + ErrStartMovs);
346     }
347   }
348
```

Figura 5.23 Notificación de inicio del procesamiento por parte del cliente

En la Figura 5.24 el objeto remoto recibe los valores y notifica a los clientes mediante hilos concurrentes de la clase *AvisarClientes* (líneas 16-19). También se les pasa la información necesaria para propagar el cálculo del movimiento, es decir, la fuerza del taco y el ángulo de rotación del mismo, que fueron recopilados por la invocación de los métodos remotos de la clase *Billar* cuando el usuario estableció la fuerza y la rotación del taco con el uso de la interfaz gráfica.

```
3  public class Billar extends UnicastRemoteObject implements InterfaceBillar , Runnable {
4   // private Bola bb,ba,br;  //Bolas
5
6   ...
7      public void ComenzarMovimiento(int intbolaActual) throws RemoteException{
8        ...
9        InterfaceApplet JugadorSiguiente=null;
10       Vector listaHilos = new Vector();
11       Enumeration enumer = appletList.keys();
12        while(enumer.hasMoreElements()) {
13           String llave = (String)enumer.nextElement();
14           InterfaceApplet cte = (InterfaceApplet) appletList.get(llave);
15              try {
16                AvisarClientes adv =
17                  new AvisarClientes(cte,intbolaActual,Fuerza,(-taco.getAngulo() + 3.14159264f));
18                listaHilos.add(adv);
19                adv.start();
20              } catch(Exception e) {
21                  ...
22              }
23       }
24       //Esperar por todos los clientes
25       Enumeration numHilos = listaHilos.elements();
26       while(numHilos.hasMoreElements()) {
27         try {
28            ((AvisarClientes) numHilos.nextElement()).join();
29         } catch (InterruptedException e) {
30           ...
31         }
```

Figura 5.24 Notificación del objeto remoto con la pieza de información

Aprovechando la característica bloqueante de la invocación a métodos remotos del cliente (código mostrado en la Figura 5.25, línea 47), la clase *Billar* espera a que todos los hilos que lanzo terminen si ejecución (línea 28 del código mostrado en la Figura 5.24) para continuar con el turno siguiente. La finalización del método remoto significará que el cliente ha terminado su procesamiento.

```
37
38 class AvisarClientes extends Thread {
39   InterfaceApplet refCte;
40   ...
41     public AvisarClientes (InterfaceApplet refApp, int bAct,int f,float rotTaco) {
42       refCte = refApp;
43       ...
44     }
45     public void run() {
46        try {
47               refCte.ComenzarMovimientoCte(bolaActual,Fuerza,RotTaco);
48        } catch(RemoteException e) {
49                ...
50        }
51     }
```

Figura 5.25 Clase AvisarClientes (invocación del método remoto del cliente)

En el código de la Figura 5.26, se recibe la notificación en el cliente para que comience el procesamiento (método ComenzarMovimientoCte), se pone el objeto tipo *MiMonitor* a verdadero (línea 12) para que la clase ProxyScene actualice la escena.

```
import java.applet.*;
import java.rmi.*;
import vrml.external.Browser;
...

public class CaromApplet extends Applet implements InterfaceApplet, Serializable {
      ...

       public void ComenzarMovimientoCte(int bAct,int fuerza,float rotTaco) throws RemoteException {
         ...
         //RepEscena.start();
           monitor.estado = true;
           try {
                  synchronized(monitor) {
                              monitor.notify();
                  }
            } catch(Exception e) { ... }

           billarLocal.ComenzarMovimiento(bAct,fuerza,rotTaco);
           monitor.estado = false;
         //poner el taco en la bola que sigue...
         RepEscena.TACOset_posicion(f);
       }
```

Figura 5.26 Método remoto del cliente para iniciar procesamiento local

El procesamiento local para calcular el movimiento de las bolas se muestra en el bloque de código de la Figura 5.27. Es exactamente igual a como se realizaba del lado del servidor.

```
...
public class BillarLocal {

  private Bola ba,br,bb;  //Bola Amarilla, Roja, Blanca
      ...
       void ComenzarMovimiento(int intbolaActual, int Fuerza, float Angulo) {
            Bola bolaActual  =  new Bola();
            switch(intbolaActual){
              case Billar.intBOLA_BLANCA :
                    bolaActual=bb; break;
              case Billar.intBOLA_AMARILLA :
                    bolaActual=ba; break;
              case Billar.intBOLA_ROJA :
                    bolaActual=br; break;
            }
            bolaActual.setVel(new Vector2D(Fuerza,Angulo));
            procesarEventos();
      }
      void procesarEventos() {
            do{
                intBB=bb.Calcula(Mesa.DT, Mesa.FF);
                checaColisiones(bb);
                intBA=ba.Calcula(Mesa.DT, Mesa.FF);
                checaColisiones(ba);
                intBR=br.Calcula(Mesa.DT, Mesa.FF);
                checaColisiones(br);
                ...
              } while(intBB==0 || intBA==0 || intBR==0);
     }
      void checaColisiones(Bola bAct) {
      ...
      }
      ...
}
```

Figura 5.27 Procesamiento local (en el cliente)

La Figura 5.28 muestra como se lleva a cabo la actualización de la escena con la clase *ProxyScene*. Dicha actualización se activa con la clase *MiMonitor* (mecionada anteriormente) iterando sobre el objeto *BillarLocal* para obtener el estado de los elementos de la escena. (Líneas 10 a 19)

```
1  import vrml.external.*;
2
3  public class ProxyScene extends Thread{
4    private BillarLocal billar;
5    Browser browser = null;
6
7    ...
8
9     public void  run(){
10         while( true) {
11                     try {
12                       synchronized(monitor) {
13                           if(!(monitor.estado)) monitor.wait();
14                       }
15                       BBset_posicion.setValue(billar.getPosBlanca());
16                       BAset_posicion.setValue(billar.getPosAmarilla());
17                       BRset_posicion.setValue(billar.getPosRoja());
18
19                       Thread.sleep(SLEEP_TIME);
20                     } catch(Exception e) {
21                       ...
22                   }
23         }
24   }
25 }
```

Figura 5.28 Actualización de la escena con escrutinio sobre la clase local

5.4 Observaciones

Se llevaron a cabo todas las implementaciones propuestas en el Capitulo 3, pero el análisis se realizó trabajando sobre el modelo más representativo y optimizado de cada esquema. En el siguiente Capitulo se presentan los resultados de las pruebas de desempeño realizadas, las conclusiones y recomendaciones de utilización de cada modelo de acuerdo a las necesidades de los elementos que poblarán el Mundo Virtual Distribuido.

5.5 Resultados en pruebas de desempeño

Tomando los esquemas implementados con el caso de estudio, la Figura 6.1 muestra el resultado de las pruebas aplicadas, las cuales miden la carga de la red para cada esquema.

La gráfica muestra la utilización de la red para diferentes momentos de la ejecución:

- **Idle**: Nada esta sucediendo en el Mundo Virtual.
- **Actualización**: Cambios ocurren continuamente en el Mundo Virtual.
- **Cambio Remoto**: Existen cambios producidos por un usuario remoto.
- **Cambio Local**: Existen cambios producidos por un usuario local.

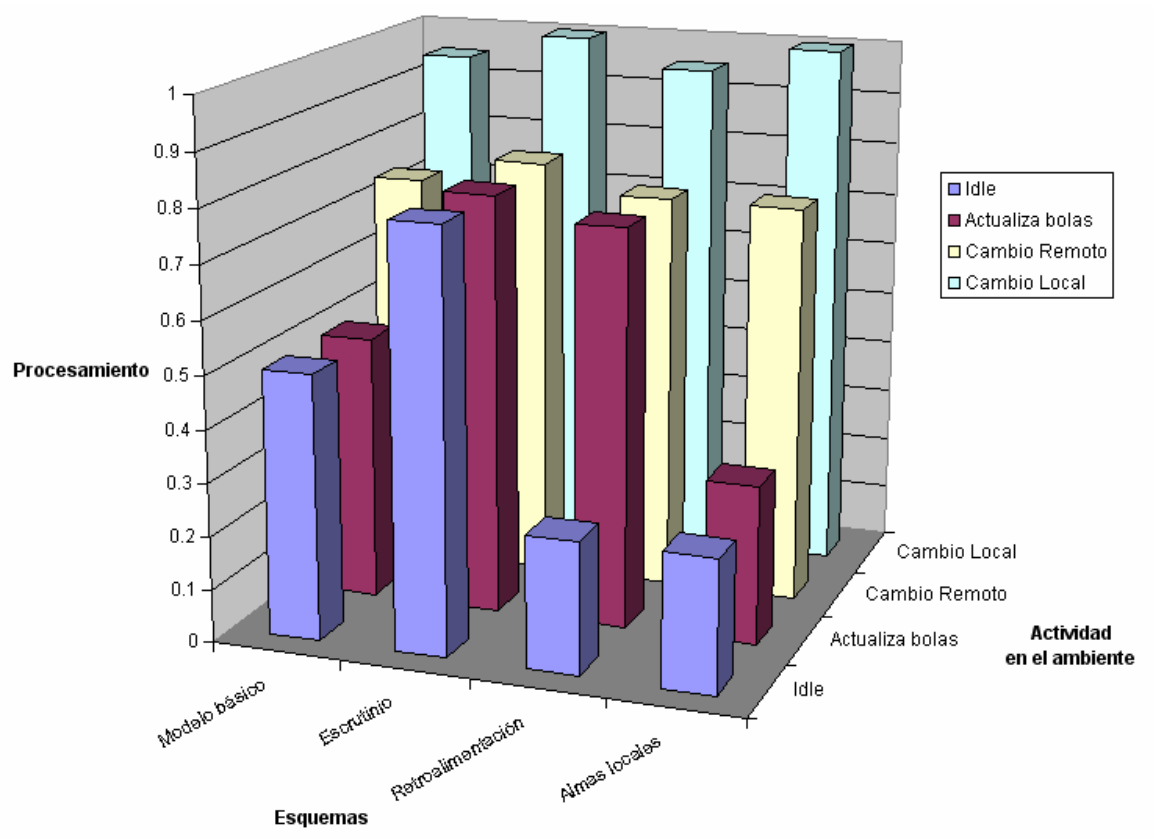


Figura 5.29 Utilización de la red con los diferentes esquemas (incluyendo el esquema básico)

El procesamiento está definido en una escala de 0 a 1, dónde 1 es el nivel mayor de actividad del procesador. Como se puede observar en la Figura 6.1, en el modelo básico y en el escrutinio (polling) optimizado existe una constante carga de trabajo sobre la red; la razón es que no importa que este sucediendo en el Mundo Virtual, el estado de los sujetos que lo pueblan son continuamente actualizadas.

En el segundo esquema, se incrementa la carga de trabajo sobre la red, pero solo cuando existen actualizaciones del estado de los sujetos.

Por otro lado, los esquemas 1 y 2 tienen una consistencia más fuerte respecto al tercer esquema. Hablando estrictamente, el esquema 3 tiene el mejor desempeño pero tiene la restricción que la naturaleza de la aplicación debe permitir que algo del trabajo de procesamiento pueda ser realizado en los clientes, lo que podría ocasionar variaciones en la consistencia en cada instancia del Mundo Virtual (en cada cuerpo).

En el último esquema (almas locales) se puede apreciar una reducción considerable de la carga de trabajo sobre la red; esto es por que la parte principal del procesamiento es realizada en cada una de las instancias del Mundo Virtual, es decir, por los clientes (cuerpos). Este esquema presenta inconsistencias debido a las características tecnológicas de la computadora de cada cliente, ya que el procesamiento se lleva a cabo más rápido o más lento en cada uno.

De acuerdo a los resultados obtenidos en las pruebas de desempeño, en la Tabla 5.1 se presentan algunas características adicionales usadas para evaluar los esquemas presentados:

Parámetro / Esquema	1 Polling	2 Callbacks	3 Almas Locales
Consistencia	Alta	Alta	Media
Tráfico de red	Alto	Medio	Bajo.
Procesamiento en clientes	Medio	Bajo	Alto
Procesamiento en servidor	Bajo	Alto	Bajo

Tabla 5.1 Parámetros de desempeño de los esquemas de implementación del modelo Alma-Cuerpo

Se establecen las conclusiones generales y se exponen los posibles trabajos a futuro sobre esta línea de investigación.

Capítulo 6: Conclusiones y trabajos a futuro

6.1 Conclusiones

En este trabajo se presenta el análisis y la implentación de algunas variantes de la arquitectura de desarrollo de MVDs llamada modelo Alma-Cuerpo. Se utilizaron recursos estándar y gratuitos de programación como lo son Java RMI y VRML, integrandolos en una aplicación de prueba.

1) Se cumplieron los objetivos planteados, al implementar la arquitectura alma-cuerpo con Java RMI y VRML integrándola en una aplicación de prueba.

   Se desarrollaron las extensiones al modelo básico de Alma-Cuerpo, modificando la forma en que los Mundos Virtuales son actualizados, es así como surgieron los 3 grandes esquemas presentados en este trabajo.

   Las pruebas fueron realizadas de acuerdo a la arquitectura propuesta en el capitulo 4, modificando la aplicación de prueba apegándose a cada modelo.

2) Se obtuvieron los parámetros de medición de los modelos propuestos, basándose en el compromiso que existe entre el uso del procesador, la cantidad de tráfico en la red y el grado de consistencia en los escenarios participantes.

3) Se procesó la información de cada modelo para obtener un esquema comparativo con los resultados obtenidos, el cual puede ser de utilidad para diseñar la utilización de los modelos dependiendo de los requerimientos de los objetos que conforman el ambiente y la infraestructura con que se cuenta.

4) En el caso de retroalimentación se documentaron importantes consideraciones de seguridad debido a que cada cliente es actualizado utilizando una interfaz expuesta del mismo, caso que es necesario tomar en cuenta en el caso de implementar este mecanismo.

5) Uso de esquemas ante diferentes situaciones de colaboración. Se analizaron principalmente los tres esquemas de comunicación entre los elementos del modelo Alma-Cuerpo. Cada uno tiene sus ventajas y desventajas, afortunadamente el uso de uno u otro no es exclusivo; se pueden usar esquemas mezclados dentro de una aplicación dependiendo de un análisis previo de cómo se va a comportar cada uno de los elementos que lo componen.

La Tabla 6.1 presenta algunos casos de comportamiento de elementos y el esquema recomendado:

Caso/Esquema	1 Polling	2 Callback	3 Almas locales
Objetos fijos o estáticos	Si	No	No
Estado estable	No	Si	No
Estado estable, cambios por acciones del usuario	No	Si	No
Estado bastante estable, cambios predictivos	No	Si	Si
Estado bastante estable, cambios no predictivos	No	Si	No
Estado inestable, cambios predictivos	Si	No	Si
Estado inestable, cambios no predictivos	Si	No	No

Tabla 6.1 Casos en los que se recomienda la utilización de los diferentes esquemas.

En esta tesis se presentaron los modelos con base en abstracciones computacionales como hilos, patrones de comunicación, callbacks, monitores, mecanismos de sincronización y concurrencia, con el fin de que estos modelos puedan ser implementados con casi cualquier tecnología. Es decir, independientemente del lenguaje que se usa para comunicación y para visualización. Sin embargo, cada día siguen surgiendo nuevas tecnologías que hacen más fácil la comunicación entre individuos y computadoras, por lo que se procede a enumerar una lista de posibles trabajos para facilitar la construcción de Mundos Virtuales Distribuidos.

6.2 Trabajos a Futuro

Dado que los modelos propuestos se presentan con bases conceptuales bien definidas, las arquitecturas no están fijas a las tecnologías aquí usadas, más bien están abiertas al uso de otras tecnologías como CORBA, .NET, J2EE, etc.

Aún existen muchas características por explorar en la combinación de tecnologías de mundos en 3D y objetos distribuidos, entre ellas:

- Utilizar activación de objetos remotos sobre demanda (Véase Figura 6.1)

La implementación de la capa de sesión de Java/RMI soporta activación de objetos por demanda a través de las interfaces de activación, las cuales evitan la necesidad de que los objetos servidor de Java/RMI estén en memoria todo el tiempo y los hace capaces de ser activados bajo demanda.

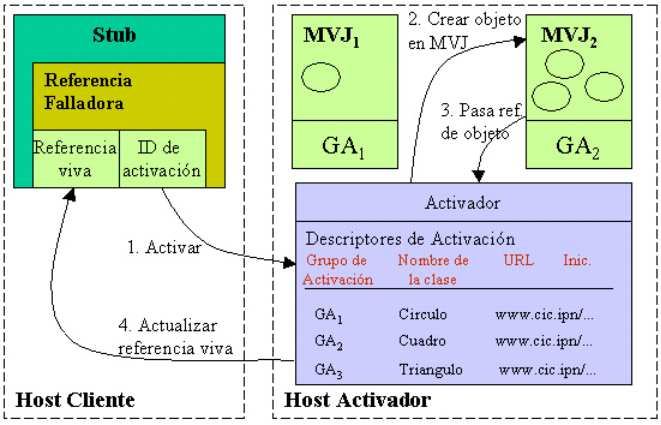


Figura 6.1: Esquema de activación de objetos distribuidos

- Extensión de algún lenguaje con un soporte tecnológico respaldado por lo obtenido es este trabajo de tesis:

Se propone crear un lenguaje descriptivo (posiblemente XML) que permita en base a la definición de una metodología desarrollar Mundos Virtuales Distribuidos de manera eficiente y con la facilidad de cambiar los esquemas de actualización con etiquetas propias de dicho lenguaje.

Este lenguaje deberá estar soportado por herramientas de implementación que permitan crear los programas necesarios independientes a la tecnología tanto de gráficos en 3D como de Objetos Distribuidos.

También las herramientas deberán ser capaces de parametrizar los diferentes modelos presentados en cada esquema del presente trabajo, permitiendo definir valores de hilos iniciales, esquemas dinámicos para realizar callbacks, sincronización de secciones críticas, etc.

Integración con plataformas orientadas a Servicios Web (SOA), definiendo los estados de los sujetos como mensajes XML, que permitan extraer información de diferentes fuentes, e inclusive encontrar las definiciones de las habilidades como definiciones de servicios.

- Integración e implementación con otras tecnologías

Crear nuevas implementaciones de referencia usando estos modelos, pero sobre nuevas tecnologías, como dispositivos móviles (celulares, asistentes personales digitales). Integración con tecnologías empresariales para manejo de contenido enriquecido como soporte a las decisiones de colaboración: repositorios de documentos en diferentes formatos, repositorios de medios digitales como fotografías y videos.

- Definir esquemas de actualización para Mundos Virtuales Distribuidos a gran escala

Por la magnitud de los grandes Mundos Virtuales Distribuidos, sería pesado para los equipos clientes y servidores mantener el estado de todos los elementos de un Mundo Virtual Distribuido.

Deberá ser posible definir e implementar esquemas de actualización de los elementos más cercanos “físicamente” al usuario del portal, para reducir de manera considerable el número de actualizaciones sobre los clientes y sobre los servidores.

Bibliografía

[1] Menchaca R. & Quintero R., “Distributed Virtual Worlds for Collaborative Work based on Java RMI and VRML”, Proceedings of the IEEE 6th International Workshop on Groupware CRIWG 2000.

[2] Sutherland, I. E. (1965), "The Ultimate Display", In Proceedings of IFIP Congress, (pp.506-508), Arlington, UA: Federation of Information Proceedings Societies.

[3] Singhal, Sandeep; & Zyda, Michael; (1999), “Networked Virtual Environments: Design and Implementation”, Addison Wesley

[4] Diehl, Stephan; “Distributed Virtual Worlds: foundations and implementation techniques using VRML, Java, and CORBA”; Springer, 2001.

[5] Broll, Wolfgang and England , David.: “Bringing Worlds Together: Adding Multi-user support to VRML”. 1995.

[6] Broll, Wolfgang. “Distributed Virtual Reallity for Everyone, a framework for Networked VR on the Internet”. 1996.

[7] Broll, Wolfgang. “Populating the Internet: Supporting Múltiple Users and Shared Application with VRML”. 1997.

[8] Locke, John. “An Introduction to the Internet Networking Enviroment and SIMNET/DIS”.
http://www.web3d.org/WorkingGroups/dis-java-vrml/DISIntro.ps

[9] Galli, R. and Lou, Y. “Mu3D: A causal consistency Protocol for a Collaborative VRML Editor”. Proceedings of 4th Simposium on the virtual Reallity Modeling Languaje VRML99. ACM SIGGRAPH, 1999.

[10] Virtual Reality Transfer Protocol
http://www.web3d.org/WorkingGroups/vrtp/

[11] Park, Sungwoo and Han, Taison. “Object-Oriented VRML For Mutli-user Enviroments”. 1997.

[12] Diehl, Stephan. “Distributed Virtual Worlds: Foundations and Implementation Techniques Using VRML, Java, and CORBA”. Springer

[13] VSPLUS: A high-level multi-user extension library for interactive VRML worlds
Yoshiaki Araki , Yoshiaki Araki
Proceedings of the third symposium on Virtual reality modeling language February 1998.

[14] Carlson, John A. and Clark, Adrian F.: "Multicast Shared Virtual Worlds Using VRML97". Proceedings VRML 99 of the fourth symposium on The virtual reality modeling language.

[15] Barber, K. S., McKay, R., MacMahon, M., Martin, C.E., Lam, D. N., Goel, A., Han, D. C. , Kim, D. "Sensible Agents: An Implemented Multi-Agent System and Testbed".

[16] Emmerich, Wolfgang. "Engineering distributed objects".2000. Wiley&Sons

[17] Second Life Project. http://www.secondlife.com

[18] Locke, John. "An Introduction to the Internet Networking Environment and SIMNET/DIS". http://www.web3d.org/WorkingGroups/dis-java-vrml/DISIntro.ps

[19] Web3D Consortium. http://www.web3d.org

[20] Java Technology. http://java.sun.com

[21] Alur, Deepak; Crupi, John; Malks, Dan; "Core J2EE Patters: Best Practices and Design Strategies"; Second Edition, Prentice Hall, 2003.

[22] Carreto Arellano, Chadwick: "Modelo y Arquitectura para la Implementación de Mundos Virtuales Distribuidos Colaborativos". Tesis de Maestria. Centro de Investigación en Computación – IPN, México (Concluida - 16/2/2004)

Apéndice A
VRML97

Estructura de VRML97

El intérprete de VRML recorre el ***grafo de la escena*** y reúne la información contenida en los nodos a lo largo de cada ruta (transformaciones, colores y datos geométricos). El *grafo de la escena* no es sólo un árbol, más bien es un ***grafo acíclico dirigido***, por lo que un nodo puede ser evaluado en contextos diferentes.

En el *grafo de la escena* que se muestra en la Figura A-1, los rectángulos con bordes redondeados representan nodos, las cadenas junto a las flechas son campos y los valores de los campos están en rectángulos (aunque pueden ser otros nodos).

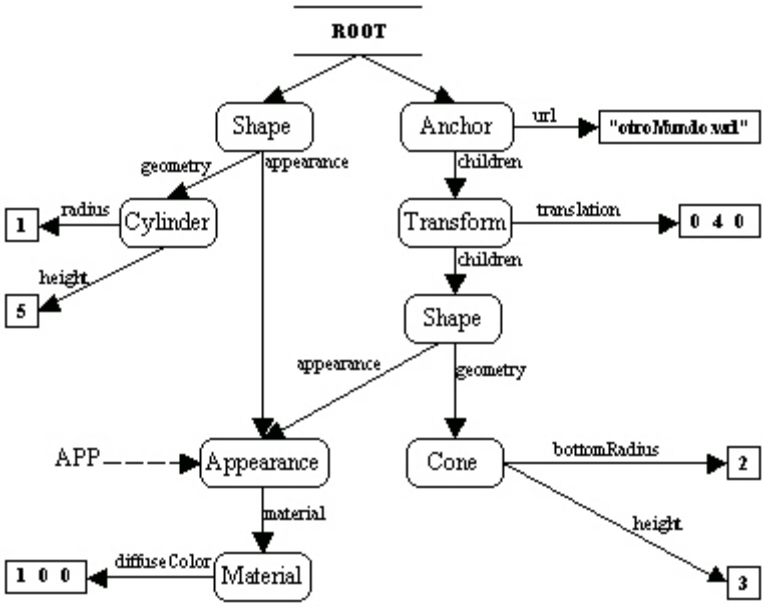


Figura A-1.- Ejemplo de un grafo de escena

Algunas herramientas para la edición y tratamiento de código VRML muestran el grafo de la escena como un árbol, tal es el caso del software de *ParallelGraphics* llamado *VrmlPad* [1] en su *versión 1.2* cuyo árbol se muestra en la Figura 2.

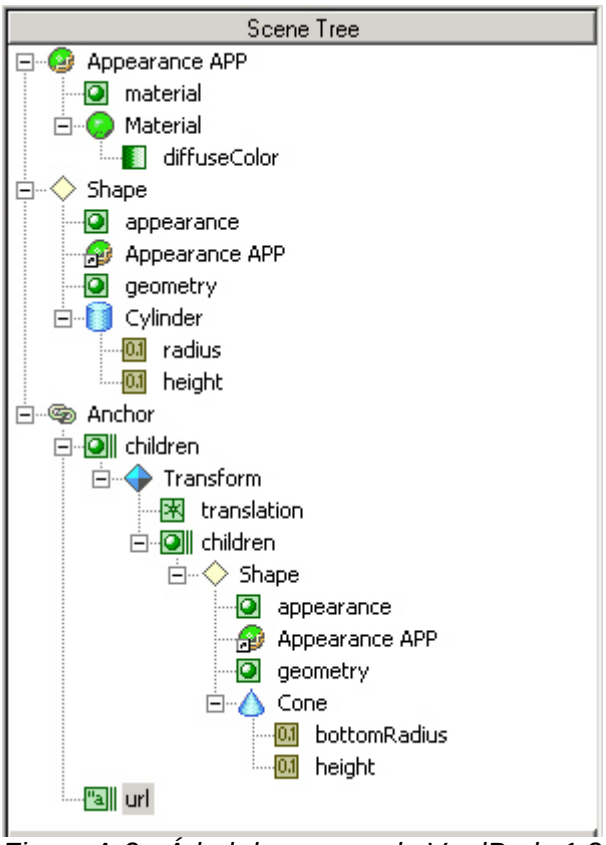


Figura A-2.- Árbol de escena de VrmlPad v1.2

Los ***nodos*** son los constructores de VRML ya que ellos especifican el grafo de la escena. Son definidos por sus *tipos de nodo* y *valores* para sus campos (los cuales pueden ser otros nodos). Los campos solo aceptan valores de cierto tipo (mostrados en la Tabla A-1).

Sintaxis de los nodos:

Nodo { $campo_1$ $valor_1$... $campo_2$ $valor_2$ }

Tipo de valor	**Descripción**	**Ejemplo**
SFBool	Valor de verdad	TRUE o FALSE
SFColor, MFColor	3 números de punto flotante entre 0 y 1. Los cuales especifican la contribución de colores Rojo, Verde y Azul (Red, Green, Blue –RGB-) respectivamente.	1.0 0.5 0.3
SFFloat, MFFloat	Número de punto flotante	3.1416
SFImage	Imagen especificada como una secuencia de escala de grises o valores de colores. Formato: *m n d* p_n... p_{n*m} donde: *m* es el ancho, *n* es el alto y *d* es el número de bytes por valor de color.	3 3 1 0x00 0xFF 0x00 0xFF 0xFF 0xFF 0x00 0xFF 0x00

SFInt32, MFInt32	Número entero de 32 bits	3014 ó 0xFF03FFA0
SFNode, MFNode	Un nodo VRML	Box { } o USE BALL
SFRotation, MFRotation	Cuatro números de punto flotante: Los primeros tres especifican el eje de rotación y el último número especifica el ángulo de rotación en radianes.	0 1 0 3.1416
SFString, MFString	Cadena de caracteres	"CIC-IPN"
SFTime, MFTime	Tiempo, como un número de segundos transcurridos desde 1.1.1970 00:00:00 GMT	
SFVec2f, MFVec2f	Vector con dos números de punto flotante	3.0 4.0 ó 3.0, 4.0
SFVec3f, MFVec3f	Vector con tres números de punto flotante	2.1 4.5 53.45 ó 2.1, 4.5, 53.45
Notas: *Las medidas en VRML se especifican en metros y los ángulos en radianes* *El prefijo* **SF** *es para tipo de valor Simple* *El prefijo* **MF** *es para tipo de valor Múltiple*		

Tabla A-1.- Tipos de valores de VRML

El código fuente que corresponde al grafo de escena presentado en la Figura A-1, es el que se muestra a continuación:

```
#VRML V2.0 utf8
DEF APP
      Appearance {
            material Material { diffuseColor 1 0 0     }
      }
      Shape {
            appearance USE APP
            geometry Cylinder { radius 1 height 5 }
      }
      Anchor {
            children
             Transform {
                translation 0 4 0
                children
                Shape {
                      appearance USE APP
                      geometry Cone {
                            bottomRadius 2
                            height 3
                      }
                }
             }
             url "otroMundo.wrl"
      }
```

Campo

Valores

Nodo

Como se muestra en el código anterior, los archivos de escenas VRML comienzan con la línea: #VRML V2.0 utf8 para indicar la versión de VRML y la codificación de caracteres, la cual en VRML 1.0 era ASCII. (Excluyendo la primera línea, las demás que empiecen con el carácter # se consideran como comentarios).

Comportamiento en VRML97

A la modificación del estado[5] de los objetos del mundo virtual como respuesta a alguna acción del usuario, tiempo, o evento se le denomina *comportamiento*. El comportamiento puede ser desde cambiar de color algún objeto mediante un *click* del ratón hasta el cálculo de la trayectoria de un misil dirigido.

Existen tres mecanismos para brindarle comportamiento a los mundos VRML:

Con *eventos, enrutamientos, nodos sensores e interpoladores* (VRML puro)
Con el nodo *Script* (programa en Java o JavaScript)
Con un applet de Java vía la *Interfaz de Autoría Externa* (EAI) de VRML

1. Eventos, enrutamientos, nodos sensores e interpoladores

Los hipervínculos no son el único modo de interactuar con el usuario, existen **sensores** para capturar y generar cierto tipo de **eventos** externos. Los *sensores* generan *eventos de salida* llamados *eventOut*, los cuales pueden ser enviados a otros nodos, de manera que los eventos de salida de un nodo son conectados a los *eventos de entrada*, *eventIn*, de otros nodos. A esta conexión de comunicación entre eventos se le denomina **enrutamientos** en VMRL, es decir, una ***ruta*** es la conexión entre un nodo que genera un evento y un nodo que recibe el evento.

Un ejemplo de un sensor es un TimeSensor (sensor de tiempo), el cual puede enviar eventos periódicamente a un interpolador. Un interpolador define una función lineal la cual esta dada por valores llave y los valores de entrada que son interpolados linealmente. El interpolador recibe un evento con valor *e* desde el sensor de tiempo, calcula el valor de la función *f(e)* y envía *f(e)* a otro nodo. De esta manera un nodo interpolador puede determinar la posición de un objeto con respecto al tiempo. Los eventos están relacionados directamente con el *tipo de campos*:

Tipo de campos	Indica que:
field	El valor del campo es especificado cuando el nodo es instanciado y no se puede cambiar después.
eventIn	El valor del campo es recibido como un evento, vía rutas, desde otros nodos
eventOut	El valor del campo puede ser enviado, vía rutas, a otros nodos
exposeField	Es ambos, un evento de entrada y un evento de salida. Existe una convención de nombres para indicar el tipo de evento, por ejemplo, para un campo tipo exposedField con el nombre translation, el evento de entrada (eventIn) asociado tiene el nombre set_translation, y el evento de salida asociado (eventOut) tiene el nombre translation_changed.

Tabla A-2.- Tipo de campos

[5] Modificar el estado de un objeto puede ser el cambio de posición, tamaño, apariencia, orientación, etc.

Los **enrutamientos** se declaran en el archivo de VRML de la siguiente manera:

ROUTE $DefName_1$.EventOutName **TO** $DefName_2$.EventInName

donde los DefName son nodos nombrados con la sentencia DEF y los EventOutName y EventInName, son nombres de eventos de esos nodos.

En la Figura A-3 se puede observar cómo es el enrutamiento de los eventos a través del grafo de la escena.

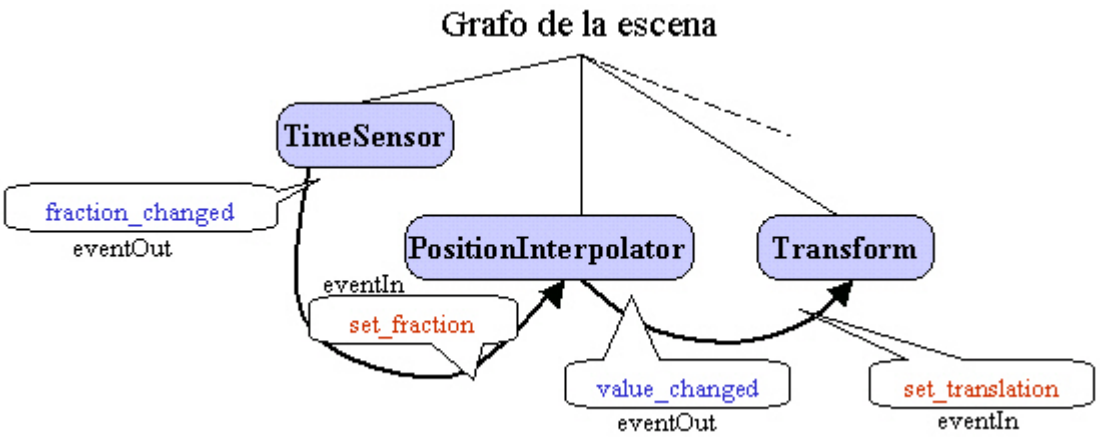


Figura A-3.- Enrutamiento de eventos entre nodos

A continuación se muestra el código correspondiente con los enrutamientos del grafo de la escena de la figura 3:

```
#VRML V2.0 utf8
Viewpoint { position 0 3.3 10 }

DEF     BALL Transform {
        children Shape {
                geometry Sphere     { radius 0.5 }
                appearance Appearance  {
                        material Material { diffuseColor 1 0 0       }
                }
        }
}
DEF     CLOCK TimeSensor {
        loop TRUE
        cycleInterval 3
}
DEF     POSITION PositionInterpolator {
        key     [       0  , 0.166,  0.333,  0.5 , 0.666, 0.833,   1  ]
        keyValue [0 0 0, 0 4 0, 0 6 0, 0 7 0, 0 6 0, 0 4 0, 0 0 0]
}
ROUTE CLOCK.fraction_changed TO       POSITION.set_fraction
ROUTE POSITION.value_changed TO       BALL.translation
```

En el código anterior se define un nodo tipo **TimeSensor** (reloj), el cual empieza a generar eventos a partir de que se carga la escena. Su evento de salida (eventOut) fraction_changed es de tipo SFFloat y su valores está dentro del intervalo [0..1] incrementándose en los eventos subsecuentes (si el campo loop del TimeSensor esta en TRUE entonces cuando se alcanza el final del intervalo el reloj comienza sobre el inicio del intervalo). El evento generado por el reloj, fraction_changed, es enrutado hacia el evento llamado event_fraction del nodo **PositionInterpolator**, el cual genera un evento de salida llamado value_changed de tipo SFVec3f, que es una posición en un espacio en tres dimensiones, ese evento es enviado (otro enrutamiento) al evento set_translation del nodo **Transform** para que el objeto se mueva a otra posición.

Otros nodos sensores e interpoladores para VRML se describen a continuación:

CilynderSensor	Los movimientos del ratón son convertidos a rotaciones alrededor del eje Y local.
PlaneSensor	Los movimientos del ratón son convertidos a translaciones en la dirección de los ejes X y Y locales.
ProximitySensor	Si el usuario entra, se va o se mueve dentro de área (espacio delimitado) entonces este sensor genera eventos.
SphereSensor	Los movimientos del ratón son convertidos a rotaciones alrededor del origen del sistema de coordenadas local.
VisibilitySensor	Si un objeto entra o deja el campo de visión del usuario, entonces este nodo genera eventos.
ColorInterpolator	Este nodo define un función lineal piecewise sobre triples o listas de triples números de punto flotante(MFVec3f). Para PositionInterpolator los valores son solamente triples y no listas.
OrientationInterpolator	Este nodo define una función lineal piecewise sobre valores de rotación (SFRotation).
ScalarInterpolator	Este nodo define una función lineal piecewise sobre números de punto flotante (SFFloat).

Tabla A-3.- Sensores e Interpoladores

El tipo de animaciones los cuales pueden ser implementados usando nodos sensores e interpoladores son limitadas, por lo que si las animaciones o interacciones dependen por ejemplo de otras acciones del usuario que no se pueden predecir y que requieren de cálculos sofisticados y complejos será necesario auxiliarse de lenguajes de programación externos a VRML tales como JavaScript, Java, o los que la especificación de VRML [2] tenga soportados.

2. Nodo *Script*

Este nodo proporciona mayor flexibilidad para lograr comportamientos complejos, con el inconveniente que se requiere de programación. Fue creado para encapsular funcionalidad de ciertos objetos del mundo con comportamiento propio.

El nodo *Script* (el cual es escrito en un lenguaje de programación que soporte el visualizador de VRML, como Java), es un conjunto de funciones procedurales ejecutadas normalmente como parte de una *cascada de eventos*[6] (se asume que todos los eventos en una cascada de eventos han ocurrido simultáneamente). Una función

[6] Cascada de eventos: Secuencia de eventos iniciados por un script o un evento sensor y propagado de nodo a nodo a lo largo de una o más rutas.

Script puede también ser ejecutada de modo asíncrono, es decir, es ejecutado por el visualizador de VRML cuando se ha cargado el grafo de la escena completamente.

El nodo Script puede:

Recibir eventos de otros nodos, procesarlos y enviarlos a otros nodos,
Mantiene la pista de la información entre ejecuciones subsecuentes,
Retiene el estado interno sobre el tiempo.

La sintaxis de creación de un nodo *Script* es diferente que la de otros nodos, ya que se definen nuevos tipos de eventos que serán necesarios para la comunicación del programa encapsulado con los nodos del grafo de la escena. También se definen campos que se usarán para almacenar valores entre las llamadas al código del programa encapsulado (como las variables globales en lenguajes de programación).

Sintaxis del nodo Script:

```
Script { Declaraciones de campo o evento
       directOutput      valor de verdad
       mustEvaluate      valor de verdad
       url               código del programa o su URL
      }
```

El procesamiento de eventos es desempeñado por un programa referenciado por el campo *url* del nodo *Script*.

Si el programa encapsulado cambia el grafo de la escena a través del nodo Script y también pone los valores de los eventos de otros nodos directamente en el código del programa, entonces el valor del campo directOutput es verdadero (TRUE).

Si el visualizador puede retardar el procesamiento de eventos de salida del nodo Script hasta que sus valores sean requeridos por los eventos de entrada que se reciben, entonces el valor del campo mustEvaluate debe ser falso (FALSE).

Ejecución del *Script*

El nodo *Script* se activa cuando recibe un evento, de manera que el visualizador ejecuta el programa especificado en el *url.* El nodo *Script* podría ser también ejecutado después de ser creado.

Los eventos que recibe el nodo *Script* se dan en estampas de tiempo que corresponden al evento que las generó.

El *Script* que es especificado en el URL implementa tres métodos importantes:

El método initialize()
El método shutdown()
El método processEvent()

El método initialize() será invocado antes de que el visualizador de VRML presente el mundo al usuario y antes de que cualquier evento sea procesado por cualquier nodo en el mismo archivo VRML como el nodo *Script* que contenga este código.

Los eventos generados por el método initialize() tendrán estampas de tiempo menores que cualquier otro evento que genere el nodo *Script*. Esto con la finalidad de permitirle al nodo Script optimizar tareas de inicialización prioritarias para que el usuario interactúe con el mundo.

De la misma manera, el método shutdown() es invocado cuando el nodo *Script* es borrado o cuando el mundo que contiene el nodo *Script* es "descargado" o reemplazado por otro mundo. Es un método que se usa como una operación de limpieza como avisar que se borren archivos temporales. Ningún método puede ser invocado después de la ejecución del método shutdown(). El borrado del nodo *Script* que contiene el método shutdown() no se completa hasta que se termina la operación de su método shutdown().

El método eventsProcessed() es llamado después de que uno o más eventos son recibidos. Este método permite a los *Scripts* que no siguen el orden de los eventos recibidos, generar menos eventos que un *Script* equivalente que genera eventos cada vez que éstos son recibidos. Si es utilizado en alguna otra forma no dependiente del tiempo, entonces el método *eventsProcessed(),* podría ser no determinístico, ya que diferentes implementaciones de visualizadores podrían llamarlo en diferentes momentos.

Para una simple cascada de eventos, el método eventsProcessed() de un nodo *Script* debe ser llamado *a lo más una vez* (esa es la semántica de la invocación). Los eventos que son generados desde el método eventsProcessed() son dados en la estampa de tiempo del último evento procesado.

El Script tiene la funcionalidad de:

Acceder a campos y eventos de salida del *Script*
Acceder a eventos de entrada y eventos de salida de otros nodos
Enviar eventos de Salida

La especificación de VRML97 describe el API de programación para el lenguaje de programación java, donde se describen las funciones y el procedimiento para construir mundos virtuales con comportamiento interno propio usando el nodo *Script*.

3. EAI

La interfaz de Autoria Externa (EAI) es una interfaz de programación para comunicación entre VRML y programas externos, como los applets de Java.

La EAI proporciona dos servicios importantes que se ilustran en la figura 4:
Le permite a los programas externos leer y cambiar el grafo de la escena
Le permite a los programas externos registrar algunas de sus funciones como *callbacks*.

Cada *callback* está ligada a un evento, siempre que este evento sea generado en la escena VRML, el visualizador invoca la función *callback* asociada y pasa el valor actual del evento como un argumento.

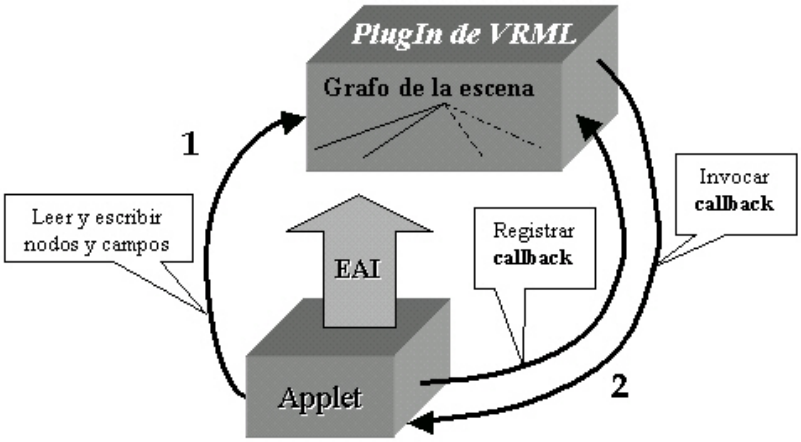


Figura A-4.- EAI: comunicación entre el applet y el PlugIn de VRML

Para que un applet de Java pueda usar los servicios que la EAI proporciona, es necesario usar la etiqueta DEF en los archivos de VRML.

Ejemplo:

```
#VRML V2.0 utf8
DEF ROOT Transform { }
```

De esta manera el applet podrá obtener la referencia de la raíz del grafo de la escena en el visualizador VRML. El paquete vrml.external contiene las clases y subpaquetes de la EAI, las cuales se deben importar en adición al paquete de java.applet.

En el método init() del applet se le solicita la referencia al PlugIn de VRML con la llamada al método getBrowser() de la clase Browser. Entonces se invoca al método getNode() de esta referencia para poder acceder a algún nodo marcado con la sentencia DEF.

Problemas con la EAI

La EAI fue desarrollada como un compromiso y así proporcionar solo funcionalidad limitada. Una regla de diseño fue que no debería permitir más accesos al grafo de la escena que los que fueron proporcionados antes con el nodo *Script* y los eventos.

Esto fue considerado necesario de manera que los vendedores de *plugins* de VMRL no tuvieran que sacrificar optimización del grafo de la escena por lo que les fue fácil añadir la EAI a los visualizadores existentes.

Un defecto esencial de la EAI es que no proporciona métodos para obtener el tipo de un nodo o una lista de sus campos. Por esta razón es imposible implementar un esqueleto genérico del grafo de la escena, por ejemplo, para buscar todas las ocurrencias de un cierto tipo de nodo en el grafo.

Eventos en la EAI

Manipular eventos hacia y desde los nodos:

Después de obtener la instancia de **Node**, puedes obtener las referencias a los campos **EventIn** y **EventOut** del nodo. Entonces se pueden usar los métodos setValue() y getValue() de esos EventIn y eventOut para obtener información del mundo VRML y enviar de vuelta las actualizaciones a los nodos del mundo VMRL.

```
// instancia EventIn
EventInSFVec3f setnewtranslation;

// Valores aleatorios para poner
float[] valuetoset={0, 0, 0};

// Aquí obtenemos EventIn de Node
try{
        setnewtranslation=(EventInSFVec3f)
        somenode.getEventIn("set_translation");
} catch (InvalidEventInException e){
        System.out.println("Error: Invalido EventIn");
}

// Aquí ponemos el valor del EventIn hacia Node
setnewtranslation.setValue(valuetoset);
```

Recibir eventos desde el mundo VRML en el applet:

El **EventOutObserver** es una interfaz pública para obtener eventos desde el mundo VRML. Implementar **EventOutObserver** en el objeto java que manejará los eventos recibidos desde la escena.

public ObservadorDeEventos extends Object implements EventOutObserver {

En tu definición del objeto para el **EventOutObserver** se necesita redefinir el método **callback()** de la interfaz **EventOutObserver**:

```
public void callback( EventOut event,
                      double timestamp,
                             Object someobject ) {
//repartir valores y usarlos para hacer alguna cosa
int four=4;
if (four ==((Integer)someobject).intValue()){
        function();
}
}
```

En el applet, hacer una instancia del EventOutObserver antes de usarlo:

```
ObservadorDeEventos theeventoutobserver;
```

Después obtener el nodo, obtener una instancia del evento de salida (eventOut):

```
EventOutSFVec3f theeventout;

try{
theeventout=(EventOutSFVec3f)somenode.getEventOut("translation_changed");
} catch (InvalidEventOutException e) {
        System.out.println("Invalid EventOut Exception");
}
```

Después de obtener una referencia a el **EventOut,** se necesita notificar el evento de salida del dato hacia el **EventOutObserver** de esta manera se recibirán los eventos desde el mundo VMRL.

```
theeventout.advise(theeventoutobserver, null);
```

Null en este ejemplo representa un objeto que es pasado al callback del registrado EventoOutObserver. Se puede reemplazar **null** con otros objetos pero se necesita convertiros a valores útiles (cast) dentro del método **callback()** redefinido de tu EventOutObserver.

A veces al estar observando los eventos se puede colisionar el browser. Advertencia: en la mayoria de las versiones de Cosmo Player, usando un argumento de cualquier otra cosa que **null** llevará hacia una falta y caída de memoria. De esta manera si se tiene que observar un conjunto de nodos, se recomienda usar un **Observer** por cada uno de ellos.

El applet externo puede implementar **EventOutObserver**. En este caso se usa una declaración del método como esta:

```
eventout.advise(this, new Integer(4));
```

Además se tiene que redefinir el método **callback()** en el applet, para corresponder con la firma del método abstracto de la interfaz **EventOutObserver**.

```
public void callback(EventOut event, double timestamp, Object someobject) {
        Integer x=(Integer)someobject;
}
```

Si las callback se congelan o se bloquean, no se intenta enviar métodos o nodos en el hilo del callback, pero desde otro hilo.

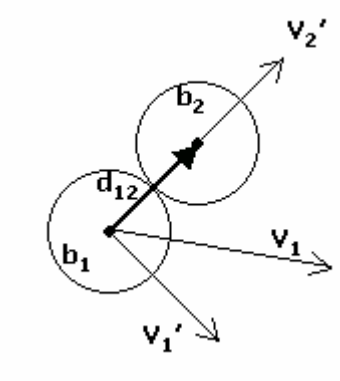
REFERENCIAS

[1] VRMLPAD. http://www.parallelgraphics.com/products/vrmlpad/

[2] VRML97 Especificación. Homepage http://www.web3d.org/

Apéndice B
Cálculo de las colisiones de las bolas de billar

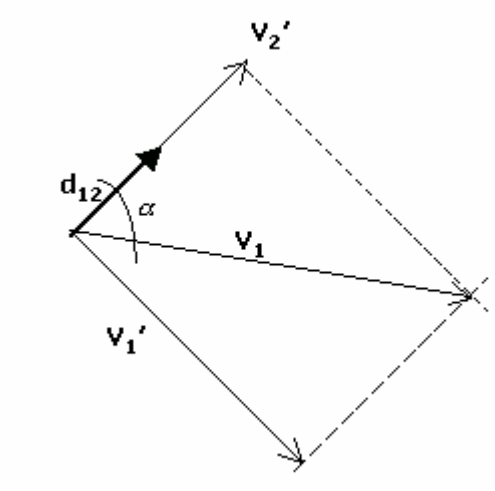
Suponiendo la bola $b_1$en movimiento y la bola $b_2$en reposo:



Donde:
V_1= dirección original de b_1
V_1'= dirección resultante de b_1 después de la colisión
V_2'= dirección resultante de b_2 después de la colisión.
d_{12}= es la distancia entre los centros de las esferas b_1 y b_2.

Por lo tanto, necesitamos calcular los valores de V_1', y V_2'.

Al momento de la colisión se tiene el siguiente sistema de vectores resultante:



De donde α = ángulo entre los vectores V_2' y V_1.

De donde se tiene:

$$\cos\alpha = \frac{d_{12}.v_2}{|d_{12}||v_1|}$$

Proyectando la magnitud de $\boldsymbol{V_1}$ en la dirección de $\boldsymbol{d_{12}}$:

El $\cos\alpha = \dfrac{|V_2'|}{V_1}$ por lo que

$$|V_2'| = |V_1|\cos\alpha$$

Por otro lado, dado que conocemos $|V_2|$ y su dirección, se tiene:

$$V_2' = |V_2'|\left(\frac{d_{12}}{|d_{12}|}\right)$$

Sustituyendo $|V_2'|$ en la ecuación anterior:

$$V_2' = \left(|V_1|\cos\alpha\right)\left(\frac{d_{12}}{|d_{12}|}\right)$$

de donde se sustituye cos α, por lo que queda:

$$V_2' = \left(\frac{d_{12}V_1}{|d_{12}|}\right)\left(\frac{d_{12}}{|d_{12}|}\right)$$

Tomando d_{12} como un vector unitario, es decir $|d_{12}| = 1$, nos queda:

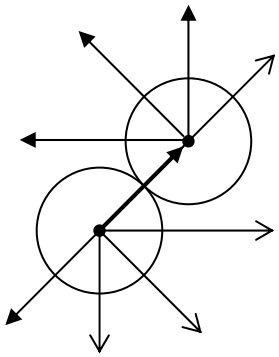
$$V_2' = \left(d_{12} \bullet V_1\right)d_{12}$$

Conociendo V_2' se tiene que:

$$V_1' = V_1 - V_2'$$

Consideraciones:

- Si $\cos\alpha < 0$ no hay afectación ya que las dos bolas van en la misma dirección.
- Si las bolas están ambas en movimiento se hacen los cálculos para cada una de ellas y las velocidades resultantes se suman.



Apéndice C
Firma de Java Applets para Netscape

Cuando se requiere que un Applet acceda a recursos restringidos de una máquina tales como el sistema de archivos, propiedades del sistema y conexiones de red, es necesario firmarlo digitalmente para lo que se requiere de un certificado digital el cual debe distribuirse a los clientes que usaran el Applet.

Consideraciones previas

Antes de firmar un Applet digitalmente es deben tomar en cuenta las siguientes consideraciones:

1. Contar con un certificado digital, el cual es proporcionado por autoridades certificadoras [1][2]. Para este trabajo se utilizó un certificado de prueba (gratuito) usando la herramienta *SignTool* [3].

2. Disponer de una herramienta de firmado de applets. En el caso de usar Netscape Navigator se utilizó *SignTool versión 1.1*[3] la cual está disponible de manera gratuita.

3. Darle privilegios al Applet: es necesario incluir en el código fuente del Applet las invocaicones a las funciones necesarias para garantizar los privilegios requeridos. Esas llamadas se encuentran en las clases del API de Capacidades Java de Netscape (*Netscape Java Capabilities API* [4]).

   Privilegios para el applet:

   Los applets tienen que pedir permiso de usar recursos de una manera correcta, esto se consigue mediante el API de Capacidades. Cada vez que un applet quiere efectuar alguna acción restringida debe pedir permiso al navegador para ver si éste se lo concede, para lo cual se sirve de las funciones de dicho API. Cuando el applet le pide permiso al navegador, éste busca en su base de datos si existe un registro para el firmante del applet y si la acción le está permitida, en caso de no existir ningún registro, el navegador presenta una ventana pidiendo confirmación antes de conceder el permiso, el usuario decide si garantiza o niega el permiso con la posibilidad de almacenar la decisión en la base de datos para futuras referencias.

   La forma de modificar el applet consiste en preceder todas las llamadas a recursos protegidos, con una llamada al administrador de privilegios, dependiendo del tipo de recursos que el applet accede ya que existen distintas llamadas dependiendo del tipo de recurso:

Recursos del sistema	Llamada que precederá al acceso
Propiedades del sistema	PrivilegedManager.enablePrivilege("UniversalPropertyRead");
Para leer, crear y escribir archivos	PrivilegedManager.enablePrivilege("UniversalFileAccess");
Callbacks en RMI	Netscape.security.PrivilegedManager.enablePrivilege("UniversaConnect");

Tabla C-1.- Llamadas para conceder privilegios al applet

Una vez terminadas las operaciones que requieren privilegios, se recomienda revocar los privilegios concedidos, para ello se utiliza la siguiente función:

```
PrivilegeManager.revertPrivilege("UniversalPropertyRead");
```

Existe información en Internet para consultar la lista completa de recursos que se pueden acceder[5].

El gran inconveniente de este enfoque es que exige la modificación del código fuente del applet, teniendo como consecuencia de los cambios introducidos, que no funcionará ya en ningún otro navegador, echando por tierra la filosofía de Java de "Escribir una vez, ejecutar en cualquier sitio".

Obtención del certificado digital

Para utilizar un certificado de prueba (generado a partir de la herramienta SignTool), se requiere del seguimiento de un proceso que va desde preparar el Netscape para la instalación del certificado hasta la creación y distribución del mismo.

Base de datos de certificados de Netscape:

Para instalar el certificado en la base de datos de Netscape Communicator, es necesario crear una contraseña para acceder a la base de datos. Esto se logra pulsando el botón de *Seguridad* en la barra de herramientas de Netscape:

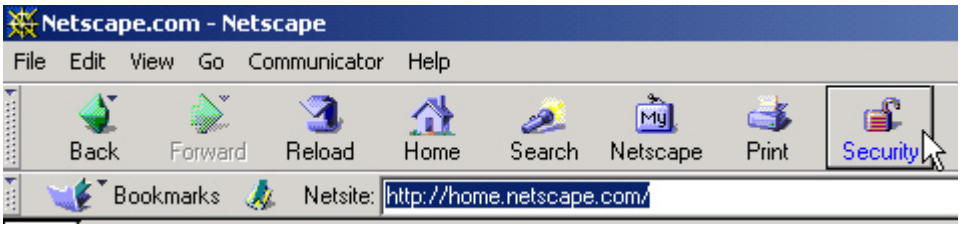


Figura C-1 Botón de seguridad en Netscape

Se elige *Passwords* y después se pulsa el botón de "*Establecer Password"* para crearlo*:*

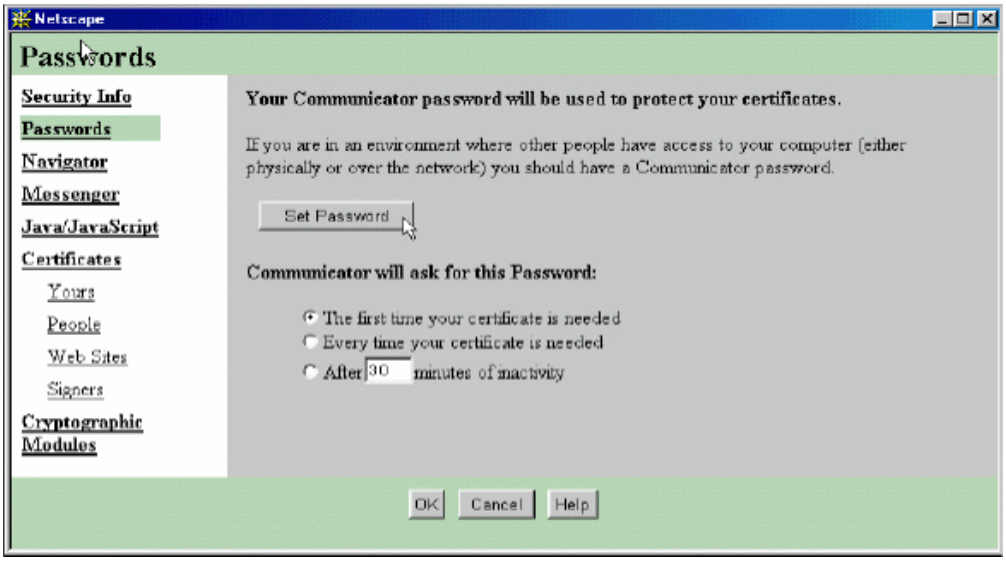


Figura C-2 Almacen de contraseñas en Netscape

Luego se introduce la contraseña para las bases de datos de claves y certificados de Netscape Communicator:

Figura C-3 Establecimiento de contraseña en Netscape

Generación del certificado

Se debe ejecutar SignTool para generar el certificado de prueba. Es necesario cerrar todas las ventanas abiertas del *Netscape Communicator* para evitar que se corrompan las bases de datos de claves y certificados.

Para generar un certificado de prueba se utilizan las opciones:

-G seguida de un nombre que será el nombre del certificado.
-d para indicar el directorio donde se encuentra la base de datos de certificados (el archivo **cert7.db**) y claves (el archivo **key3.db**) de Netscape Communicator.

Por ejemplo, para *crear un certificado*[7] para Manuel Estrada del Centro de Investigación en Computación del IPN:

C:\>**signtool -G mestrada -d c:\netscape\users\mestrada**
using certificate directory: **c:\netscape\users\mestrada**
Enter certificate information. All fields are optional. Acceptable characters are numbers, letters, spaces, and apostrophes.
certificate common name: **Certificado para pruebas**
organization: **IPN**
organization unit: **CIC**
state or province: **DF**
country (must be exactly 2 characters): **MX**
username: **mestrada**
email address: **mestrada@sagitario.cic.ipn.mx**
Enter Password or Pin for "Communicator Certificate DB": *(sin eco)*
generated public/private key pair
certificate request generated
certificate has been signed
certificate "mestrada" added to database
Exported certificate to x509.raw and x509.cacert
Al terminar este paso la base de datos de claves y certificados del Netscape Communicator especificado en la opción –d queda actualizada automáticamente.

Existen dos maneras para comprobar la instalación del certificado:

En el Netscape Communicator, se pulsa el botón de Security de la barra de herramientas, luego en Certificates y a continuación en Yours y deberá aparecer el certificado recién creado:

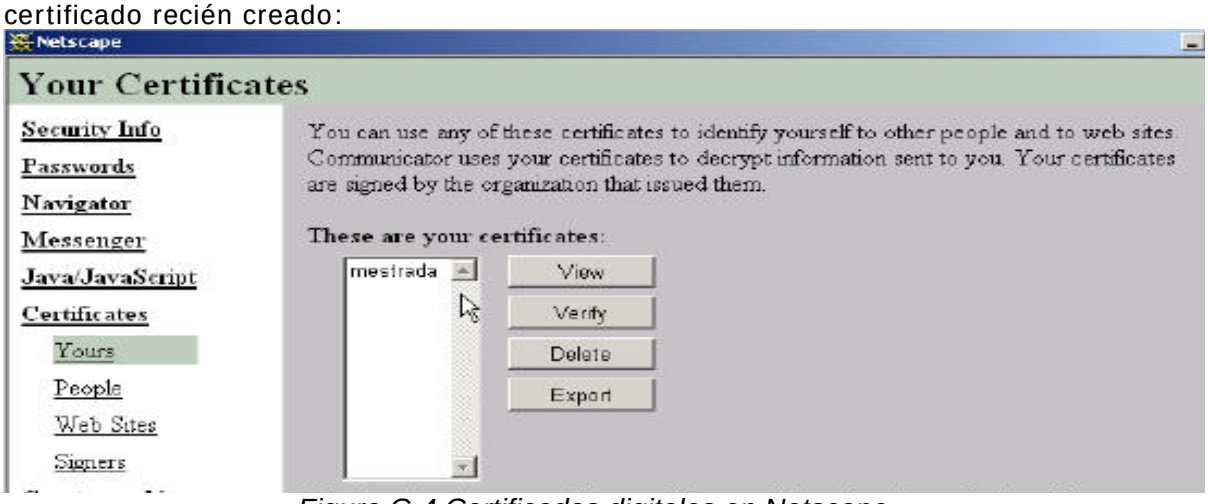


Figura C-4 Certificados digitales en Netscape

o también con la herramienta SignTool, con la opción -l:

C:\>signtool **-l -d C:\Netscape\Users\mestrada**
using certificate directory: C:\Archivos de programa\Netscape\Users\mestrada

[7] Con letra en negritas aparece lo que se debe de introducir en la línea de comandos.

```
-------------------------------------
mestrada
Issued by: mestrada (Certificado para pruebas)
Expires: Wed Sep 05, 2007
-------------------------------------
For a list including CA's, use "signtool -L"
```

Con este certificado se está listo para firmar los applets. Sin embargo, no basta con firmarlos, ya que para que los usuarios puedan ejecutar los applets firmados es necesario que instalen en su versión de Netscape Communicator el certificado que se acaba de generar.

Distribución e instalación del certificado

SignTool generó dos ficheros automáticamente: **x509.raw**, y **x509.cacert**

El archivo x509.cacert contiene el certificado en formato base64, y es el que se tiene que proporcionar a los usuarios que deseen ejecutar el applet. Para distribuir el certificado, se siguen los siguientes pasos:

a. Crear una liga al archivo x509.cacert en una página HTML:
   <h2> Instalar mi certificado. **</h2>**

b. Asegurarse que el servidor Web exporta el archivo como un tipo MIME:
   El tipo MIME debe ser **application/x-x509-ca-cert.** Los servidores de Netscape están configurados así por defecto y también algunos servidores Unix. A los demás servidores Web como el IIS o Apache habrá que configurarlos para que asocien este tipo MIME a la extensión *.cacert.

c. Instalación del certificado:
   Los usuarios necesitan darle click a la liga para que automáticamente se ejecute el proceso de instalación de certificados de Netscape Communicator. El usuario podrá a partir de ahora ejecutar correctamente todos los applets firmados con ese certificado.

Firmado del applet

Los pasos básicos para firmar el applet son:

1. Se crea un directorio al que se copian todas las clases que componen el applet,
2. Se firma el directorio utilizando la herramienta SignTool. La propia herramienta se encarga de generar un archivo comprimido Java Archive (*.jar) con todas las clases y demás archivos que se hayan copiado en el directorio.

Supongamos que se quiere firmar la clase myClass.class, se deberá copiar a un directorio llamado myClass y ejecutar SignTool pasándole como argumento la clave del certificado almacenada en la base de datos del Netscape Communicator.

Las opciones usadas como argumentos son:

-d Para especificar la ruta de la base de datos de certificados de usuarios
-k Nombre del certificado
-Z Nombre que tendrá el archivo JAR, junto con el nombre del directorio donde se encuentran las clases.

Por ejemplo:

```
C:\>signtool -d C:\Netscape\Users\mestrada -k mestrada -Z myClass.jar myClass
using certificate directory: .
Generating myClass/META-INF/manifest.mf file..
--> myClass.class
adding myClass/gon.txt to myClass.jar...(deflated 0%)
Generating zigbert.sf file..
```

Aquí pide la contraseña:

```
Enter Password or Pin for "Communicator Certificate DB": (sin eco)
```

Si se ha introducido correctamente, continúa:

```
adding myClass/META-INF/manifest.mf to myClass.jar...(deflated 14%)
adding myClass/META-INF/zigbert.sf to myClass.jar...(deflated 27%)
adding myClass/META-INF/zigbert.rsa to myClass.jar...(deflated 43%)
tree " myClass" signed successfully
```

Ahora ya está disponible el archivo myClass.jar para distribuirlo con su firma incorporada.

Para verificar que el proceso se ha realizado correctamente se utiliza la opción -v:

```
C:\> signtool -v myClass.jar
using certificate directory: .
archive " myClass.jar" has passed crypto verification.
status path
------------ -------------------
verified myClass.class
```

Añadir el archivo JAR en la página web

La etiqueta <applet> cambia para permitir la inserción del archivo JAR:

```
<applet
code= myClass.class
archive= myClass.jar
width=...
height=...>
</applet>
```

Nota: Es importante eliminar el archivo myClass.class original del directorio, ya que en caso contrario el navegador lo cargará en lugar del firmado.

En cada ocasión en la que el applet necesite acceder a un recurso protegido, al usuario se le presentará una ventana de advertencia, en la que se le informa de los permisos que está solicitando ese applet, acompañados de una estimación del riesgo potencial derivado de su concesión (esta ventana aparecerá cada vez que el applet intente

acceder de nuevo a ese recurso hasta que se active la casilla para recordar la decisión). Para mostrar los datos del certificado de prueba que se generó, se puede oprimir el botón "C*ertificate*" de la ventana de advertencia.

Cabe mencionar que este proceso fue el que se utilizó en el presente trabajo de tesis debido a que se utilizó Netscape para el desarrollo de la aplicación, sin embargo existen otras herramientas y mecanismos para firmar Java Applets [6].

Referencias

[1] VeriSign en URL: http://www.verisign.com

[2] Otras Autoridades Certificadoras en http://certs.netscape.com/

[3] Herramienta SignTool versión 1.1 http://www.signtool.com o en http://developer.netscape.com/software/signedobj/jarpack.html

[4] Netscape JAVA CAPABILITIES API (API de Capacidades de Java de Netscape) URL: http://developer.netscape.com/docs/manuals/signedobj/capsapi.html y en URL: http://developer.netscape.com/library/documentation/signedobj/capsapi_classes.zip

[5] Netscape System Targets URL: http://developer.netscape.com/docs/manuals/signedobj/targets/index.htm

[6] Álvarez Marañón, Gonzalo; Servicio Criptonomicón del Instituto de Física Aplicada del CSIC. http://www.iec.csic.es/criptonomicon/

Glosario

3D Tercera dimensión

API *Application Program Interface.* Es el conjunto de métodos que ofrece cierta librería para ser utilizados por otro software como una capa de abstracción

Avatar Palabra hindú que significa: “Dios en la tierra”
Se le conoce a la representación de un usuario dentro de la escena o mundo virtual.

AVD Ambiente Virtual Distribuido.

EAI API de interfaz de autoría externa para VRML.

HTTP *Hypertext Transfer Protocol.* Protocolo que es utilizado en cada transacción de la Web.

ISO La Organización Internacional para la Estandarización (ISO) es una organización internacional no gubernamental, compuesta por representantes de los organismos de normalización (ONs) nacionales, que produce normas internacionales industriales y comerciales. Dichas normas se conocen como normas ISO y su finalidad es la coordinación de las normas nacionales, en consonancia con el Acta Final de la Organización Mundial del Comercio, con el propósito de facilitar el comercio, facilitar el intercambio de información y contribuir a la transferencia de tecnologías.

JVM La máquina virtual de Java (en inglés Java Virtual Machine, JVM) es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

Middleware El Middleware es un software de conectividad que permite ofrecer un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red).

Monolítico Moduló único ensamblado de software que engloba todos los servicios del sistema.

MV Mundos Virtuales

Plugins Conector para extender la funcionalidad de una aplicación.

Proxy Código que representa alguna entidad dentro del sistema.

VRML97 VRML (acrónimo del inglés Virtual Reality Modeling Language. "Lenguaje para Modelado de Realidad Virtual") - formato de archivo que tiene como objetivo la representación de gráficos interactivos tridimensionales; diseñado particularmente para su empleo en la web.

Reenderizado o Desplegado Cuando la computadora despliega las descripciones que conforman una escena en 3d

RMI RMI (Java Remote Method Invocation) es un mecanismo ofrecido en Java para invocar un método remotamente. Al ser RMI parte estándar del entorno de ejecución Java usarlo provee un mecanismo simple en una aplicación distribuida que solamente necesita comunicar servidores codificados para Java.

Rmic Compilador de metodos remotos de la tecnología Java RMI

RV Realidad Virtual

Stub Representante del objeto remoto en el cliente que lo invoca.

X3D X3D es un lenguaje informático para gráficos vectoriales definido por una norma ISO, que puede emplear tanto una sintaxis similar a la de XML como una del tipo de VRML (Virtual Reality Modelling Language) (VRML). X3D amplía VRML con extensiones de diseño y la posibilidad de emplear XML para modelar escenas completas en tiempo real.