Universidad de las Ciencias Informáticas

Facultad 5

Módulo de Red para Sistemas de Realidad Virtual que utilizan Ogre como motor gráfico.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor(es):

Frank Del Río Hernández. Reyneris Terrero Delfino

Tutor(es):

Ing. Gadied Carrero Sotolongo Ing. Luis Angel Ravelo Hernández

Ciudad de la Habana, Diciembre 2010

"Año 52 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
Autor	Tutor.
_____	_____
Autor	Tutor.

“Pero la juventud tiene que crear. Una juventud que no crea es una anomalía realmente.”

Ernesto Che Guevara

AGRADECIMIENTOS

A Riolvis Acosta, por guiarme en el momento que mas lo necesitaba. A mis amigos Lien, Annier, Beatriz, por todo el tiempo que me dedicaron.
A todos los amigos que ya no están aquí, pero que siempre me apoyaron.

Reyneris.

Primero que quisiera agradecer a mis tutores por su apoyo, también a Riolvis, Annier, Lien y Beatriz, por su ayuda incondicional con mas lo necesitabamos. Y a todas la amigos que de una forma u otra hicieron posible que este trabajo saliera adelante.

Frank.

DEDICATORIA

A mis padres.

A mi hermana.

A la memoria de mi tía Nely.

A mi familia en general.

Reyneris.

A la memoria de mi madre que aunque no se encuentre físicamente conmigo se que esta presente.

A la memoria de mis abuelos que realmente una padres para mi.

A mi padre.

Mis hermanas.

A tio Gilberto.

A mis tias.

En fin a todas mi familia en general.

Frank.

TABLA DE CONTENIDO

INTRODUCCIÓN

La historia de redes en informática es compleja. Participaron en ella muchas personas de todo el mundo a lo largo de los últimos 35 años. A partir de la década de 1960 y durante las décadas de 1970, 1980 y 1990, el Departamento de Defensa de Estados Unidos (DoD) desarrolló Redes de Área Amplia (WAN) de gran extensión y alta confiabilidad, para uso militar y científico. Permitía la conexión de varios computadores mediante diferentes rutas. La red en sí determinaba la forma de transferir datos de un computador a otro. En lugar de poder comunicarse con un solo computador a la vez, se podía acceder a varios computadores mediante la misma conexión. La WAN del Departamento de Defensa finalmente se convirtió en la Internet.

Desde el año 2001, Ogre ha crecido hasta convertirse en uno de los motores gráficos más populares de código abierto. Ha ganado un respeto considerable desde su creación, por ser potente, flexible y fiable, y ha sido utilizado en un gran número de proyectos de producción, en áreas tan diversas como videojuegos, simuladores, programas informáticos educativos, arte interactivo, visualización científica, y otros.

Cuba, centrando su esfuerzo en el desarrollo de la industria del software, como una de las principales tareas de la Batalla de Ideas ha comenzado a dar sus primeros pasos en este campo. Con el objetivo de insertar a Cuba en el mercado del software a nivel mundial y para la informatización del país se crea en el año 2002 la Universidad de las Ciencias Informáticas (UCI), la cual desde sus inicios tiene como objetivo revolucionar la industria del software en el país, alcanzando hoy en día notables logros en el ámbito internacional.

La UCI, en aras de convertir la Informática en una de las ramas más productivas de la nación cubana y asumir con orgullo y placer el reto de informatización de la sociedad cubana, se ha convertido en un centro científico donde se vincula la docencia con el proceso productivo, en el desarrollo de varios perfiles informáticos.

Situación problémica

En los últimos años se ha incrementado en el área de Entornos Virtuales el desarrollo de sistemas de realidad virtual capaces de comunicarse entre sí de manera remota, este hecho ha aumentado la

atracción de los usuarios, pero al mismo tiempo, la complejidad de este tipo de software. Un ejemplo concreto lo constituye el desarrollo de videojuegos, donde es necesario que las máquinas envíen y reciban datos por la red, en dependencia del valor de los datos y según la lógica correspondiente al videojuego en cuestión, se ejecutarán acciones que modificarán el estado del mismo. En el área de Entornos Virtuales de la facultad 5 de la UCI no existe un módulo de red que permita la interacción entre varios usuarios en las aplicaciones desarrollados con la herramienta Ogre y que permita la comunicación entre nodos, independientemente de la localización de los mismos. Esta situación limita la producción en la facultad de uno de los software que más ganancias económicas aporta en el área de Entornos Virtuales: los videojuegos multijugador.

Problema científico

Inexistencia de funcionalidades para el envío y recepción de datos por redes en sistemas de Realidad Virtual que utilizan Ogre como motor gráfico.

Objeto de estudio

Transmisión de datos por red en sistemas computarizados.

Campo de acción

Transmisión de datos en sistemas de Realidad Virtual con arquitectura cliente/servidor que utilizan Ogre como motor gráfico.

Objetivo General

Desarrollar un módulo de red para sistemas de Realidad Virtual que utilizan Ogre como motor gráfico.

Idea a defender

Con la implementación de este módulo se podrá lograr la comunicación entre nodos de aplicaciones que utilizan Ogre como motor gráfico, así como la interacción entre varios usuarios de estas aplicaciones.

Tareas

- Identificar los elementos y fundamentos de la transmisión de datos por red y los distintos tipos de arquitectura, protocolos y APIs teniendo en cuenta las tendencias actuales.
- Analizar las distintas bibliotecas de red existentes y middleware y establecer una comparación de acuerdo a sus ventajas y desventajas.
- Diseñar un módulo de red con funcionalidades que permitan una adecuada comunicación entre las aplicaciones que utilizan a Ogre como motor gráfico.
- Implementar el módulo de red con las funcionalidades diseñadas.
- Probar las funcionalidades del módulo mediante la realización de una aplicación de demostración.

Métodos científicos empleados:

Teóricos:

- Analítico – sintético: Permite hacer un estudio de la teoría y documentación referente al desarrollo de software de conexión, con el fin de extraer los elementos más importantes y particulares que se relacionan con el tema.
- Inductivo – deductivo: Permite hacer un buen razonamiento de la investigación realizada acerca del software de conexión y llegar a una conclusión satisfactoria sobre la misma.
- Análisis histórico – lógico: Permite conocer con mayor profundidad cómo han evolucionado los software de conexión, conociendo así la trayectoria histórica real de su desarrollo.

Capítulo I

FUNDAMENTACIÓN TEÓRICA.

En el presente capítulo se tratan temas respecto al motor gráfico Ogre y las arquitecturas de comunicación por redes. Detallando la estructura que debe tener el funcionamiento de la red. Además se analizan las técnicas de transmisión, los protocolos y la tecnología empleada en el desarrollo de este trabajo.

1.1 Arquitectura de red

Las computadoras se comunican por medio de redes. La red más sencilla es una conexión directa entre dos computadoras. Sin embargo, también pueden conectarse a través de grandes redes que permiten a los usuarios intercambiar datos, comunicarse mediante correo electrónico y compartir recursos.

En general la arquitectura o topología de red no es más que la disposición física en la que se conectan los nodos de una red de ordenadores o servidores, mediante la combinación de estándares y protocolos.

1.2 Protocolo TCP/IP

TCP/IP son las siglas de Protocolo de Control de Transmisión/Protocolo de Internet (en inglés Transmission Control Protocol/Internet Protocol), un sistema de protocolos que hacen posibles servicios Telnet, FTP, E-mail, y otros, entre ordenadores que no pertenecen a la misma red.

El Protocolo de Control de Transmisión (TCP) permite a dos anfitriones establecer una conexión e intercambiar datos. El TCP garantiza la entrega de datos, es decir, que los datos no se pierdan durante la transmisión y además de que los paquetes sean entregados en el mismo orden en el cual fueron enviados.

La arquitectura de un sistema en TCP/IP tiene una serie de metas:

- La independencia de la tecnología usada en la conexión a bajo nivel y la arquitectura del ordenador.
- Conectividad universal a través de la red.
- Reconocimientos de extremo a extremo.
- Protocolos estandarizados.
- El modelo básico en internet es el modelo cliente/servidor. El cliente es un programa que solicita a otro que le preste un servicio. El servidor es el programa que proporciona dicho servicio.

La arquitectura de Internet está basada en capas. Esto hace más fácil implementar nuevos protocolos. El conjunto de protocolos TCP/IP, al estar integrado plenamente en Internet, también dispone de este tipo de arquitectura. El modelo de capas de TCP/IP es algo diferente al propuesto por ISO (International Standard Organization) para la interconexión de sistemas abiertos (OSI).[1]

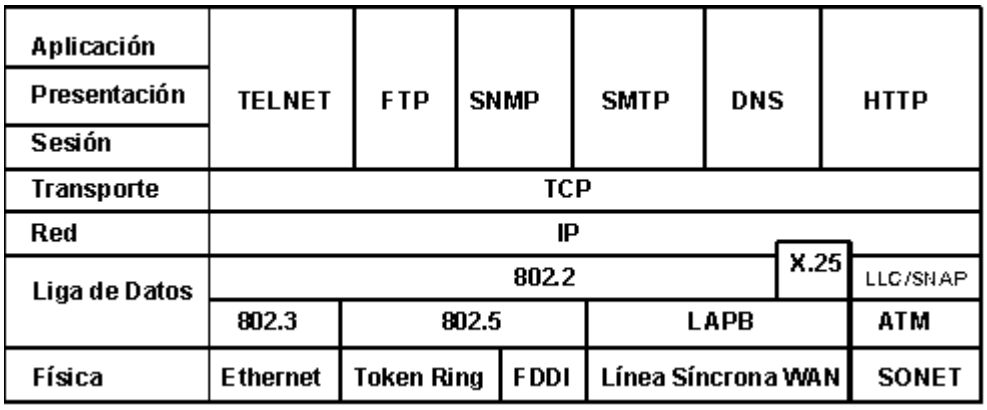


Figura 1 Comparación entre el modelo OSI y el protocolo TCP/IP.

1.2.1 Capa de Aplicación

Esta capa corresponde a las aplicaciones (de usuario o no), ofreciendo la posibilidad de acceder a los servicios de las demás capas y define los protocolos que utilizan las aplicaciones para intercambiar datos,

como correo electrónico (POP y SMTP), gestores de bases de datos y protocolos de transferencia de archivos (FTP).

La Capa de Aplicación es construida en un nivel superior a la Capa Lógica. Las aplicaciones interactivas de red y de tiempo real han sido objeto de estudio en el área del desarrollo de simuladores, entornos virtuales y videojuegos. Los factores fundamentales que influyen en el diseño de estos sistemas son: la escalabilidad, la persistencia y la colaboración entre jugadores. La escalabilidad es la capacidad de adaptar el poder de cómputo, en dependencia de la cantidad de recursos disponibles en todo momento. Por lo tanto la aplicación debe ser capaz de adaptarse dinámicamente y distribuir de forma adecuada el cómputo en cada nodo ante la variación en el número de jugadores. Para ello la implementación de la aplicación tiene que basarse en software concurrente, usando el paralelismo en hardware a través de la red de nodos. La escalabilidad cumple con el siguiente principio: Cada participante nuevo sobrecarga la comunicación pero al mismo tiempo, ofrece un mayor poder de cómputo a la aplicación. La persistencia es la manera en que un nodo coexiste con la aplicación, esto se refiere a la entrada y salida del nodo y en cómo esto afecta al estado de la aplicación. Por ejemplo cuando un nodo abandona la aplicación ésta debe actualizar el estado del videojuego redistribuyendo las responsabilidades de este nodo y por otra parte si el nodo se desconecta bruscamente, la aplicación pierde los objetos existentes en ese nodo. Por lo tanto la persistencia tiene que garantizar la administración de la configuración, así como también la detección y recuperación de errores.

La colaboración se refiere a la existencia de equipos en los cuales los miembros actúan unidos con el propósito de lograr un objetivo común (por ejemplo, eliminar a otro equipo). Para lograr la colaboración la aplicación debe proveer a un jugador información abundante y exacta acerca de los otros jugadores. Esta información puede ser conocimiento acerca del estado de cada jugador o pueden compartir un canal de comunicación dedicado. Técnicamente la colaboración requiere que la comunicación entre dos jugadores sea priorizada y está claro que el concepto de equipo es a nivel de aplicación, pues el concepto de colaboración a distancia puede ser complejo y requiere más recursos. En la capa de aplicación se interpretan los datos según el contexto y la lógica (por ejemplo, un valor entero representa una posición), por lo tanto esta capa está estrechamente relacionada con los usuarios finales y las aplicaciones específicas, ésta se deriva de las especificaciones del diseño de la aplicación ya sea un videojuego o cualquier otro sistema.[2]

1.2.2 Capa de Transporte

Esta capa provee comunicación extremo a extremo desde un programa de aplicación a otro. Puede proveer un transporte confiable asegurándose de que los datos lleguen sin errores y en la secuencia correcta. Coordina a múltiples aplicaciones que se encuentren interactuando con la red simultáneamente de tal manera que los datos que envíe una aplicación sean recibidos correctamente por la aplicación remota.[3]

En esta capa se encuentran los protocolos UDP y TCP. A continuación se muestra una pequeña comparación entre los dos protocolos.

	TCP	UDP
Conexión	Necesita establecer un canal de comunicación.	No necesita conexión previa para enviar paquetes.
Transferencia	Garantiza que los datos sean transferidos plenamente entre los dos agentes.	No garantiza que se reciban todos los datos.
Orden de los paquetes	Se reciben en el mismo orden que fueron enviados.	No garantiza que se reciban en el mismo orden que fueron enviados.
Tamaño de los paquetes	Los paquetes con los datos pueden tener un tamaño variable.	Todos los paquetes tienen la misma longitud.
Reenvío de los paquetes	De haber un problema en la transmisión los paquetes se reenvían.	No existe reenvío de paquetes.
Velocidad	Es un protocolo lento ya que tiene que garantizar que todos los paquetes se reciban y en orden.	Es un protocolo rápido, ya que no requiere comprobaciones adicionales.

Figura 2 Comparación entre los protocolos UDP y TCP.

1.2.2.1 UDP (User Datagram Protocol)

El protocolo UDP (User Datagram Protocol) proporciona aplicaciones con un tipo de servicio de datagramas orientado a transacciones. El servicio es muy parecido al protocolo IP en el sentido de que no es fiable y no está orientado a la conexión. El UDP es simple, eficiente e ideal para aplicaciones como el TFTP y el DNS. Una dirección IP sirve para dirigir el datagrama hacia una maquina en particular, y el número de puerto de destino en la cabecera UDP se utiliza para dirigir el datagrama UDP a un proceso específico localizado en la cabecera IP. La cabecera UDP también contiene un número de puerto origen que permite al proceso recibido conocer cómo responder al datagrama.[3]

1.2.2.2 TCP (Transmission Control Protocol)

El protocolo TCP proporciona un servicio de comunicación que forma un circuito, o sea, que el flujo de datos entre el origen y el destino parece que sea continuo. TCP proporciona un circuito virtual, el cual es llamado una conexión.

Al contrario que los programas que utilizan UDP, los que utilizan el TCP tienen un servicio de conexión entre los programas llamados y los que llaman, chequeo de errores, control de flujo y capacidad de interrupción.[3]

1.2.3 Capa de Red

Controla la comunicación entre un equipo y otro. Conforma los paquetes IP que serán enviados por la capa inferior. Desencapsula los paquetes recibidos pasando a la capa superior la información dirigida a una aplicación.

1.2.3.1 IP (Internet Protocol)

El Protocolo IP proporciona un sistema de distribución que es poco fiable, incluso en una base sólida. El protocolo IP especifica que la unidad básica de transferencia de datos en el TCP/IP es el datagrama.

Los datagramas pueden ser retrasados, perdidos, duplicados, enviados en una secuencia incorrecta o ser fragmentados intencionadamente para permitir que un nodo con un *buffer* limitado pueda capturar todo el datagrama. Es la responsabilidad del protocolo IP reensamblar los fragmentos del datagrama en el orden

correcto. En algunas situaciones de error los datagramas son descartados sin mostrar ningún mensaje mientras que en otras situaciones los mensajes de error son recibidos por la máquina origen.

El protocolo IP también define cuál será la ruta inicial por la que serán enviados los datos.

Cuando los datagramas viajan de unos equipos a otros, es posible que atraviesen diferentes tipos de redes. El tamaño máximo de estos paquetes de datos puede variar de una red a otra, dependiendo del medio físico que se emplee para su transmisión. A este tamaño máximo se le denomina MTU (*Maximum Transmission Unit*), y ninguna red puede transmitir un paquete de tamaño mayor a dicha MTU. El datagrama consiste en una cabecera y datos.[3]

1.2.4 Capa Física

Esta capa es la de más bajo nivel, pues está relacionada con el cableado y el hardware. En la misma están presentes las limitaciones de los recursos físicos, que influyen en el diseño de la aplicación. Además, una vez establecida la conexión entre los nodos se necesitan técnicas para la transmisión de información de un nodo a otro, así como los protocolos de comunicación que definen la forma de transmisión para que los nodos puedan comunicarse y la información sea entregada de forma correcta.[3]

1.2.4.1 Técnicas de Transmisión

Las técnicas para la transmisión de mensajes pueden ser divididas en tres tipos:

- Unicasting: La comunicación se establece entre dos nodos (uno emisor y el otro receptor); esta técnica permite controlar y direccionar el tráfico de punto a punto. Si el mismo mensaje es deseado por varios receptores, la transmisión unicasting derrocha ancho de banda enviando mensajes repetitivos.[4]

- Multicasting: La comunicación se establece entre un emisor y múltiples receptores subscriptos a grupos específicos. Cuando el emisor envía un mensaje, es recibido sólo por los receptores interesados (los que pertenezcan al grupo al cual se envía el mensaje), sin necesidad de duplicarlo, por lo que ésta es una técnica eficiente para transmitir información a un gran número de nodos.[4]

- Broadcasting: La comunicación se establece entre un emisor y todos los restantes nodos de la red, por lo que cada nodo tiene que recibir y procesar cada mensaje emitido. Esto provoca que en redes con grandes números de nodos no se garantice el broadcasting (como es el caso de las redes WAN).[4]

1.3 Arquitectura Cliente/Servidor

El modelo cliente/servidor es la arquitectura más usada actualmente en el desarrollo de videojuegos. En esta arquitectura hay un nodo especial (el servidor) al cual se conectan los demás nodos (los clientes), es por ello que en general la arquitectura sea centralizada. Son varias las razones por las cuales la arquitectura cliente/servidor centralizada es tan popular. En primer lugar, es simple de implementar en relación con las demás arquitecturas, pues sólo se tiene en cuenta un tipo de conexión: la existente entre el cliente y el servidor. Por otra parte el servidor centralizado garantiza una alta consistencia y además los proveedores de videojuegos (para el caso de los videojuegos por Internet) pueden crear y controlar una estructura de subscripción para utilizarla con objetivos de mercado. Hay que destacar que también existen algunos problemas al usar esta arquitectura. El modelo implica una alta dependencia con el servidor, por lo que si este nodo falla, lo hará también toda la aplicación. El requerimiento de ancho de banda para el servidor siempre supera significativamente al del cliente. Y por último, el envío de mensajes hacia los clientes a través del servidor provoca una latencia adicional a la comunicación.

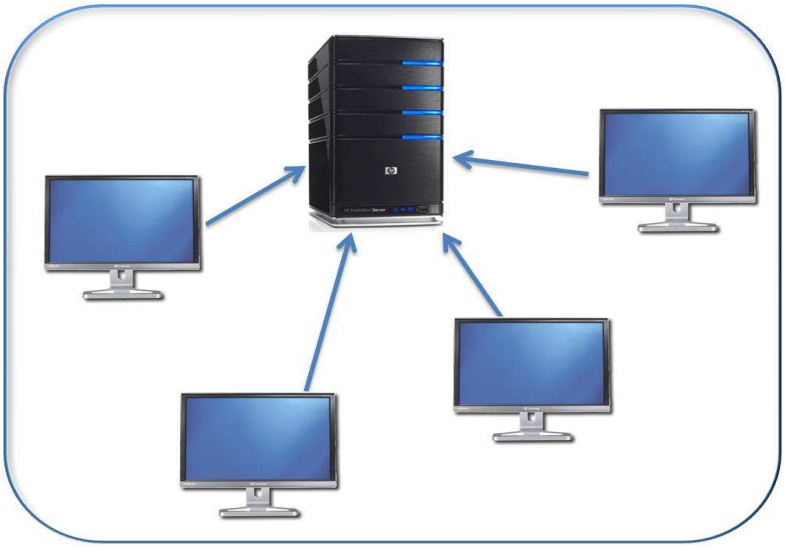


Figura 3 Arquitectura Cliente-Servidor.

1.3.1 El Servidor

El servidor es una sesión que acepta nuevas conexiones de clientes. Este programa es responsable de recibir todas las entradas de los clientes, calcular todo el estado de la aplicación y mantener a los clientes actualizados. Para optimizar, el servidor puede filtrar sólo información concerniente a cada cliente en particular, por ejemplo, enviarle al usuario sólo la información de las entidades que están en su campo de visión y de manera similar con los sonidos, actualizar sólo los que el usuario puede escuchar. Para ello existen técnicas de filtrado, con las cuales se puede lograr disminuir los requerimientos de ancho de banda a un orden inclusive inferior al lineal, dependiendo de la aplicación específica que se está diseñando.

1.3.2 El Cliente

El cliente es el programa con el cual el usuario interactúa, ejecuta los gráficos y los sonidos además de manipular la entrada de los dispositivos como teclado, mouse y joystick. Por lo tanto el cliente es responsable de enviar la entrada del usuario al servidor y de esperar por la respuesta de éste último para actualizar la vista de la aplicación. Para lograr un mayor realismo y una mejor sensibilidad, algunas de las simulaciones pueden realizarse en el cliente, sobre todo para el caso en que éstas no influyan en el

estado del videojuego para el resto de los usuarios. Por ejemplo, si un jugador se voltea, esta acción puede ser ejecutada al instante, sin necesidad de esperar por la respuesta del servidor. También se pueden realizar en el cliente detección de colisiones.

1.4 Bibliotecas de red

Luego de identificar los elementos y fundamentos de la transmisión de datos por red y los distintos tipos de arquitectura y protocolos, se procede a analizar las distintas bibliotecas de red existentes y middleware.

1.4.1 Biblioteca Raknet

Raknet está programada en C++, constituye un motor para la gestión y trabajo con redes, principalmente en videojuegos. Permite trabajar con sistemas cliente/servidor y punto a punto. Trabaja con mensajes UDP. Tiene soporte para trabajar en la transmisión de voz con la librería de sonido FMOD. Está diseñada para tener un alto desempeño, fácil de integrar y constituye una solución completa para videojuegos y otras aplicaciones. Posee un sistema de replicación de objetos que automáticamente crea, destruye, serializa y trasmite los objetos de videojuego. Tiene una robusta capa de comunicación con un control de flujo automático, ordenamiento de mensajes por múltiples canales, reagrupamiento del mensaje, fraccionamiento y el posterior reensamblaje del mensaje. Llamadas remotas a procedimientos propios de C y C++, con una lista automática de parámetros serializados. A pesar de ser una biblioteca robusta y tener todas estas comodidades para los programadores no constituye una solución para este trabajo debido a que es software propietario y en el caso de crear una aplicación con carácter comercial, sería necesario comprar una licencia para el uso de esta librería.[5]

1.4.2 Biblioteca HawkNL

También se encuentra HawkNL que es una API de red orientada a videojuegos, es libre, de código abierto, liberada bajo la licencia GNU LGPL (GNU Lesser General Public License), se considera una API de bajo nivel ya que trabaja bastante cercano a los sockets, tiene funciones sobre Berkeley/Unix Sockets y WinSock. Permite conexiones TCP y UDP. HawkNL también brinda otras ventajas incluyendo soporte para distintos sistemas operativos, grupos de sockets, un temporizador de alta exactitud, estadísticas de los sockets, macros para leer y enviar datos en paquetes, soporte para múltiples transportes de red. Ha sido probada en Windows 9X/ME/NT/2000/XP/CE, Linux, Mac OS y otros sistemas operativos. Esta biblioteca

trabaja básicamente con los sockets brindando funciones para la conexión, la lectura y escritura, los cierres de conexión y todo lo referente con el envío de estos datos, haciendo más fácil el trabajo. Tiene como principal desventaja la falta de soporte, desde el año 2003 su desarrollo está descontinuado.[6]

1.4.3 Biblioteca Zoidcom

Zoidcom es una biblioteca de red de alto nivel, esta biblioteca basada en el protocolo UDP provee ventajas para la replicación de objetos de videojuego y la sincronización de sus estados sobre la conexión de red de una manera altamente eficiente referente al ancho de banda. Esto se logra al multiplexar y desmultiplexar la información de los objetos desde y hasta secuencias de bits, lo que hace posible evitar el envío de datos redundantes. Tiene como principal desventaja su carácter no comercial, por lo que si se quiere emplear con este fin hay que pagar licencia.[7]

1.5 Middleware

Después de hacer un estudio de las principales bibliotecas existentes y determinar que ninguna de estas satisfacen las necesidades para solucionar los problemas que presenta este trabajo se decidió investigar sobre los middleware más importantes.

Se puede entender como middleware, un software de conectividad que hace posible que aplicaciones distribuidas pueden ejecutarse sobre distintas plataformas heterogéneas, es decir, sobre plataformas con distintos sistemas operativos, que emplean distintos protocolos de red y, que incluso, involucran distintos lenguajes de programación en la aplicación distribuida.

Desde otro punto de vista, un middleware se puede entender como una abstracción en la complejidad y en la heterogeneidad que las redes de comunicaciones imponen. De hecho, uno de los objetivos de un middleware es ofrecer un acuerdo en las interfaces y en los mecanismos de interoperabilidad, como contrapartida de los distintos desacuerdos en hardware, sistemas operativos, protocolos de red, y lenguajes de programación.

1.5.1 Corba

Unos de los middleware más renombrados es CORBA (Common Object Request Broker). Desde sus principios, el objetivo de CORBA fue permitir la interconexión abierta de distintos lenguajes,

implementaciones y plataformas. De esta forma, CORBA cumple con las cuatro propiedades enumeradas como deseables de los middleware. Para lograr estos objetivos, se decidió no establecer estándares binarios, todo está estandarizado para permitir implementaciones diferentes y permitir que aquellos proveedores que desarrollan CORBA puedan ofrecer valor agregado. La contrapartida es la imposibilidad de interactuar de manera eficiente a nivel binario. Todo producto que sea compatible con CORBA debe utilizar los costosos protocolos de alto nivel.

CORBA está constituido esencialmente de tres partes: un conjunto de interfaces de invocación, el ORB (object request broker) y un conjunto de adaptadores de objetos. CORBA va más allá de simples servicios middleware, provee una infraestructura para construir aplicaciones orientadas a objetos. Las interfaces definen los servicios que prestan los objetos, el ORB se encarga de la localización e invocación de los métodos sobre los objetos y el adaptador de objeto es quien enlaza la implementación del objeto con el ORB.

Para que las interfaces de invocación y los adaptadores de objetos funcionen correctamente, se deben cumplir dos requisitos importantes. En primer lugar, las interfaces de los objetos deben describirse en un lenguaje común. En segundo lugar, todos los lenguajes en los que se quieran implementar los objetos deben proveer un mapeo entre los elementos propios del lenguaje de programación y el lenguaje común. La primera condición permite generalizar los mecanismos de pasaje de parámetros. La segunda permite relacionar llamadas de un lenguaje en particular con el lenguaje de especificación común. Este lenguaje común fue una parte esencial de CORBA desde sus orígenes y es conocido como el OMG IDL: Interface Definition Language.

CORBA automatiza muchas tareas comunes y “pesadas” de programación de redes tales como registro, localización y activación de objetos, manejo de errores y excepciones, codificación y decodificación de parámetros, y protocolos de transmisión. [8]

Los servicios de CORBA representan un conjunto de funcionalidades que complementan el desarrollo funcional básico de los objetos que dan lugar a una aplicación. Estos son:

- **Ciclo de Vida:** define operaciones para crear, copiar, mover y eliminar objetos.

- **Eventos:** permite registrarse para recibir eventos que pueden ser producidos por otros objetos bajo un modelo consumer/supplier.
- **Nombre:** permite localizar objetos por un nombre.
- **Registro:** permite localizar objetos por sus propiedades.
- **Persistencia:** ofrece una interfaz para almacenar objetos.
- **Transacciones:** proporciona coordinación transaccional.
- **Concurrencia:** proporciona un gestor de bloqueos.
- **Externalización:** permite obtener y producir datos como streams (flujos).
- **Seguridad:** da soporte a la autenticación, listas de control de acceso, confidencialidad.
- **Tiempo:** permite definir y gestionar eventos temporizados.
- **Propiedades:** permite asociar propiedades a un objeto.
- **Encuesta:** ofrece una interfaz SQL para realizar operaciones en objetos.
- **Licencia:** ayuda a medir el uso de los objetos.
- **Relaciones:** permite crear asociaciones dinámicas entre objetos.
- **Colección:** proporciona las interfaces para crear y manipular colecciones de objetos (listas, conjuntos, entre otros.).

Ventajas de CORBA:

- Interfaz independiente del lenguaje de programación.
- Integración de herencia.

- Infraestructura de objetos distribuidos.
- Transparencia en la localización.
- Transparencia en la red.
- Comunicación directamente con los objetos.
- Interfaz de invocación dinámica.

1.5.2 Ice

El Internet Communication Engine (ICE) es un estándar desarrollado por ZeroC para el desarrollo de aplicaciones basada en objetos distribuidos, surge como alternativa de CORBA y difiere de él en algunos conceptos claves. ICE es un middleware ligero, abierto y orientado a objetos, es decir, ICE proporciona herramientas, APIs y soporte de bibliotecas para construir aplicaciones cliente/servidor orientadas a objetos. Una aplicación ICE se puede usar en entornos heterogéneos: los clientes y los servidores pueden escribirse en diferentes lenguajes de programación (C, C++, Java, C#, Visual Basic, Phyton, PHP), pueden ejecutarse en distintos sistemas operativos (Windows, Linux, y otros) y en distintas arquitecturas, y pueden comunicarse empleando diferentes tecnologías de red. Además, el código fuente de estas aplicaciones puede portarse de manera independiente al entorno de desarrollo.[9]

Clientes y servidores

Los términos cliente y servidor no están directamente asociados a dos partes distintas de una aplicación, sino que más bien hacen referencia a los roles que las diferentes partes de una aplicación pueden asumir durante una petición:

- Los clientes son entidades activas, es decir, emiten solicitudes de servicio a un servidor.
- Los servidores son entidades pasivas, es decir, proporcionan un servicio en respuesta a las solicitudes de los clientes.

Normalmente, los clientes no son clientes puros en el sentido de que sólo solicitan peticiones. En su lugar, los clientes suelen ser entidades híbridas que asumen tanto el rol de cliente como el de servidor.

Proxies

Para que un cliente sea capaz de comunicarse con un objeto ICE ha de tener acceso a un proxy para el objeto ICE. Un proxy es un componente local al espacio de direcciones del cliente, y representa al (posiblemente remoto) objeto ICE para el cliente. Además, actúa como el embajador local de un objeto ICE, de forma que cuando un cliente invoca una operación en el proxy, el núcleo de ejecución de ICE:

- Localiza al objeto ICE.
- Activa el servidor del objeto ICE si no está en ejecución.
- Activa el objeto ICE dentro del servidor.
- Transmite los parámetros de entrada al objeto ICE.
- Espera a que la operación se complete.

Invocación de métodos síncrona

Por defecto, el modelo de envío de peticiones utilizado por ICE está basado en la llamada síncrona a procedimientos remotos: una invocación de operación se comporta como una llamada local a un procedimiento, es decir, el hilo del cliente se suspende mientras dure la llamada y se vuelve activar cuando la llamada se ha completado (y todos los resultados están disponibles).

Invocación de métodos asíncrona

ICE también proporciona invocación de métodos asíncrona (asynchronous method invocation) o AMI: un cliente puede invocar operaciones de manera asíncrona, es decir, el cliente utiliza un proxy de la manera habitual para invocar una operación pero, además de pasar los parámetros necesarios, también pasa un objeto de retrollamada (callback object) y retorna de la invocación inmediatamente. Una vez que la

operación se ha completado, el núcleo de ejecución de la parte del cliente invoca a un método en el objeto de retrollamada pasado inicialmente, proporcionando los resultados de la operación a dicho objeto (o en caso de error, provee la información sobre la excepción asociada).

El servidor no es capaz de diferenciar una invocación asíncrona de una síncrona. De hecho, en ambos casos el servidor simplemente aprecia que un cliente ha invocado una operación en un objeto.

El tratamiento de métodos asíncrono es útil, por ejemplo, si un servidor ofrece operaciones que bloquean a los clientes por un periodo largo de tiempo. Por ejemplo, el servidor puede tener un objeto con una operación Get que devuelva los datos de una fuente de datos externa y asíncrona, lo cual bloquearía al cliente hasta que los datos estuvieran disponibles.

Adaptador de objetos

El adaptador de objetos es una parte de la API de ICE específico al lado del servidor: sólo los servidores utilizan los adaptadores de objetos. Un adaptador de objetos tiene varias funciones, como traducir las peticiones de los clientes a los métodos específicos al lenguaje de programación empleado, asociarse a uno o más puntos finales de transporte, o crear los proxies que pueden proporcionarse a los clientes.

Las interfaces, las operaciones, y los tipos de datos intercambiados entre el cliente y el servidor se definen utilizando el lenguaje Slice (Specification Language for ICE). Este permite definir el contrato entre el cliente y el servidor de forma independiente del lenguaje de programación empleado. Las definiciones Slice se compilan por un compilador a una API para un lenguaje de programación específico, es decir, la parte de la API que es específica a las interfaces y los tipos que previamente han sido definidos y que consisten en código generado.

El núcleo de ICE proporciona una sofisticada plataforma cliente-servidor para el desarrollo de aplicaciones distribuidas. También maneja una serie de servicios que gestionan la activación de servidores bajo demanda, la distribución de proxies a los clientes, la distribución de eventos asíncronos, la configuración de aplicaciones, entre otros. Manteniendo estos servicios disponibles como parte de la plataforma, permite al desarrollador centrarse en el desarrollo de la aplicación en lugar de construir la infraestructura necesaria en primer lugar. Algunos de estos servicios son (5):

- **IceGrid:** es la implementación de un servicio de localización ICE.
- **IceBox:** es un único servidor de aplicaciones que permite gestionar el arranque y la parada de un determinado número de componentes de aplicación.
- **IceStorm:** es un servicio de publicación-subscripción que actúa como un distribuidor de eventos entre servidores y clientes. Los publicadores envían eventos al servicio IceBox, el cual notifica a los subscriptores. De esta forma, un único evento publicado por el publicador se puede enviar a múltiples subscriptores. Los eventos están categorizados en función de un tema, y los subscriptores especifican los temas en los que están interesados. IceBox permite seleccionar entre distintos criterios de calidad de servicio para que las aplicaciones puedan obtener una relación fiabilidad-rendimiento apropiada.
- **IcePatch2:** es un servicio que permite la fácil distribución de actualizaciones de software a los clientes.
- **Glacier2:** es el servicio de cortafuegos de ICE, el cual permite que tanto los clientes como los servidores se comuniquen de forma segura a través de un cortafuegos sin comprometer la seguridad. El tráfico entre el cliente y el servidor queda completamente encriptado utilizando certificados de clave pública y es bidireccional. Glacier2 también proporciona soporte para la autenticación mutua y para la gestión segura de sesiones.
- **IceSSL:** Un SSL dinámico que transporta extensiones para el núcleo de ICE. Ofrece autenticación, cifrado e integridad en los mensajes utilizando el protocolo estándar de la industria SSL.

La arquitectura propuesta por ICE proporciona una serie de beneficios a los desarrolladores de aplicaciones:

- Mantiene una semántica orientada a objetos.
- Proporciona un manejo síncrono y asíncrono de mensajes.
- Soporta múltiples interfaces.
- Es independiente de la máquina.

- Es independiente del lenguaje de programación.
- Es independiente de la implementación, es decir, el cliente no necesita conocer cómo el servidor implementa sus objetos.
- Es independiente del sistema operativo.
- Soporta el manejo de hilos.
- Es independiente del protocolo de transporte empleado.
- Mantiene transparencia en lo que a localización y servicios se refiere (IceGrid).
- Es seguro (SSL y Glacier2).
- Permite hacer persistentes las aplicaciones (Freeze).
- Tiene disponible el código fuente.

1.6 Fundamentación de la Tecnología

1.6.1 Biblioteca gráfica Ogre

Ogre (Object Oriented Graphics Engine) es un motor de gráficos en tres dimensiones multiplataforma, o sea, permite crear aplicaciones para Windows, Linux o Mac .La principal ventaja que presenta sobre otros motores 3D es que es gratuito y apenas existen exigencias para su uso.

Desde el principio, Ogre fue diseñado bajo la filosofía orientada a objetos, por lo que su interfaz es clara, intuitiva y fácil de utilizar, con Ogre se puede hacer videojuegos o cualquier tipo de aplicación que requieran gráficos tridimensionales que tengan poco que envidiar a los realizados por la mayoría de los motores del mercado.

Ogre no es un motor diseñado sólo con los videojuegos en mente, es un motor de gráficos 3D general. Sin embargo, puede ser fácilmente vinculado junto con otras bibliotecas para crear un motor de videojuego. Entre los módulos que se necesitan para crear un videojuego están:

- Módulo de Sonido.
- Módulo de Red.
- Módulo de Física.
- Módulo de Entrada/Salida.
- Módulo de Inteligencia Artificial.

Ogre no incluye estas bibliotecas nativas, a pesar de que expone una interfaz que facilita la labor de Ogre en una aplicación existente.

Ogre es diferente a otros motores gráficos, prioriza el diseño frente a la acumulación de características. Todas las características se han considerado a fondo y se incluyen en el diseño general de la forma más elegante posible y siempre plenamente documentadas. Así, se consigue que todos los elementos del motor formen parte siempre de un conjunto coherente. Por consiguiente, la calidad es antepuesta a la cantidad. Como la cantidad se puede añadir con el paso del tiempo, se sigue la filosofía de ir poco a poco, pero con la seguridad de un trabajo bien hecho. Por ello, el núcleo del equipo de desarrollo se mantiene deliberadamente pequeño y todos sus miembros son veteranos ingenieros de software con muchos años de experiencia en el mundo real. Pero esto no impide que las actualizaciones de la comunidad sean bienvenidas, si bien, se someten a un estricto control de calidad y cohesión con la filosofía de Ogre antes de ser aceptadas.

Características Generales de Ogre:

- Diseño orientado a objetos. Interfaz simple y fácil de usar, diseñada para que requiera poco esfuerzo el renderizado de escenas en tres dimensiones.
- Arquitectura basada en plugins muy flexible que permite extender las funcionalidades del motor.
- Diseño limpio y bien documentado de las clases del motor.

Independiente de la API gráfica, se puede utilizar OpenGL o DirectX.

1.6.2 Lenguaje de Programación C++

C++ es un lenguaje orientado a objetos de probada eficacia para generar aplicaciones de alto rendimiento como deben ser las aplicaciones gráficas. Éste aporta un nivel de productividad muy superior a C, sin

comprometer la flexibilidad, el rendimiento o el control. Es uno de los lenguajes de sistemas más conocido en el mundo, altamente compatible y soporta las bibliotecas gráficas Glide, OpenGL, DirectX, G3D y Ogre que son muy utilizadas en la industria de los videojuegos. En general son considerables las ventajas que este lenguaje posee, al hacer uso del paradigma de la programación orientada a objetos.

1.6.3 Herramienta CASE Visual Paradigm

Visual Paradigm es una herramienta que sirve para realizar modelado UML siguiendo el estándar UML 2.1. Esta herramienta posee características graficas muy cómodas que facilitan la realización de los diagramas de modelado que sigue el estándar de UML que son:

- Diagramas de clase, Casos de Uso, Comunicación, Secuencia, Estado, Actividad, Componentes, entre otros.

Otras características importantes son:

- Integración con diversos IDE's como:
    - NetBeans(de Sun)
    - JDeveloper(de Oracle)
    - Eclipse (de IBM)
    - JBuilder (de Borland)
- Ingeniería Inversa para:
    - JAVA
    - NET
    - XML
    - Hibernate
- Exportación de imágenes jpg, png y svg(w3g estandar).

1.6.4 Metodologías para el desarrollo de software

Las metodologías y estándares utilizados en un desarrollo de software proporcionan las guías para poder conocer todo el camino a recorrer desde antes de empezar la implementación, con lo cual se asegura la calidad del producto final al captar las mejores prácticas que el estado actual de la tecnología permite. Dentro de las más usadas se encuentran:

eXtreme Programming (XP)

La programación extrema es una metodología reciente utilizada en el desarrollo de software. La filosofía de XP es satisfacer al completo las necesidades del cliente, por eso, lo integra como una parte más del equipo de desarrollo. Esta consiste en una programación rápida o extrema.

XP fue inicialmente creada para el desarrollo de aplicaciones dónde el cliente no tiene una concepción clara de las funcionalidades que tendrá la aplicación que se desarrollará. Este desconocimiento podría provocar un cambio constante en los requisitos que debe cumplir la aplicación por lo que es necesaria una metodología ágil como XP que se adapta a las necesidades del cliente y dónde la aplicación se va revisando constantemente.

XP está diseñada para el desarrollo de aplicaciones que requieren un grupo de programadores pequeño, dónde la comunicación sea más factible que en grupos de desarrollo grandes. La comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes.

Las principales características de esta metodología son las siguientes:

- Comunicación: Los programadores están en constante comunicación con los clientes para satisfacer sus requisitos y responder rápidamente a los cambios de los mismos. Muchos problemas que surgen en los proyectos se deben a que después de concretar los requisitos que debe cumplir el programa no hay una revisión de los mismos, pudiendo dejar olvidados puntos importantes.
- Simplicidad: Codificación y diseños simples y claros. Muchos diseños son tan complicados que cuando se requiere mantenimiento o ampliación resulta imposible hacerlo y se tienen que desechar y partir de cero.

- Realimentación (Feedback): Mediante la realimentación se ofrece al cliente la posibilidad de conseguir un sistema adecuado a sus necesidades. Se le va mostrando el proyecto a tiempo para sugerir cambios y poder retroceder a una fase anterior para rediseñarlo a su gusto.
- Tenacidad: Se debe ser tenaz para cumplir los tres puntos anteriores. Hay que tener valor para comunicarse con el cliente y enfatizar algunos puntos a pesar de que esto pueda dar sensación de ignorancia por parte del programador; hay que ser decidido para mantener un diseño simple y no optar por lo que pudiera parecer mejor o un camino más fácil y por último hay que enfatizar que la realimentación será efectiva.

Rational Unified Process (RUP)

El Proceso Racional Unificado o RUP (Rational Unified Process), es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP es un producto de Rational (IBM). Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, entre otros.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso). El RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al terminar cada ciclo, estos se dividen en fases (inicio, elaboración, construcción y transición) que finalizan con un hito donde se debe tomar una decisión importante. Se caracteriza por ser:

- Dirigido por casos de uso.
- Centrado en la arquitectura.
- Iterativo e Incremental.

Conclusiones del capítulo

A lo largo de este capítulo, como base para el estudio del tema en que se desenvolverá el trabajo, se introdujo al lector en una serie de conceptos básicos como arquitectura de red, protocolo TCP/IP, técnicas de transmisión de datos por la red y arquitecturas de comunicación, los que son fundamentales para entender lo expuesto. Además se estudiaron características distintivas de varias bibliotecas de red, las que constituyen la base para escoger la que más se ajusta a las necesidades de este trabajo. También se

abordó el tema de las metodologías de desarrollo que más se utilizan actualmente, así como las herramientas CASE.

Capítulo II

MODELADO DE LA SOLUCIÓN Y DISEÑO DEL SISTEMA

En este capítulo se proponen las soluciones técnicas para la creación del módulo de trasmisión de datos, los métodos específicos que se implementarán y las funcionalidades más adecuadas según los requerimientos del trabajo.

2.1 Propuesta de solución

Para solucionar los problemas de conectividad en los sistemas de Realidad Virtual que utilizan Ogre como motor gráfico, se propone realizar un módulo de red que brinde las funcionalidades necesarias para el envío, procesamiento y recepción de datos, que permita la interacción en tiempo real entre los usuarios.

Después de realizar un estudio de las bibliotecas y middleware más usados en sistemas de Realidad Virtual, se propone, por sus características, utilizar el middleware ICE como base para desarrollar la solución, ya que proporciona una implementación eficiente en ancho de banda, en uso de memoria y en carga de CPU, proporciona una implementación basada en la seguridad, de forma que se pueda usar sobre redes no seguras.

Como solución se desarrollará un módulo de red haciendo uso de la tecnología middleware ICE y utilizará la arquitectura cliente/servidor permitiendo establecer comunicación, así como enviar y recibir datos entre el emisor y el receptor. Este módulo permitirá que el cliente envíe por la red la información referente a sus entidades, dicha información se almacena en el servidor, cuando el cliente va a renderizar su escena pide al servidor la información almacenada de todos sus clientes para actualizar la escena, de esta manera se logra una sincronización visual en todos los clientes conectados al servidor.

Se utilizará la metodología de desarrollo XP, ya que está revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento

no la ven como alternativa para las metodologías tradicionales. Además es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos de corto plazo, pequeño equipo y cuyo plazo de entrega era “ayer”. Para el modelado de la solución se utilizará el lenguaje de modelado UML. Se modelará sobre la herramienta CASE Visual Paradigm Studio, específicamente el Visual Paradigm for UML 6.4. El ambiente de desarrollo será Code::Blocks y el lenguaje de programación C++. Todo esto se desarrollará sobre el sistema operativo GNU/Linux.

Antes de continuar es necesario analizar algunos conocimientos relacionados con la tecnología que se empleará.

ICE, como la mayoría de los middleware, trata de acercar el modelo de programación a un punto de vista local, es decir, enmascarando la llamada a los procedimientos remotos. Esto consiste en el envío de un mensaje por la red mediante un proxy creado por un cliente y el recibo de este mensaje por parte de un esqueleto en un servidor que se encarga de traducirlo a invocaciones sobre un objeto en cuestión. Lo cual permite que el cliente pueda realizar invocaciones de procedimientos remotos, que no es más que una petición de un cliente, que ha de terminar en la ejecución de un determinado código en el servidor. Estas invocaciones pueden tener o no parámetros de entrada y de salida.

2.2 Requerimientos de software

La IEEE Standard Glossary of Software Engineering Terminology define un requerimiento como condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo y que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente. Los requisitos se pueden clasificar en funcionales y no funcionales. Los funcionales son capacidades o condiciones que el sistema debe cumplir y los no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

2.2.1 Requerimientos funcionales

1. Crear los datos a enviar.

Requerimientos funcionales del subsistema Cliente:

2. Abrir conexión.
3. Detectar servidores.
4. Terminar la conexión con el servidor.

Requerimientos funcionales del subsistema Servidor:

5. Abrir conexión.
6. Aceptar la conexión de un nuevo cliente.
7. Detectar desconexiones de los clientes.
8. Desconectar cliente.
9. Calcular latencia.

2.2.2 Requerimientos no funcionales

Diseño e implementación

- Lenguaje de programación C++.
- Se utilizará la herramienta de desarrollo integrado Code::Blocks.
- Se utilizará el middleware ICE.

Software

- La aplicación se ejecutará sobre los sistemas operativos GNU/Linux y Windows (versiones superiores a Windows 2000)

Seguridad.

- Los paquetes antes de ser enviados deben ser encriptados.

2.2.3 Exploración.

La exploración es la etapa del proceso de desarrollo de software que propone XP para comenzar la construcción de un producto. Una vez que los clientes entregan su propuesta al equipo de trabajo, comienza el análisis en grupo, las horas en los pizarrones, las tormentas de ideas y la conceptualización del software. Un aspecto clave en el esclarecimiento de las dudas sobre los procesos a automatizar es que, desde el mismo inicio del proyecto, un miembro del equipo de trabajo del cliente, se adiciona al de desarrolladores.

2.2.4 Historia de usuario

Como define la metodología XP, primeramente se realizarán las Historias de usuario, estas tienen la misma finalidad que los casos de uso pero con algunas diferencias: Constan de tres ó cuatro líneas escritas por el cliente (en este caso serán realizadas por los autores) en un lenguaje no técnico sin hacer mucho hincapié en los detalles, no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, entre otros.

Tabla 1 Historia de usuario Abrir conexión.

Historia de Usuario	
1. Nombre: Abrir conexión.	
Usuario: Servidor	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta
Estimación: 7 días	**Iteración Asignada**: 1
Descripción: El servidor abre el puerto de conexión, se pone en línea para esperar cualquier conexión por parte de algún cliente.	
Información adicional (Observaciones):	

Tabla 2 Historia de usuario: Abrir conexión del cliente.

Historia de Usuario	
2. Nombre: Abrir conexión.	
Usuario: Cliente	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta
Estimación: 3 días	**Iteración Asignada**: 1
Descripción: El cliente abre un puerto, se pone en línea listo para conectarse a un servidor.	
Información adicional (Observaciones):	

Tabla 3 Historia de usuario: Crear los datos a enviar.

Historia de Usuario	
3. Nombre: Crear los datos a enviar	
Usuario: Usuario que va utilizar la aplicación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta
Estimación: 5 días	**Iteración Asignada**: 1
Descripción: El usuario define una estructura única para cada dato que desee enviar utilizando el módulo de red.	
Información adicional (Observaciones):	

Tabla 4 Historia de Usuario: Aceptar conexión de un nuevo cliente.

Historia de Usuario	
4. Nombre: Aceptar la conexión de un nuevo cliente.	
Usuario: Servidor	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta
Estimación: 4 días	**Iteración Asignada**:2
Descripción Se crea una sesión entre el cliente y el servidor.	
Información adicional (Observaciones):	

Tabla 5 Historia de usuario: Calcular latencia.

Historia de Usuario	
5. Nombre: Calcular latencia.	
Usuario: Cliente	
Prioridad en el Negocio: Alta	**Prioridad en el Negocio**: Alta
Estimación: 1 día	**Iteración Asignada**:2
Descripción: El cliente calcula el tiempo que demora el servidor en darle respuesta.	

Información adicional (Observaciones):

Tabla 6 Historia de usuario: Desconectar un cliente.

Historia de Usuario	
6. Nombre: Desconectar un cliente.	
Usuario: Servidor	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta
Estimación: 3 días	**Iteración Asignada**: 3
Descripción: El servidor cierra la sesión con el cliente por determinadas razones(exceso latencia, petición del cliente, pérdida de la conexión).	
Información adicional (Observaciones):	

Tabla 7 Historia de usuario: Desconectarse del servidor.

Historia de Usuario	
7. Nombre: Desconectarse del servidor.	
Usuario: Cliente	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta
Estimación: 1 día	**Iteración Asignada**: 3

Descripción: A petición del cliente se cierra la sesión con el servidor, cerrando la conexión entre ambos.
Información adicional (Observaciones):

Tabla 8 Historia de usuario: Detectar desconexiones de los clientes.

Historia de Usuario	
8. Nombre: Detectar desconexiones de los clientes	
Usuario: Servidor	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta
Estimación: 1 semana	**Iteración Asignada**:4
Descripción: El servidor chequea mediante un timestamp cada sesión abierta para verificar si los clientes están realmente conectados (online).	
Información adicional (Observaciones):	

Tabla 9 Historia de usuario: Detectar servidores.

Historia de Usuario	
9. Nombre: Detectar servidores.	
Usuario: Cliente	
Prioridad en el Negocio: Alta	**Nivel de Complejidad**: Alta

Estimación: 1 semana	**Iteración Asignada**: 4
Descripción: El cliente envía un mensaje de difusión a la subred en que se encuentra y los servidores que estén disponibles les responderán.	
Información adicional (Observaciones):	

2.2.5 Planificación.

Durante la fase de planificación se realiza una estimación del esfuerzo que costará implementar todas las historias de usuario juntas, partiendo de la que se le asignó a cada una en la fase de exploración. Una vez terminado esto, se procede a organizarlas en las iteraciones correspondientes, teniendo en cuenta la prioridad especificada por el cliente y del tiempo de desarrollo de cada una.

2.2.6 Iteraciones.

Una iteración no es más que un mini-proyecto que se realiza. Al finalizar cada iteración, se obtiene un resultado en software con valor para el cliente. Claro está que este quedará totalmente satisfecho al finalizar la última iteración, debido a que es la que concluye y completa el producto acordado inicialmente.

En la organización de las iteraciones no es recomendable extenderlas más de 1 mes laboral, que por lo general se extiende 20 o 21 días. Los plazos de entrega o retroalimentación atentan contra el cumplimiento de los objetivos del cliente, sometidos a cambios pequeños de manera constante. Esto sucede, aun así, cuando un miembro de su equipo de trabajo se encuentre incluido en el equipo de desarrollo.

Tabla 10 Muestra las iteraciones.

Iteraciones	Orden de las Historias de Usuario a implementar	Cantidad de tiempo de trabajo
Iteración 1	1- Abrir conexión en el servidor.	3 semanas

	2- Abrir conexión en el cliente. 3- Crear el paquete con los datos a enviar.	
Iteración 2	4- Aceptar la conexión de un nuevo cliente. 5- Calcular latencia.	2 semanas
Iteración 3	6- Desconectar un cliente. 7- Terminar la conexión con el servidor.	2 semanas
Iteración 4	8- Detectar desconexiones de los clientes. 9- Detectar servidores.	3 semanas

2.2.7 Plan de entregas.

El plan de entregas es el compromiso final del equipo de desarrollo con los clientes. Es una cuestión de vital importancia para el negocio entre ambas partes, ya que la entrega tardía o temprana de la solución, repercute notablemente en la economía y moral de todos los involucrados. La estimación es uno de los temas más complicados del desarrollo de un proyecto de software y es por ello que resulta de vital importancia tener bien claros los requerimientos del cliente, el estilo de trabajo del equipo de desarrollo y el tiempo con que dispone el cliente para tener en sus manos la solución.

2.3 Arquitectura del sistema.

La arquitectura que se va a utilizar es la arquitectura cliente/servidor, justificada por varios factores: primero, es la arquitectura más popular en el área de desarrollo de videojuegos en la actualidad, esto a su vez se debe a que esta es una arquitectura simple de implementar en relación con otras. Por otra parte, el propio modelo es muy adecuado para ser del tipo centralizada, debido a las características especiales del servidor, el cual manipula y procesa toda la información que se intercambia entre los clientes. La centralización proporciona la ventaja de garantizar una alta consistencia.

Estructura lógica interna de los clientes y servidores en Ice**:**

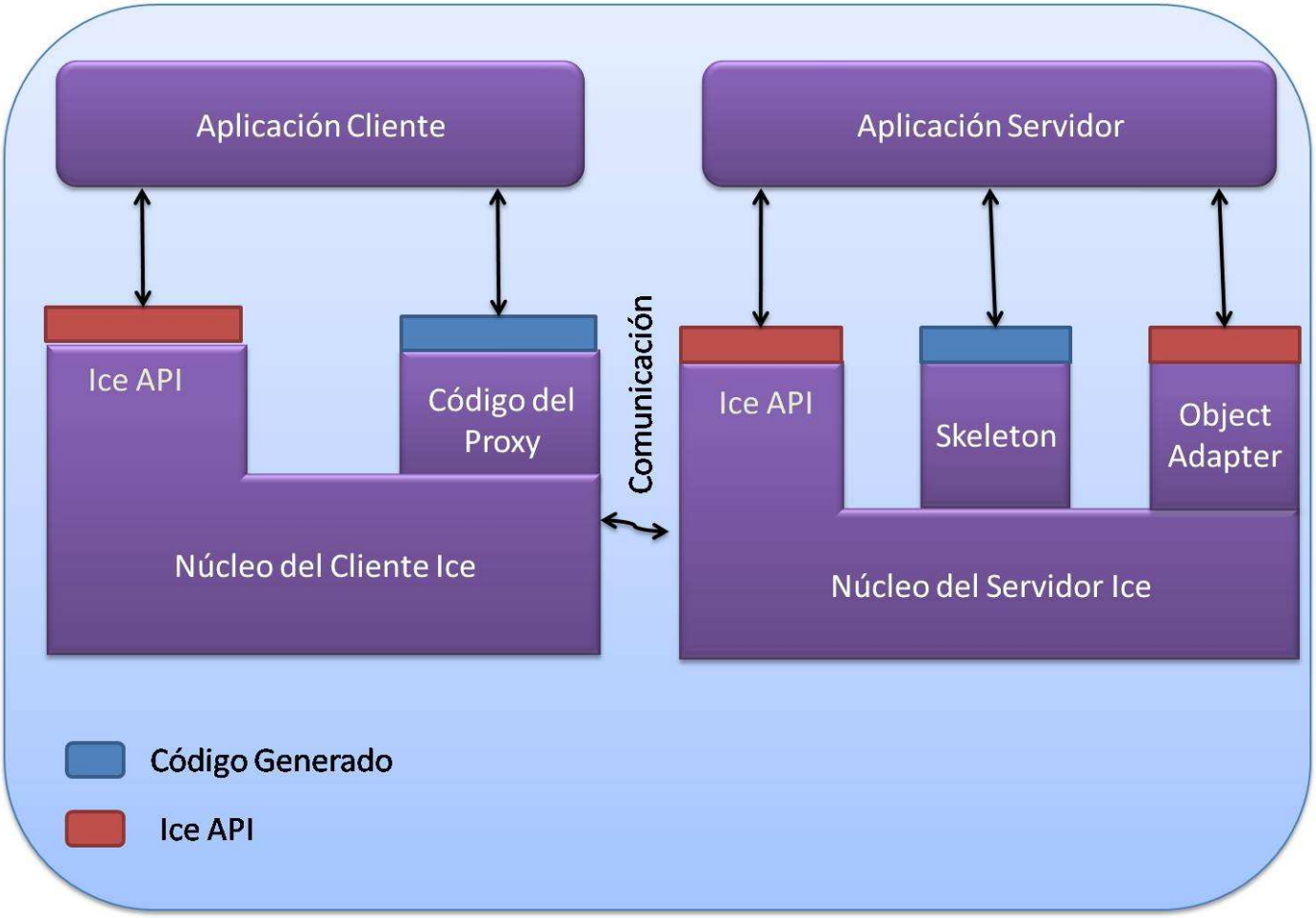


Figura 4 Estructura lógica interna de Ice.

2.4 Descripción de las clases del diseño

Middleware Technology (Tecnología Middleware) se encargará de la distribución de los datos y abstraerá a las aplicaciones de los protocolos de red. En él existirá una estructura de datos genérica, con la cual será posible combinar cualquier tipo de dato, lo que permitirá crear los datos deseados. También se definirán en este las interfaces de comunicación las cuales permitirán el envío y recepción de dichos datos.

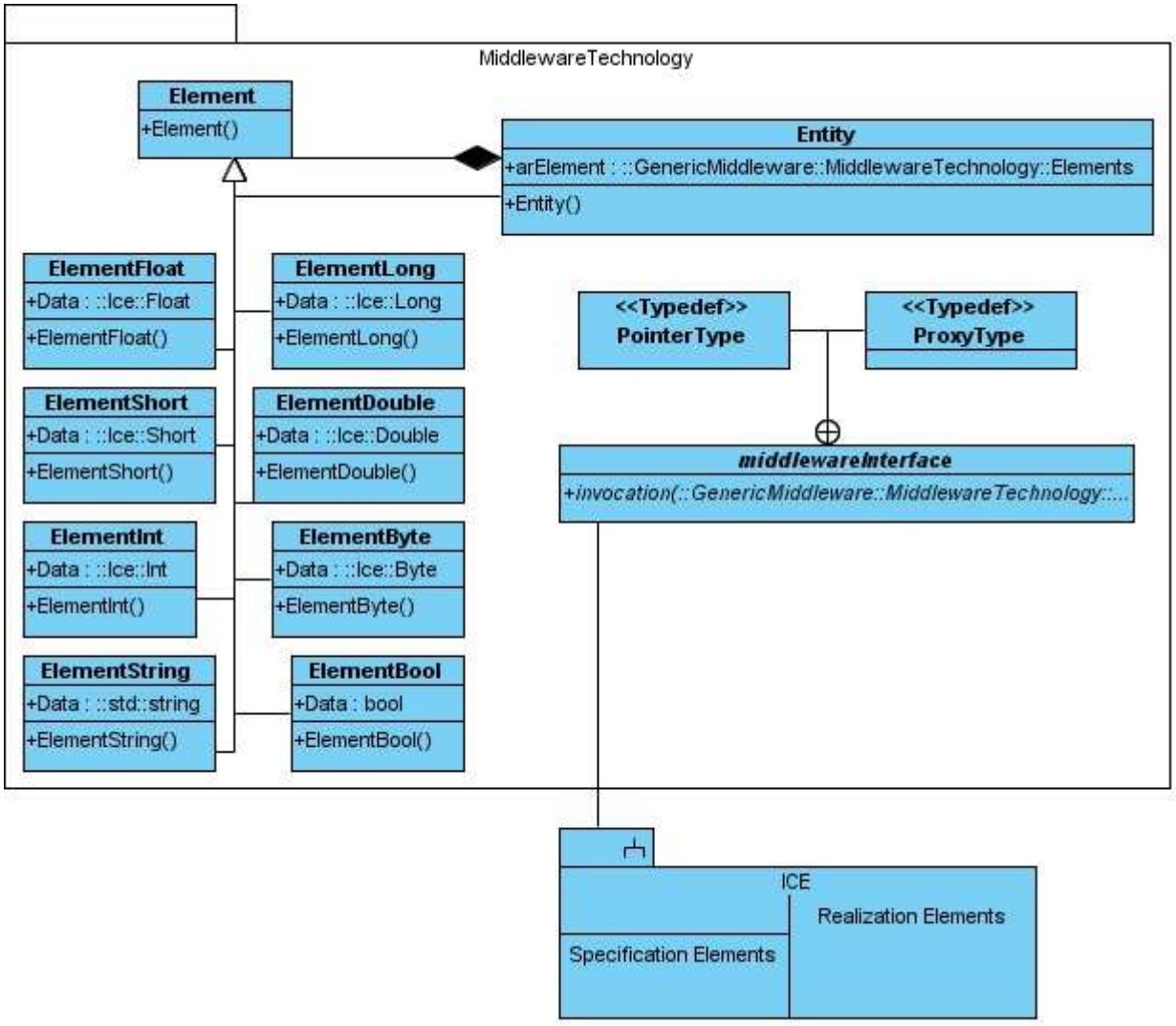


Figura 5 Diagrama de clase del paquete Middleware Technology.

Con este módulo se vinculará el módulo Communication Model (Modelo de comunicación) que definirá la forma en que se envían y reciben los datos. Para lograr esto se definió la clase CommunicationModel, en la cual existe un hilo principal destinado al envío y recibo. A partir de este hilo se genera además un hilo secundario, el objetivo de este último es establecer la comunicación, la cual puede ser cliente o servidor, esto posibilita que se pueda enviar o recibir siempre que la comunicación esté abierta. De esta clase van a heredar las clases Sender, que tiene un proxy para enviar las invocaciones y Receiver que además de tener un adaptador de objetos para recibirlas, tiene una lista de procedimientos almacenados. De las

clases Sender y Receiver heredan Client y Server respectivamente, donde se define como modelo de comunicación cliente/servidor. Del módulo Communication Model dependerán los módulos Invoker y Procedure.

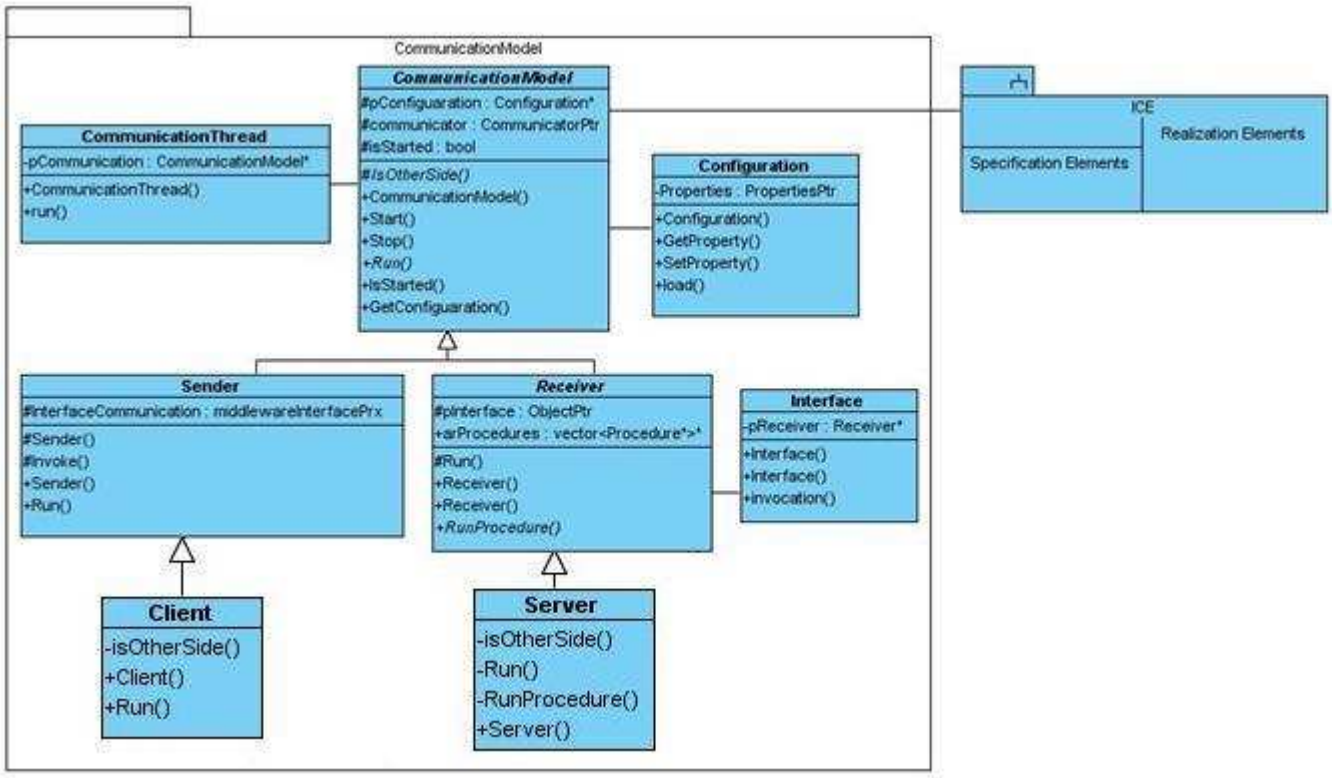


Figura 6 Diagrama de clase del paquete CommunicationModel.

El objetivo del módulo Invoker es definir los tipos de invocaciones que pueden realizar los emisores, para esto se define una clase con su mismo nombre, quien está estrechamente relacionada con Sender, pues es la única que puede invocar los procedimientos. Los tipos de invocaciones son: sin respuesta y con respuesta (oneway y twoway), el primer caso significa que no recibirá respuesta la invocación realizada, o sea, no tendrá parámetros de salida. El segundo caso si tendrá parámetro de salida la invocación de procedimiento, este envío se realiza de forma asíncrona. Estos parámetros son definidos por el desarrollador y pueden ser del tipo deseado, incluso hasta excepciones. La clase Invoker utiliza un emisor para enviar las invocaciones de procedimientos y en el caso específico de invocación con respuesta, va a utilizar al mismo tiempo un receptor para recibir la respuesta de dicha invocación.

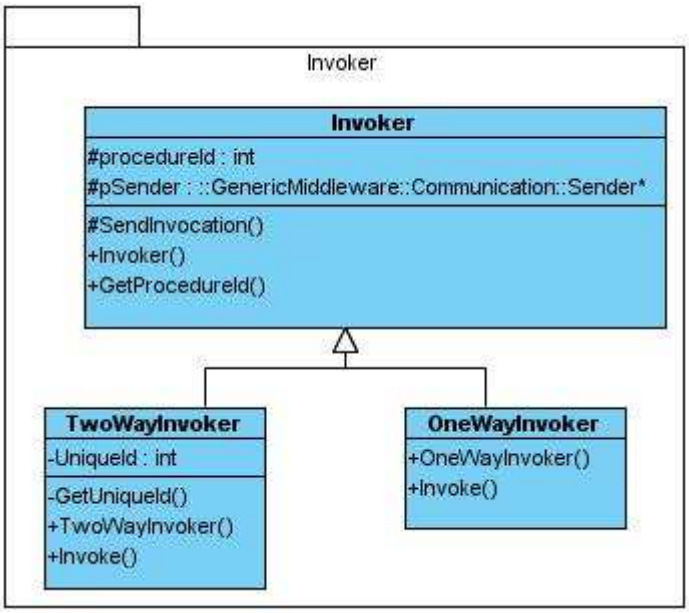


Figura 7 Diagrama de clase del paquete Invoker.

Para definir el tipo de procedimiento (con respuesta o sin respuesta) con que cuentan los receptores, se define el módulo Procedure, en él se especifica la clase Procedure, quien esta relacionada con la clase Receiver ya que tiene los procedimientos a ejecutar. En el caso de los procedimientos con respuesta, la clase TwoWayProcedure va a usar un emisor para enviarla.

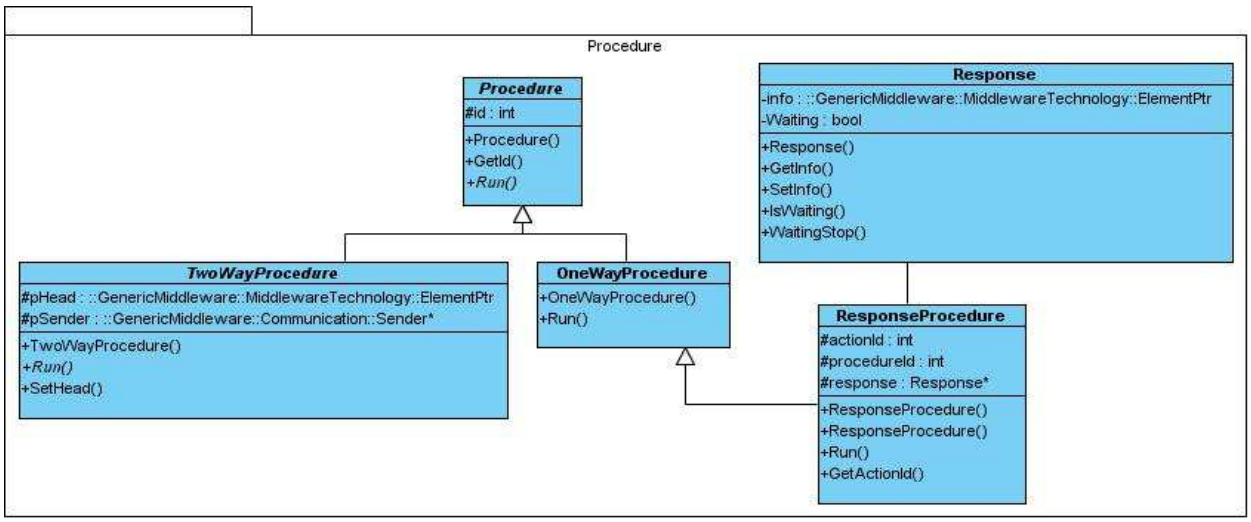


Figura 8 Diagrama de clase del paquete Procedure.

Hasta el momento se tiene el diseño de un middleware capaz de enviar y recibir cualquier invocación de procedimiento, pero para que los sistemas que lo utilicen puedan enviar y recibir sus datos, necesitarán definirlos, es decir, especificar una estructura única para cada uno de ellos. Estos se definen en el módulo Demo Definition (Definiciones del demo), utilizando la estructura genérica brindada en el módulo Middleware Technology. Este módulo será fácil de entender y a su vez sencillo de implementar. Para este demo solo se especificó el dato Vector_3D, que será enviado al servidor a través de la invocación EnviarVector. El servidor almacena el valor de este vector ejecutando el procedimiento RecibirVector. Para el cliente poder saber el valor de este vector almacenado debe ejecutar la invocación Consultar, que será atendida por el servidor mediante el procedimiento AtenderConsulta.

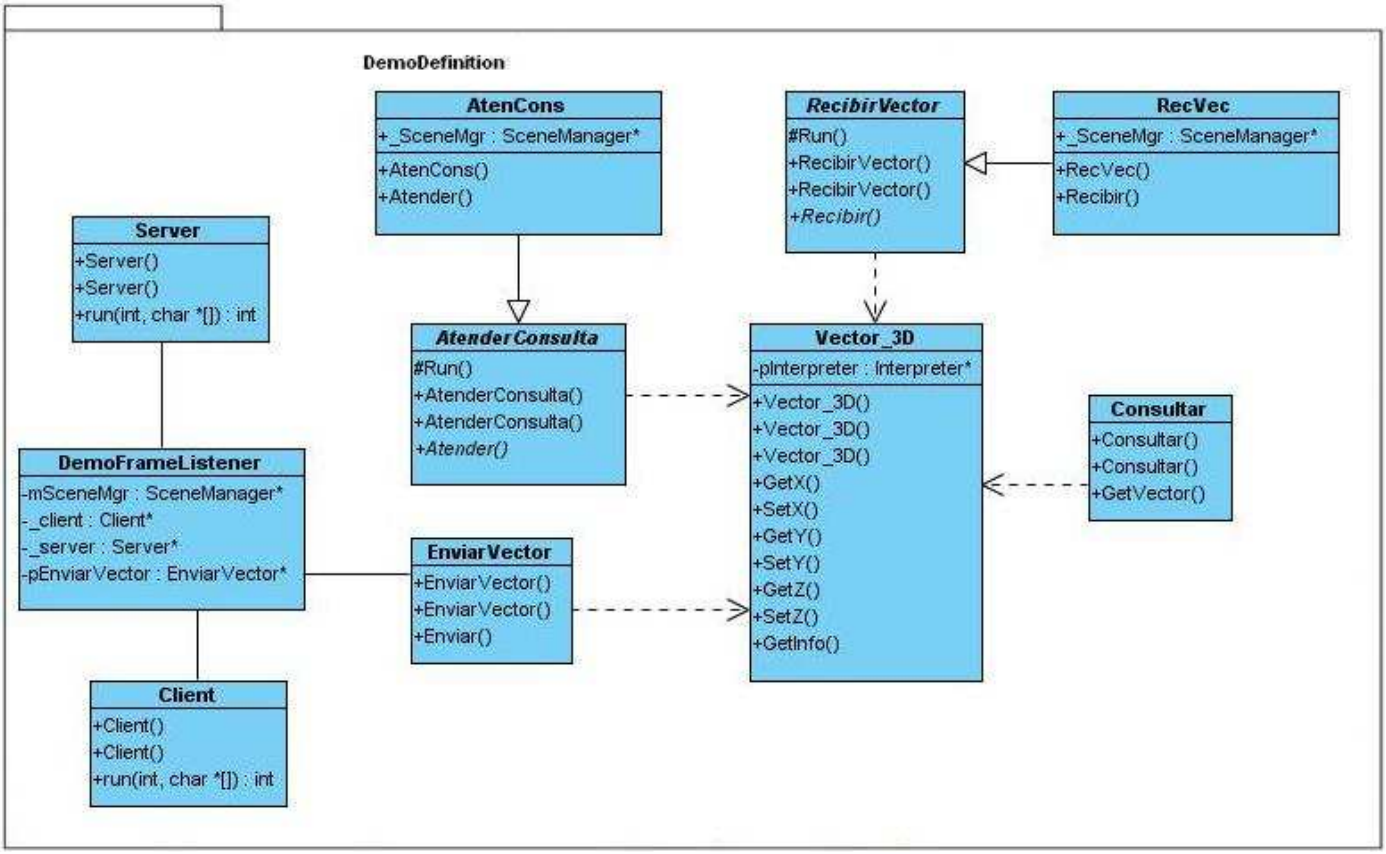
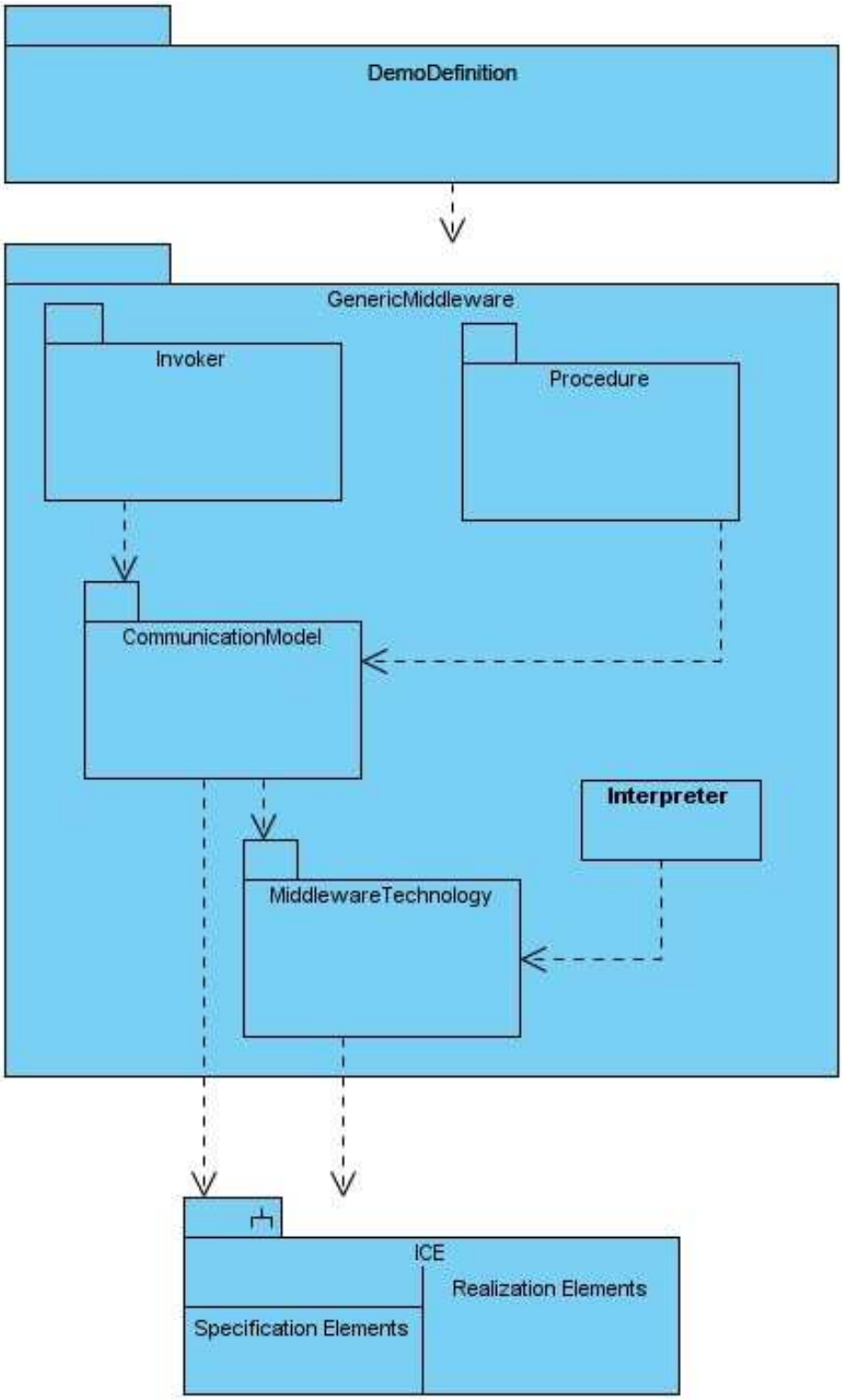


Figura 9 Diagrama de clase del paquete Demo Definition.

En el diseño del Middleware se definieron varios paquetes. Como se había visto anteriormente, con la tecnología middleware que se escogió (ICE), se desarrollará una parte del software de comunicación, este será MiddlewareTechnology. Communication Model permite definir el modelo de comunicación que se usará, Invoker y Procedure permiten especificar los tipos de invocaciones y procedimientos que será posible usar. Los paquetes mencionados anteriormente formaran parte del middleware genérico por lo que estarán dentro del paquete GenericMiddleware. Y para definir los datos para la integración con el motor gráfico Ogre se creó el paquete DemoDefinition.


DemoDefinition
GenericMiddleware
Invoker
Procedure
CommunicationModel
Interpreter
MiddlewareTechnology
ICE
Realization Elements
Specification Elements

La aplicación de esta arquitectura beneficiará al sistema. Este estilo de arquitectura permite asilar los cambios en tecnologías al módulo Tecnología Middleware ICE para reducir el impacto en el sistema total, esto se pondría en práctica en caso que este módulo deba ser desarrollado nuevamente pero utilizando otra tecnología middleware. Las funcionalidades estarán claramente limitadas y el diseño precisa la separación entre la funcionalidad de cada módulo.

2.4.1 Patrones de diseño

Un patrón es un conjunto de información que proporciona respuesta a un conjunto de problemas similares, o sea, un patrón es una solución a un problema en un contexto, donde:

- Contexto son las situaciones recurrentes a las que es posible aplicar el patrón.
- Problema es el conjunto de metas y restricciones que se dan en ese contexto.
- Solución es el diseño a aplicar para conseguir las metas dentro de las restricciones.

Esto proporciona una serie de ventajas entre las cuales se encuentran, la producción de software más resistente al cambio, ayudan a especificar interfaces, permite la reutilización del código y un uso de documentación estándar.

Existen dos familias de patrones de diseño, estos son los GRASP y los GoF. Los GRASP como sus siglas en inglés definen (General Responsability Assignment Software Patterns) son los patrones de software para la asignación general de responsabilidad y describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. GoF es el acrónimo de "The Gang of Four" (la banda de los cuatro). Sus autores son Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, los cuales subdividieron los patrones en tres clases: los patrones creacionales, los estructurales y los de comportamiento.

Patrón polimorfismo

El polimorfismo tiene varios significados relacionados. En este contexto significa “asignar el mismo nombre a servicios en diferentes objetos” cuando los servicios son parecidos o están relacionados. Es decir cuando las alternativas o comportamientos relacionados varían según el tipo (clase), asigne la responsabilidad para el comportamiento, utilizando operaciones polimórficas, a los tipos para los que varía

el comportamiento. El empleo de este patrón se ve evidenciado en las clases (OneWayInvoker y TwoWayInvoker que heredan de Invoker) y trae como beneficio que se puedan añadir fácilmente las extensiones necesarias para nuevas variaciones y las nuevas implementaciones se pueden introducir sin afectar a los clientes.

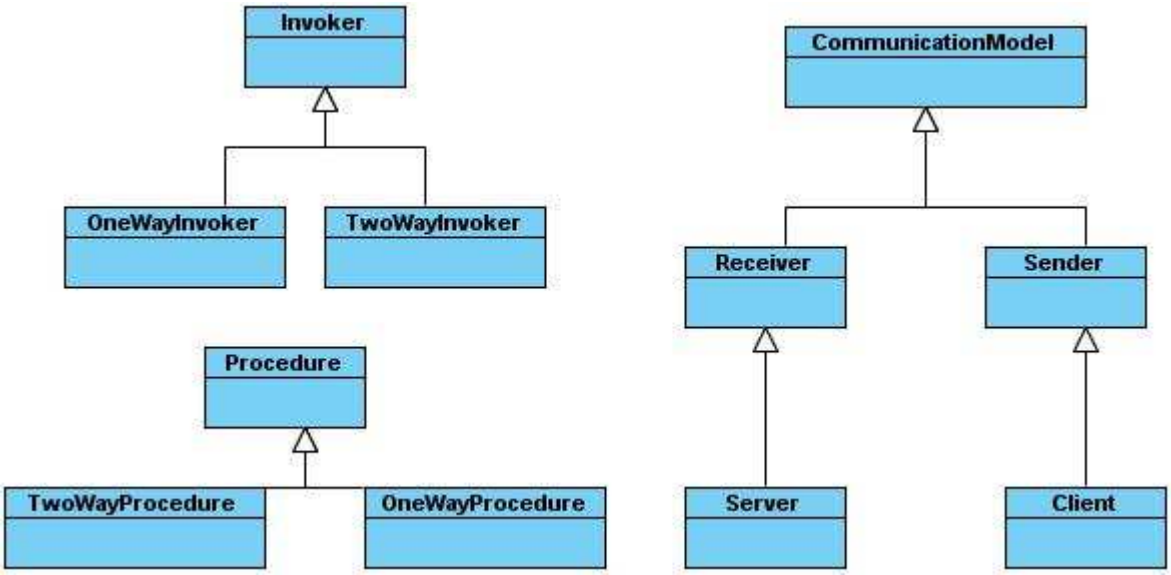


Figura 10 Evidencia del patrón polimorfismo en el sistema.

Patrón Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se deba conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. El uso de este patrón se evidencia en la clase Interpreter, ella es la encargada de crear instancias de las clases que heredan de Element ya que agrega a la clase y es un experto en la creación de dichas clases.

Patrón Compuesto.

Es uno de los patrones GoF. Es un patrón estructural y sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de

árbol . Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera. El uso de este patrón se pone en práctica en la siguiente jerarquía, donde la clase Element declara la interfaz para los objetos en la composición, implementa comportamientos por defecto comunes para todas las clases apropiadamente. ElementInt y ElementString son clases de tipo hojas y definen un comportamiento para un tipo de dato básico. Entity define el comportamiento de los nodos del árbol que poseen hijos, representa cualquier atributo compuesto del dato que se quiera definir.

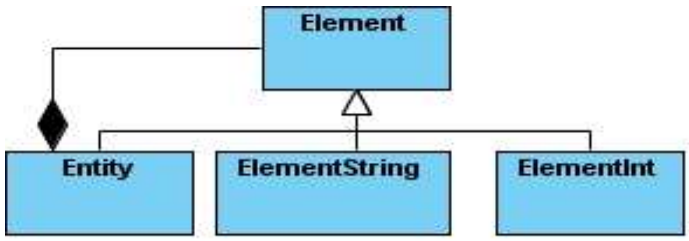


Figura 11 Evidencia del patrón Compuesto en el sistema.

Diagrama de Despliegue

El Diagrama de Despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se emplea para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. Los elementos usados por este tipo de diagrama son nodos, componentes y asociaciones.

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

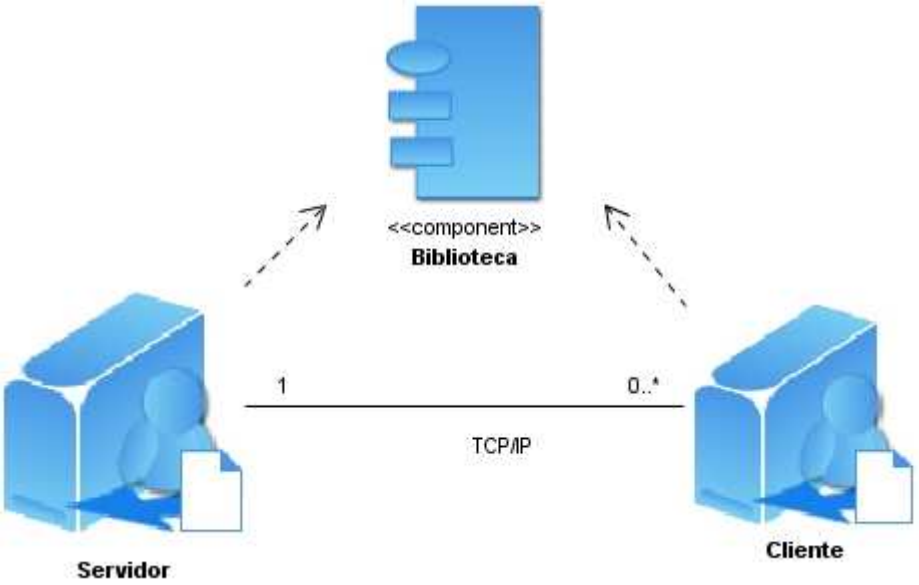


Figura 12 Diagrama de Despliegue.

Conclusiones del capítulo

En este capítulo se sentaron las bases técnicas sobre las cuales se va a implementar el módulo, que dará solución al objetivo propuesto. Se seleccionó la metodología y herramientas que se van a utilizar. Se definieron las funcionalidades del sistema a partir de la recogida de los requisitos funcionales, a los que el módulo debe responder para dar total cumplimiento a los objetivos propuestos. Se dividió la estructura del módulo en dos subsistemas funcionales, con el objetivo de encapsular las responsabilidades del cliente y el servidor, y de esta forma independizar su desarrollo.

Capítulo III

Implementación y Prueba del Sistema

3.1 Implementación

La metodología XP propone comenzar las implementación de la solución partiendo de una arquitectura lo más flexible posible, para evitar grandes cambios en las próximas iteraciones y en los cambios que generalmente el cliente propone. Es necesario, desde un inicio, tener bien definida la arquitectura del trabajo.

Trazada la misma se procede a la primera iteración.

3.1.1 1ra iteración

El objetivo de la primera iteración es desarrollar la historia de usuario Nº 1: Abrir conexión en el cliente, la historia de usuario Nº2: Abrir conexión en el servidor, y la historia de usuario Nº.3: Crear los datos a enviar. Esta iteración no finaliza con un producto visual capaz de ser probado, ya que su resultado es un módulo de red que servirá de base para las próximas iteraciones. Para ello se trazaron las siguientes tareas:

Historias de usuarios Nº 1 y Nº 2:

1. Tarea Nº 1: Implementar la clase: Client.
2. Tarea Nº 2: Implementar la clase: Server.

Historia de usuario Nº 3:

3. Tarea Nº 3: Implementar la clase: Element.

Tabla 11 Tarea Implementar la clase: Client.

Tarea	
Número de Tarea:1	**Numero de HU:1**
Nombre de la Tarea: Implementar la clase: Client.	
Tipo de Tarea: Desarrollo	**Estimación: 2 días**
Fecha de Inicio: 6 de Septiembre del 2010	**Fecha de Fin: 7 de Septiembre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: Se inicializa el cliente mediante del método run() y se actualiza el archivo config.client con las especificaciones de los puertos por donde se va a conectar.	

Tabla 12 Tarea Implementar la clase: Server.

Tarea	
Número de Tarea:2	**Numero de HU:2**
Nombre de la Tarea: Implementar la clase: Server.	
Tipo de Tarea: Desarrollo	**Estimación : 2 días**
Fecha de Inicio: 8 de Septiembre del 2010	**Fecha de Fin: 9 de Septiembre del 2010**
Programador Responsable: Frank del Rio Hernández	
Descripción: Se inicializa el servidor mediante del método run() y se actualiza el archivo config.server con las especificaciones de los puertos por donde se va a recibir las conexiones.	

Tabla 13 Tarea Implementar la clase: Element.

Tarea	
Número de Tarea: 3	**Numero de HU:3**
Nombre de la Tarea: Implementar la clase: Element.	
Tipo de Tarea: Desarrollo	**Estimación: 3 días**
Fecha de Inicio: 10 de Septiembre del 2010	**Fecha de Fin: 14 de Septiembre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: Se desarrollan los tipos de datos que va a brindar el módulo de red.	

3.1.2 2da iteración

El objetivo de esta iteración es desarrollar la historia de usuario Nº4: Aceptar la conexión de un nuevo cliente y la historia de usuario Nº5: Calcular latencia. En esta iteración se implementan todas las funcionalidades que permiten al servidor aceptar una nueva conexión, crear una sesión para cada nuevo cliente conectado, así como calcular la latencia de dichos clientes. Las tareas definidas para la iteración son las siguientes:

Historia de usuario Nº4:

1. Tarea Nº 1: Implementar la clase: Session.
2. Tarea Nº 2: Implementar la clase: SessionFactory.
3. Tarea Nº 3: Implementar la clase: ReapTask.

Historia de usuario Nº5:

4. Tarea Nº 4: Implementar la clase: Latency.

Tabla 14 Tarea Implementar la clase: Session.

Tarea	
Número de Tarea: 1	**Numero de HU: 4**
Nombre de la Tarea: Implementar la clase: Session	
Tipo de Tarea: Desarrollo	**Estimación: 4 días**
Fecha de Inicio: 15 de Septiembre del 2010	**Fecha de Fin: 20 de Septiembre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: Se crea un objeto del tipo session y un método que permita obtener el nombre de la sesión.	

Tabla 15 Tarea Implementar la clase: SessionFactory.

Tarea	
Número de Tarea: 2	**Numero de HU: 4**
Nombre de la Tarea: Implementar la clase: SessionFactory	
Tipo de Tarea: Desarrollo	**Estimación: 4 días**
Fecha de Inicio: 21 de Septiembre del 2010	**Fecha de Fin: 24 de Septiembre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: Se encarga de crear las sesiones con un identificador único y un nombre que permite identificar la conexión.	

Tabla 16 Tarea Implementar la clase: ReapTask.

Tarea	
Número de Tarea: 3	**Numero de HU: 4**
Nombre de la Tarea: Implementar la clase: ReapTask	
Tipo de Tarea: Desarrollo	**Estimación: 4 día**
Fecha de Inicio: 27 de Septiembre del 2010	**Fecha de Fin: 30 de Septiembre del 2010**
Programador Responsable: Frank del Rio Hernández	
Descripción: Su principal objetivo es mantener actualizada la lista de sesiones, o sea, añade las sesiones creadas por la SessionFactory o elimina las sesiones que no cumplan con los requisitos establecidos para mantener la conexión.	

Tabla 17 Tarea Implementar la clase: Latency.

Tarea	
Número de Tarea: 4	**Numero de HU: 5**
Nombre de la Tarea: Implementar la clase: Latency	
Tipo de Tarea: Desarrollo	**Estimación: 1 día**
Fecha de Inicio: 1 de Octubre del 2010	**Fecha de Fin: 1 de Octubre del 2010**
Programador Responsable: Frank del Rio Hernández	
Descripción: Utiliza la clase Ping de ICE para calcular la latencia de un cliente en un momento determinado.	

3.1.3 3ra iteración

El objetivo principal de esta iteración es desarrollar la historia de usuario Nº6: Desconectar un cliente y la historia de usuario Nº7: Terminar la conexión con el servidor. En esta iteración se implementan las funciones que controlan la desconexión de un cliente del servidor. Las tareas definidas para esta iteración son las siguientes:

Historia de usuario Nº6:

1. Tarea Nº1: Implementar la función refresh().
2. Tarea Nº2: Implementar la función destroy().

Historia de usuario Nº7:

3. Tarea Nº3: Implementar la función shutdown().

Tabla 18 Tarea Implementar la función: refresh.

Tarea	
Número de Tarea: 1	**Numero de HU: 6**
Nombre de la Tarea: Implementar la función refresh()	
Tipo de Tarea: Desarrollo	**Estimación: 2 día**
Fecha de Inicio: 4 de Octubre del 2010	**Fecha de Fin: 5 de Octubre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: El cliente debe refrescar la sesión mediante el envío de un timestamp.	
Clases: Session	

Tabla 19 Implementar la función: destroy.

Tarea	
Número de Tarea: 2	**Numero de HU: 6**
Nombre de la Tarea: Implementar la función destroy()	
Tipo de Tarea: Desarrollo	**Estimación: 1 día**
Fecha de Inicio: 6 de Octubre del 2010	**Fecha de Fin: 6 de Octubre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: Recibe como parámetro de entrada el id de la sesión.	
Clases: Session	

Tabla 20 Implementar la función: shutdown.

Tarea	
Número de Tarea: 3	**Numero de HU: 7**
Nombre de la Tarea: Implementar la función shutdown()	
Tipo de Tarea: Desarrollo	**Estimación: 1 día**
Fecha de Inicio: 7 de Octubre del 2010	**Fecha de Fin: 7 de Octubre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: El cliente envía al servidor la orden de cerrar la sesión para cerrar la conexión.	
Clases: Session	

3.1.4 4ta iteración

El objetivo principal de esta iteración es desarrollar la historia de usuario Nº8: Detectar desconexiones de los clientes y la historia de usuario Nº9: Detectar servidores. En esta iteración se implementan las funciones que permiten detectar servidores en una red de área local. Las tareas definidas para esta iteración son las siguientes:

Historia de usuario Nº9:

1. Tarea Nº1: Implementar la función runTimerTask().

Historia de usuario Nº10:

2. Tarea Nº2: Implementar la clase LanSearch.
3. Tarea Nº3: Implementar la clase ServerName.

Tabla 21 Implementar la función: runTimeTask.

Tarea	
Número de Tarea: 1	**Numero de HU: 8**
Nombre de la Tarea: Implementar la función runTimerTask()	
Tipo de Tarea: Desarrollo	**Estimación: 3 días**
Fecha de Inicio: 8 de Octubre del 2010	**Fecha de Fin: 12 de Octubre del 2010**
Programador Responsable: Frank del Rio Hernández	
Descripción: Se comprueba el timestamp de cada cliente para comprobar la vitalidad de la sesión.	
Clases: ReapTask	

Tabla 22 Implementar la clase: LanSearch.

Tarea	
Número de Tarea: 2	**Numero de HU: 9**
Nombre de la Tarea: Implementar la clase LanSearch	
Tipo de Tarea: Desarrollo	**Estimación: 5 días**
Fecha de Inicio: 13 de Octubre del 2010	**Fecha de Fin: 19 de Octubre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: El cliente envía un mensaje de difusión el la subred donde se encuentra para ver si hay algún servidor en línea.	

Tabla 23 Implementar la clase: ServerName.

Tarea	
Número de Tarea: 3	**Numero de HU: 9**
Nombre de la Tarea: Implementar la clase ServerName	
Tipo de Tarea: Desarrollo	**Estimación: 5 días**
Fecha de Inicio: 20 de Octubre del 2010	**Fecha de Fin: 24 de Octubre del 2010**
Programador Responsable: Reyneris Terrero Delfino	
Descripción: El servidor notifica que está en línea al cliente enviándole su nombre.	

3.2 Prueba

Las pruebas constituyen una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente" [11].

La metodología XP cuenta con una característica conocida como desarrollo dirigido por pruebas. El mismo se enfoca en la implementación orientada a pruebas. El código debe ser probado paso a paso y obtener un resultado, aunque aún no con lógica para el negocio, sí funcional.

Las pruebas de caja blanca son realizadas a los métodos u operaciones para medir la funcionalidad del mismo desde la perspectiva de la validez para el cliente. El desarrollo dirigido por pruebas, por otro lado, se aplica antes de comenzar a implementar cada paso de la tarea en desarrollo asumiendo que la prueba es insatisfactoria desde un inicio. Solo, una vez que se haya cumplido de la forma más sencilla posible, la lógica del código a probar, se asume como cumplida. Luego se realiza un proceso conocido informalmente como "refactorización" de código, el cual consiste en limpiarlo, organizarlo y adaptarlo a los patrones. En esencia, el desarrollo dirigido por pruebas se centra en la lógica del código y las pruebas de caja blanca en la del negocio. Las pruebas en el desarrollo dirigido por pruebas también son conocidas como pruebas unitarias o de unidad.

Una vez terminada cada tarea, se prueba la funcionalidad de la misma, ya desde el punto de la validez de esta para el cliente. Esta prueba es realizada por parte del desarrollador. Luego se realizan las pruebas conocidas como de "caja gris", en la cual se valida el comportamiento general de la solución en ambientes adversos pocos recursos de hardware y sobrecarga del mismo. Finalmente el usuario final, realiza las de "caja negra" o aceptación.

3.2.1 Pruebas de aceptación

Las pruebas de aceptación también son conocidas como pruebas de caja negra, ya que es el propio cliente quien la realiza en compañía de uno de los representantes del equipo de desarrollo y se orientan a las funcionalidades del sistema. Su objetivo es comprobar, desde la perspectiva del usuario final, el cumplimiento de las especificaciones de la lista de reservas del producto. A continuación, se muestran las pruebas de aceptación realizadas a la solución propuesta**:**

Tabla 24 Prueba de Aceptación: Abrir conexión en el cliente.

Caso de Prueba de Aceptación	
Código:HU1-CPA1	**Historia de Usuario:1**
Nombre: Abrir conexión en el cliente.	
Descripción:	
Condiciones de Ejecución: El usuario previamente debe haber creado un objeto de tipo Configuration.	
Entrada/ Pasos de ejecución: El usuario debe inicializar los parámetros para la conexión en un fichero que será cargado por el método Load().Se debe crear un objeto de tipo Cliente e inicializarlo con el objeto de tipo Configuration. Se inicia la comunicación al ejecutar el método Start() de la clase Cliente.	
Resultado Esperado: El cliente establece comunicación.	
Evaluación de la Prueba: prueba satisfactoria.	

Tabla 25 Prueba de Aceptación: Abrir conexión en el servidor.

Caso de Prueba de Aceptación	
Código:HU2-CPA1	**Historia de Usuario:2**
Nombre: Abrir conexión en el servidor	
Descripción:	
Condiciones de Ejecución: El usuario debe crear un objeto de tipo Configuration.	

Entrada/ Pasos de ejecución: El usuario debe inicializar los parámetros para la conexión en un fichero que será cargado por el método Load(). Se debe crear un objeto de tipo Server e inicializarlo con el objeto de tipo Configuration. Se tiene todo listo para iniciar la comunicación al ejecutar el método Start() de la clase Server.

Resultado Esperado: El servidor debe estar listo para aceptar las conexiones de los clientes.

Evaluación de la Prueba: prueba satisfactoria.

Tabla 26 Prueba de Aceptación: Aceptar la conexión de un nuevo cliente.

Caso de Prueba de Aceptación

Código:HU4-CPA1 **Historia de Usuario:4**

Nombre: Aceptar la conexión de un nuevo cliente.

Descripción:

Condiciones de Ejecución: El servidor debe estar en ejecución, debe existir una correcta configuración de los parámetros de conexión.

Entrada/ Pasos de ejecución: Cuando el cliente se va a conectar el servidor verifica que si el número de cliente conectado es menor que el número máximo de conexiones permitada, tambien verifica si la latencia del cliente esta por debajo de la latencia máxima permitida. Después de haber de verificado esto acepta la conexión**.**

Resultado Esperado: El servidor acepta la conexión de un nuevo cliente.

Evaluación de la Prueba: prueba satisfactoria.

Tabla 27 Prueba de Aceptación: Calcular latencia.

Caso de Prueba de Aceptación

Código:HU5-CPA1	**Historia de Usuario:5**
Nombre: Calcular latencia.	
Descripción:	
Condiciones de Ejecución: El cliente debe estar en ejecución, debe existir una correcta configuración de los parámetros de conexión.	
Entrada/ Pasos de ejecución: El servidor cada un tiempo determinado le pide al cliente el valor de la latencia existente en la conexión y con el resultado de la misma el servidor acepta o rechaza la conexión con el cliente.	
Resultado Esperado: Obtener una latencia inferior a los 500ms	
Evaluación de la Prueba: prueba satisfactoria.	

CONCLUSIONES

Al terminar el presente trabajo de diploma se logró una comunicación bidireccional en una arquitectura cliente/servidor, a través del envío y recepción de datos entre varios clientes desde diferentes estaciones de trabajo, utilizando una aplicación de Ogre, mediante el módulo de red GenericMiddleware, el cual reduce el tiempo y el esfuerzo de desarrollo de los componentes de comunicación.

RECOMENDACIONES

Se recomienda el estudio detallado de la biblioteca GenericMiddleware para realizar mejoras en cuanto a:

- Reducir el tiempo que demora enviar una cantidad de objetos dados.
- Incluir el servicio de cortafuegos para lograr eficiencia en redes seguras.
- Incluir el servicio activación y localización de aplicaciones.

BIBLIOGRAFÍA

Duch i Gavaldà, J., & Tejedor Navarro, H. 2008. *Sonido, interacción y redes.* Barcelona.

McMahon, R. A. 2006. *Introducción a las redes.* Huston.

Stallings, W. 2000. *Comunicaciones y Redes de Computadores.* Madrid: Editorial Prentice Hall.

Poratti, Gustavo Gabriel. 2005. *Redes. La Guía de Referencia Actual y Definitiva.* Buenos Aires, Argentina : MP Ediciones SA.

Henning, Michi y Spruiell, Mark. Febrero 2010. *Distributed Programming with Ice.* EE.UU.

Vallejo Fernández, David. 2006. *Documentación de ZeroC ICE.*

REFERENCIA BIBLIOGRÁFICA

[1] **2004.** Universitat Jaume -I. [En línea] 25 de Noviembre de 2004. [Citado el: 5 de abril de 2010.] http://www4.uji.es/~al019803/tcpip/paginas/introduccion.htm

[2] **Smed, Jouni y Hakonen, Harri. 2006.** *Algorithms and Networking for Computer Games.* Finlandia : University of Turku, 2006.

[3] **2004.** Universitat Jaume -I. [En línea] 25 de Noviembre de 2004. [Citado el: 5 de abril de 2010.] http://www4.uji.es/~al019803/tcpip/paginas/capas.htm

[4] **Rodriguez Noa, Carlos y Gómez Finalé, Yordanis**. Arquitectura de red para la confección de videojuegos multijugador. Tesis UCI, Facultad 5, 2008.

[5] **2008.** Jenkins Software. *Raknet.* [En línea] 2008. [Citado el: 12 de Abril de 2010.] http://www.jenkinssoftware.com/raknet/manual/index.html

[6] **Frisbie, Phil. 2003.** Hawksoft. *Hawk Software.* [En línea] 30 de Diciembre de 2003. [Citado el: 15 de Enero de 2010.] http://hawksoft.com/hawknl/

[7] **Jörg Rüppel. 2006.** Zoidcom. *Zoidcom Automated Networking System.* [En línea] 2006. [Citado el: 21 de Marzo de 2010.] http://www.zoidcom.com/

[8] **González Nava, Sergio, Cheang León, Guillermo y Kashiwamoto Yabuta, Jorge. 2004.** *Introducción a los Sistemas Distribuidos.* 2004.

[9] **Vallejo Fernández, David. 2006.** *Documentación de ZeroC ICE.* 2006.

[10] **Helm, Richard, Johnson, Ralph, Gamma, Erich & Vlissides, John.** *Design Patterns. Addison Wesley Longman Inc.* 1994.

[11] **Colectivo de autores de la UCI.** *Disciplina Prueba,* 2007. [Disponible en: http://eva.uci.cu/mod/resource/view.php?id=14103]

GLOSARIO DE TÉRMINOS

A

API: es una interfaz de programación de aplicaciones, conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

D

DirectX: es una colección de APIs creadas y recreadas para facilitar las complejas tareas relacionadas con multimedia, especialmente programación de videojuegos y vídeo en la plataforma Microsoft Windows.

I

IDE: Entorno integrado de desarrollo. Es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un lenguaje de programación o bien, poder utilizarse para varios.

IBM: Máquinas de Negocios Internacional. Es la empresa que consiguió hacer de su modelo de PC un estándar.

L

Latencia: Lapso de tiempo necesario para que un paquete de información viaje desde la fuente hasta su destino.

M

Multiplexar: Es la combinación de dos o más canales de información en un solo medio de transmisión usando un dispositivo llamado multiplexor. El proceso inverso se conoce como desmultiplexación.

Motor de videojuego: es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un videojuego.

O

OMG: Grupo de Gestión de Objetos. Es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA.

OpenGL: es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

P

POP: Protocolo de Oficina de Correo. Este protocolo no necesita una conexión permanente a internet puesto que es en el momento de la conexión cuando solicita al servidor el envío de la correspondencia almacenada en el servidor para dicho usuario.

S

SMTP: Protocolo simple de transferencia de correo, es el protocolo estándar que permite la transferencia de correo de un servidor a otro mediante una conexión punto a punto.

SSL: Capa de Seguridad de Socket. Es un protocolo desarrollado por Netscape Communications Corporation para dar seguridad a la transmisión de datos en transacciones comerciales en Internet. Utilizando la criptografía de llave pública, SSL provee autentificación del servidor, encriptación de datos, e integridad de los datos en las comunicaciones cliente/servidor.

Serialización (o marshalling en inglés): Consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML

U

UML: Lenguaje Unificado de Modelado. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

W

WAN: Redes de área amplia son redes informáticas que se extienden sobre un área geográfica extensa. Contiene una colección de máquinas (hosts) dedicadas a ejecutar los programas de usuarios.