Facultad Regional “Mártires de Artemisa”

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: Integración del servidor de aplicaciones GlassFish al marco de trabajo jWebSocket.

Autor: Victor Antonio Barzana Crespo.

Tutor: MSc. Yamila Vigil Regalado.

Cotutor: Lic. Gilberto Ramón Justiniani Fernández.

Artemisa, Junio 2012

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad Regional de la Universidad de las Ciencias Informáticas, “Mártires de Artemisa”, así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo. Para que así conste firmo la presente a los ____ días del mes de _____del año _____.

____

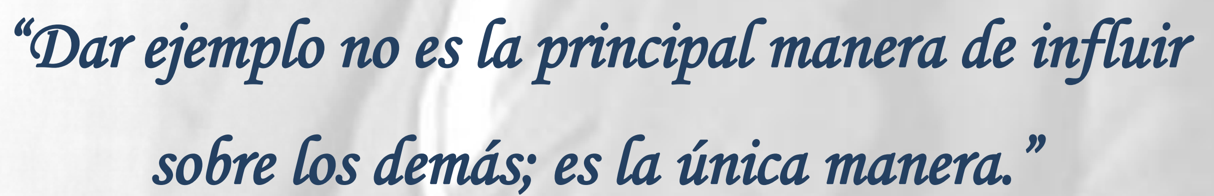
Victor Antonio Barzana Crespo

Firma del autor

______________________ ________________________________

Msc. Yamila Vigil Regalado Lic. Gilberto Ramón Justiniani Fernández

Firma de los tutores



"Dar ejemplo no es la principal manera de influir sobre los demás; es la única manera."

Albert Einstein

AGRADECIMIENTOS

Agradezco a todos los que de una forma u otra colaboraron con esta maravillosa tarea.

A mis padres, María Luisa y José Antonio por ser los mejores padres del mundo y darme siempre su apoyo incondicional.

A mi tía Migdalia por ser después de mi madre, lo más lindo que tengo en la vida.

A mis primos Carlos, Jorge y Alberto, al viejo Juanito y a mi tía Cira, gracias por ser tan bella familia y darme tanta fuerza y confianza para seguir adelante.

A mi tía Nohemí Barzana por ser la persona más bella del universo y por quererme tanto y demostrarme tanto afecto.

A mis tutores por toda su ayuda y orientación durante la realización de esta investigación.

A todos mis amigos de la escuela, en especial a Marcos, Merly, Marta, Daimí, Viquillón, Noel, Alexander y Carlos Karen, a todos gracias por haberme sido tan fieles y confiar en mí, gracias por estar ahí cuando los he necesitado.

A mis amigos Rolando Santamaría y Osvaldo Aguilar, gracias a ustedes me he convertido en el profesional que soy hoy, después que los conocí se despertó en mi un deseo insaciable de estudiar, leer y aprender nuevas tecnologías para alcanzarlos, aunque hoy no los he alcanzado me siento orgulloso de poder trabajar a la par de ustedes.

A Alexander Schulze, líder del proyecto jWebSocket y a Rebecca Schulze por darme su apoyo en todo momento, por confiar en mí y permitirme formar parte de este maravilloso equipo en esta nueva era informática, la era de los WebSockets.

A todo el equipo de jWebSocket de la Facultad Regional Mártires de Artemisa, que de una forma u otra han aportado un granito de arena en la creación de este trabajo.

A los maravillosos profesores de nuestra facultad que han sido nuestros padres durante todo el tiempo en la escuela, en especial a los profesores Mario, Margarita y Crespo, que siempre me han dado los mejores consejos para ser un buen profesional en la vida.

DEDICATORIA

A toda mi familia y amigos, en especial a mi madre, por ser mi única razón de ser. A mis abuelos, quienes me dieron toda la preparación y educación para enfrentar la vida, en especial a mi abuelo Manolo que fue quien me crió, gracias por estar a mi lado cuando más te necesité. A mis tíos Jorge Luis, José Caridad y Luis por ser mis principales guías, padres, consejeros y precursores de mi carrera como futuro profesional. Este título es de ustedes por todo el sacrificio que a mi lado han hecho a lo largo de este camino que no ha sido corto ni fácil.

Al hombre más grande que he conocido en el mundo, Fidel Castro y a la obra que ha creado, La Revolución Cubana.

RESUMEN

Cada día surgen nuevas tecnologías que potencian el desarrollo de aplicaciones web que requieren una mayor velocidad de comunicación, sin embargo, la mayoría de ellas están creadas en gran medida usando el protocolo *HTTP*[1]. Para lograr una mayor interactividad en el modelo cliente/servidor se creó un protocolo para la Web denominado WebSocket. Hoy la mayoría de los servidores de aplicaciones y contenedores de servlet brindan soporte para este protocolo. El marco de trabajo jWebSocket se especializa en la creación de aplicaciones web basadas en este protocolo, este cuenta con una serie de herramientas y componentes que le facilitan el trabajo con esta tecnología a los desarrolladores. También posee una serie de motores para la comunicación basados en implementaciones WebSocket existentes.

Uno de los servidores de aplicaciones más importantes que ofrecen soporte para el protocolo WebSocket es GlassFish. Este utiliza el marco de trabajo Grizzly para la comunicación a bajo nivel aprovechando todas las ventajas de la API de Java NIO. Debido a que jWebsocket hoy no cuenta con una integración con este servidor, implica que las aplicaciones que se realicen con jWebsocket enfrenten un grupo de limitaciones. Al ser desplegadas es necesario habilitar puertos no estándares para la comunicación con el servidor. Esto conlleva a bajos niveles de seguridad en los routers y servidores proxy. Además, jWebSocket no permite la creación de aplicaciones web basadas en servlets y no es compatible con soluciones existentes desarrolladas, lo que trae consigo que las aplicaciones sean difíciles de migrar a esta tecnología.

En consecuencia se identifica la necesidad de desarrollar un motor para la integración del servidor de aplicaciones GlassFish para el jWebSocket que solucione todos los problemas mencionados anteriormente. De esta manera se espera mejorar los niveles de seguridad y usabilidad de las aplicaciones que hoy se realizan con jWebSocket.

ÍNDICE

INTRODUCCIÓN......1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA......8
1.1 Fundamentos Teóricos de la Investigación......8
1.2 Análisis de soluciones existentes......11
1.3 Metodología usada para el desarrollo de la solución......13
1.4 Herramientas y Tecnologías usadas para el desarrollo de la solución......16
1.4.1 Lenguajes de programación y modelado......16
1.4.2 Otros Lenguajes......18
1.4.3 Marcos de Trabajo del lado del Servidor......19
1.4.4 Servidores Web......22
1.4.5 Herramientas para la construcción de proyectos......23
1.4.6 Herramientas CASE......24
1.4.7 Herramientas de Control de Versiones......25
1.4.8 Clientes de Control de Versiones......26
1.4.9 Entorno Integrado de Desarrollo – IDE......27
CAPÍTULO 2: CARACTERÍSTICAS, ANÁLISIS Y DISEÑO DEL SISTEMA......29
2.1. Propuesta de Solución......29
2.3. Planificación del proyecto por Roles.......30
2.4. Modelo de Dominio......32
2.5. Lista de Reserva del Producto (LRP).......33
2.6. Historias de Usuario y Tareas de Ingeniería.......36
2.7. Plan de Releases......47
2.8. Descripción de la Arquitectura......48
2.9. Diseño con Metáforas......49
2.10. Diagrama de Componentes......50
CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DEL SISTEMA......53
3.1. Diagrama de Despliegue......53
3.2. Estrategia de Validación......54
3.3. Resultados Obtenidos......63
3.4. Funcionalidades Obtenidas......64
3.5. Aporte Social y Económico......64
CONCLUSIONES......66
RECOMENDACIONES......67
BIBLIOGRAFÍA......68

ANEXOS 71
Anexo 1:...71
GLOSARIO DE TÉRMINOS 72

INTRODUCCIÓN

Durante los últimos 10 años, la Web ha alcanzado un gran auge, las aplicaciones que hoy se desarrollan requieren una gran interactividad para poder simular los procesos de la vida real. Cada día surgen nuevas tecnologías que potencian el desarrollo de este tipo de aplicaciones web, sin embargo, la mayoría de ellas están creadas en gran medida usando el protocolo *HTTP*. Este protocolo está basado en un modelo de solicitud y respuesta, lo que trae consigo una serie de desventajas para los desarrolladores web. Las aplicaciones que lo utilizan para la transferencia de datos se ven afectadas en cuanto a la velocidad y el consumo de ancho de banda en la red.

Con el objetivo de aumentar el rendimiento y la capacidad de respuesta de las aplicaciones web basadas en el protocolo HTTP se han creado múltiples tecnologías orientadas al mismo. Estas son conocidas como tecnologías *XHR*[2], entre ellas se pueden mencionar *Ajax*[3] y *Comet*[4], han sido un grupo de herramientas viables para cubrir parcialmente estas necesidades, la mayoría de ellas basadas en el lenguaje *JavaScript*[5] del lado del cliente. Sin embargo estas tecnologías aportaron algunos elementos en el logro del tiempo real en la Web pero con muy altos costos en el uso de ancho de banda y una mayor dependencia de recursos de red. Debido a todas estas limitaciones surge el protocolo *WebSocket*[6], el cual consiste en una comunicación bidireccional y *full-duplex*[7] en tiempo real sobre un único socket *TCP*[8].

WebSocket establece una forma estandarizada para el intercambio de paquetes y la realización del conocido *handshake*[9] para aplicaciones web estacionarias y móviles. La comunicación basada en WebSocket garantiza menos sobrecarga en la red en el orden de 400 veces y solamente 1/3 de la latencia en la red. (jWebSocket for Android - Real-time Communication for Mobile Devices, 2011)

Desde la creación del protocolo WebSocket múltiples desarrolladores compiten por brindar la mejor solución para crear aplicaciones basadas en esta tecnología. El marco de trabajo jWebSocket es una de las tecnologías orientadas al desarrollo de este tipo de aplicaciones. Este marco de trabajo permite crear innovadoras aplicaciones HTML5 que gocen de altos niveles de velocidad, escalabilidad y

seguridad. jWebSocket es una solución de código abierto basada en Java y JavaScript con una amplia gama de funcionalidades orientadas a la interacción entre los clientes y el servidor, proporcionando altos niveles de abstracción. El marco de trabajo posee un adaptador basado en *Flash*[10] para navegadores que no sean compatibles con HTML5, sin embargo se requiere tener instalado en el navegador un plug-in para Flash.

Hoy día las aplicaciones web están basadas en una arquitectura cliente/servidor. Del lado del servidor es muy importante gestionar la mayor parte o la totalidad de las funciones de lógica de negocio y de acceso a datos, la centralización de la información y la disminución de la complejidad en el desarrollo de aplicaciones. Es por esto que se creó el concepto de servidor de aplicaciones, el cual permite ejecutar ciertas aplicaciones y a la vez sirve de contenedor para los componentes que las forman. Estos últimos componentes están escritos usualmente en el lenguaje Java y son conocidos como *Servlets*[11], *Java Server Pages (JSP)*[12], *Enterprise Java Beans (EJB)*[13], entre otros. Los servidores de aplicaciones desarrollados en Java utilizan un perfil ligero para crear las aplicaciones web, siguiendo un estándar que se ajusta a la nueva generación y permitiendo el uso de todas las ventajas de esta plataforma para aplicaciones empresariales. Existe una gran diversidad de servidores de aplicaciones creados sobre el lenguaje Java que implementan las tecnologías definidas en la plataforma Java EE, entre ellos se pueden mencionar, JBoss AS, Geronimo, ColdFusion, *WebSphere*[14], *Oracle Application Server*[15] y *GlassFish*[16].

Cada servidor de aplicaciones permite controlar las conexiones y la transferencia de datos a bajo nivel entre múltiples clientes y el servidor con el uso de motores. Cada motor implementa una *API*[17] específica por cada protocolo de transferencia, *Netty*[18], *Jetty*[19], por mencionar algunos, son marcos de trabajo que utilizan el modelo cliente/servidor. Estos están basados en el motor *Java New I/O API (NIO)*[20] el cual permite el manejo de múltiples protocolos como FTP, HTTP, SMTP. Otras implementaciones de motores son Socket.IO, Java IO, NodeJS.

La mayoría de los servidores de aplicaciones brindan su propia implementación para soportar el protocolo WebSocket, gracias a esto se pueden aprovechar dentro de las aplicaciones creadas para este protocolo, todas las ventajas y tecnologías

que se incluyen dentro de un servidor de aplicaciones. Estos servidores pueden a su vez ser integrados con otros servidores de aplicaciones o contenedores de servlet que soporten WebSocket para reutilizar características específicas de ambos. El marco de trabajo jWebSocket actualmente incluye en su núcleo las implementaciones WebSocket de otros servidores como Jetty, Netty, TCP, MINA, entre otros. Este aprovecha la mayoría de sus potencialidades para obtener más estandarización, soporte y comunidad de desarrollo. Además, con esta integración también estos servidores aprovechan todas las ventajas de jWebSocket para crear aplicaciones basadas en WebSocket.

Uno de los servidores de aplicaciones de código abierto más importantes en el desarrollo de aplicaciones web sobre Java es GlassFish. El mismo implementa tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Posee ventajas que mejoran significativamente la productividad del desarrollador, como una gran comunidad de desarrollo, presenta un perfil web ligero para desarrollar aplicaciones web incluyendo las últimas versiones de tecnologías para el desarrollo.

GlassFish utiliza el marco de trabajo Grizzly para proveer a los desarrolladores un ambiente mediante el cual puedan crear aplicaciones que aprovechen todas las ventajas de la API de Java NIO. También ofrece una gran cantidad de componentes como Web Framework (Servidor Web basado en HTTP/HTTPS), Servlet, Grizzly-WebSockets (Servidor basado en WebSocket).

Teniendo en cuenta los elementos anteriores se plantea la siguiente **situación problemática:**

Debido a que jWebsocket hoy no cuenta con una integración al servidor de aplicaciones GlassFish implica que las aplicaciones que se realicen con jWebsocket enfrenten un grupo de limitaciones. Al ser desplegadas las aplicaciones utilizando jWebsocket es necesario habilitar puertos no estándares para la comunicación con el servidor. Esto conlleva a bajos niveles de **seguridad** en los routers y servidores proxy donde se despliegan las aplicaciones creadas con jWebSocket. Además, usando el marco de trabajo no es posible el uso compartido de los puertos 80 y 443 para servir páginas web y soportar conexiones WebSocket

al mismo tiempo.

Otra de las consecuencias de no contar con la integración de GlassFish es no poder crear aplicaciones web basadas en servlets compartidos con jWebSocket. Esto implica que en un futuro el marco de trabajo no será capaz de adaptarse a nuevas especificaciones de servlet. Otra limitación es que jWebSocket no es compatible con soluciones existentes desarrolladas utilizando el servidor GlassFish y las aplicaciones sean difíciles de migrar a esta tecnología. Esto trae consigo que se afecte la **usabilidad** de jWebSocket para desarrollar y desplegar aplicaciones creadas con este marco de trabajo.

De lo planteado anteriormente surge la siguiente interrogante que nos define el **problema científico:**

¿Cómo garantizar la integración del servidor GlassFish en el marco de trabajo jWebSocket para lograr mayores niveles de seguridad y usabilidad en las aplicaciones desarrolladas con esta tecnología?

Por tanto el presente trabajo centra su **objeto de estudio** en el proceso de integración de servidores de aplicaciones Java.

Como **campo de acción** se tiene el proceso de integración de servidores de aplicaciones Java en el marco de trabajo jWebSocket.

Para dar solución al problema anterior se plantea como **objetivo general** Desarrollar un motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket que garantice mayores niveles de seguridad y usabilidad en las aplicaciones desarrolladas con esta tecnología.

Para cumplir con el objetivo general planteado se han derivado las siguientes **preguntas científicas:**

- ✓ ¿Cuáles son los fundamentos teórico-metodológicos del proceso de integración de servidores de aplicaciones Java?
- ✓ ¿Cuál es la situación actual del proceso de integración de servidores de aplicaciones Java que brinden soporte al protocolo WebSocket?

- ✓ ¿Cómo desarrollar un motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket que garantice mayores niveles de seguridad y usabilidad en las aplicaciones desarrolladas con esta tecnología?
- ✓ ¿Cómo validar la capacidad del motor desarrollado para garantizar mayores niveles de seguridad y usabilidad en las aplicaciones desarrolladas con el marco de trabajo jWebSocket?

Las **tareas de investigación** desarrolladas para cumplir los objetivos propuestos son:

- Sistematización de los fundamentos teóricos de la investigación.
- Definición de la metodología, herramientas, tecnologías a utilizar para el desarrollo de la solución propuesta.
- Análisis, diseño e implementación de un motor de GlassFish para jWebSocket utilizando la librería Grizzly-WebSockets.
- Comprobar la capacidad y potencialidad del motor de GlassFish para jWebSocket mediante pruebas Funcionales y otros métodos asociados a las Ciencias Informáticas.

Los **métodos científicos** utilizados en esta investigación fueron:

Teóricos:

- **Histórico-Lógico:** Permite analizar la trayectoria completa acerca de los entornos de integración continua para aplicaciones Java, así como el estudio histórico de los mismos que permite ver deficiencias y proponer soluciones acorde a las necesidades.
- **Análisis Documental** permite realizar el estudio de una variada documentación referente al motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket y las herramientas utilizadas actualmente para lograrlo, con el objetivo de obtener a través de estas el análisis, las experiencias y las sugerencias que pudieran ser incorporadas a la investigación.
- **Analítico-Sintético:** Mediante este método se va a analizar toda la

teoría recopilada a través de los diferentes medios bibliográficos que pueda servir para configurar un mejor entorno de integración continua, y poder aplicar así estos conocimientos en la práctica de manera que se adquiera una mayor preparación sobre el tema en cuestión.

- **Modelación:** Este método permite realizar una representación de la situación que se analiza. Permite obtener mediante diagramas y objetos una mayor compresión del problema y ayudar a configurar un entorno de integración continua para las aplicaciones java partir de la situación problemática.

De la presente investigación se derivan las siguientes **variables**:

- **Independiente:** Motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket.
- **Dependiente:**
  - Nivel de seguridad de las aplicaciones desarrolladas con el marco de trabajo jWebSocket.
  - Nivel de usabilidad de las aplicaciones desarrolladas con el marco de trabajo jWebSocket.

Posibles Resultados

Motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket que garantice mayores niveles de seguridad y usabilidad en las aplicaciones desarrolladas con esta tecnología.

Para una mejor comprensión de la investigación, el contenido ha sido separado en tres capítulos, aparte de las conclusiones generales, recomendaciones, referencias bibliográficas y bibliografía utilizada, glosario de términos en el cual se detallan los términos técnicos y poco claros utilizados en la elaboración del documento, y los anexos que complementan el trabajo realizado. Los capítulos han sido estructurados de la siguiente manera:

Capítulo 1. Fundamentación Teórica:

Se realiza la fundamentación teórica de la investigación. Se expone un estudio del estado del arte del proceso de integración de servidores de aplicaciones Java en la

actualidad.

Capítulo 2. Características, Análisis y Diseño del Sistema:
Brinda una fundamentación de la solución propuesta, a partir de la cual se describen las actividades de análisis de la solución, seguidas por la descripción de los procesos del sistema y de la etapa de diseño.

Capítulo 3. Implementación y Validación del Sistema:
Se describe la etapa de implementación que conlleva a la obtención del software. Se elaboran y documentan las pruebas realizadas a la solución propuesta para demostrar el correcto funcionamiento de los requerimientos de la misma. Se realiza un análisis de los resultados de la aplicación en un entorno real, comparando indicadores antes y luego de la solución.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

El presente capítulo tiene como objetivo tratar los principales conceptos y aspectos más significativos relacionados con la temática abordada desde distintos enfoques. Se realiza un estudio acerca del estado del arte del proceso de integración de servidores de aplicaciones Java y en específico de la integración de servidores de aplicaciones Java en el marco de trabajo jWebSocket.

Se analizan además las diferentes metodologías de desarrollo de software, lenguajes de programación, tecnologías y herramientas que serán usados para el desarrollo de la solución, siempre teniendo en cuenta la necesidad del uso de aplicaciones de código abierto.

1.1 Fundamentos Teóricos de la Investigación

Los conceptos fundamentales que orientaron la presente investigación estuvieron referidos a los servidores de aplicaciones, al protocolo de comunicación WebSocket y a los motores WebSocket. Se describen además los fundamentos metodológicos del proceso de implementación basado en WebSocket del lado del servidor y la integración de servidores de aplicaciones con jWebSocket (Motores WebSocket).

La definición de servidor de aplicaciones ha sido dada por varios autores. Entre ellos se encuentra el enfoque brindado por Josef Templ, para el cual un servidor de aplicaciones representa un marco de trabajo para aplicaciones servidores, que ejecuta programas que proveen una API y permiten acceso remoto a todas sus funcionalidades (Templ, 2003)

Otra definición fue dada por José Carlos Carvajal Riola, para el cual los servidores Java EE son llamados servidores de aplicaciones, porque permiten obtener datos de las aplicaciones cliente, del mismo modo que un servidor web sirve páginas web a un navegador. Un servidor Java EE contiene diferentes tipos de componentes, los cuales se corresponden con cada una de las capas de una aplicación multicapa. (Riola, 2008)

Los conceptos anteriormente mencionados presentan elementos importantes en la definición de servidores de aplicaciones, sin embargo, en ellos no se mencionan

algunos aspectos importantes. Por ello, para esta investigación se elabora una definición de servidores de aplicaciones que toma como referencia los conceptos anteriores e incorpora los elementos ausentes en ellos.

Se plantea entonces que un servidor de aplicaciones es un dispositivo de software que proporciona servicios de aplicación a los equipos o dispositivos cliente. Generalmente gestiona toda o la mayor parte de la lógica del negocio y el acceso a datos de las aplicaciones del lado del servidor. Le garantiza al usuario una centralización de tecnologías y una disminución de la complejidad en el desarrollo de aplicaciones. El uso de un Servidor de Aplicaciones brinda a los desarrolladores una Interfaz para Programación de Aplicaciones (API), garantizando que sus aplicaciones sean multiplataforma y fáciles de extender.

En los últimos años se han hecho muchos intentos para lograr tiempo real en la Web, múltiples herramientas y tecnologías han sido combinadas, pero siempre se obtenía pérdida en rendimiento o en velocidad debido a que estas tecnologías estaban basadas en el protocolo HTTP. Es por esto que fue necesario crear un nuevo protocolo para la Web, el protocolo WebSocket.

La definición del protocolo WebSocket ha sido dada por múltiples autores, tal es el caso de la definición dada en el sitio oficial de la W3C donde se define WebSocket como un protocolo que permite realizar conexiones bidireccionales entre un cliente y un servidor. Consiste en un mecanismo de handshake seguido de intercambios de mensajes sobre la capa TCP. El objetivo de esta tecnología es proveer un mecanismo para aplicaciones basadas en navegadores de internet que necesitan comunicación bidireccional con el servidor en vez de tener que realizar múltiples conexiones HTTP. (The WebSocket protocol, 2011)

Otra definición de WebSocket es la dada por Alexander Schulze en el año 2010 donde define esta tecnología como el estándar para la Web que permite que se puedan establecer conexiones en tiempo real entre un cliente y un servidor. El uso de este protocolo en lugar de usar HTTP implica múltiples ventajas, produce un consumo de ancho de banda de 1/50 y reduce la latencia en el orden de 1/3 comparado con el consumo del protocolo HTTP. (jWebSocket instead of XHR and Comet, 2010)

Para esta investigación se asume como concepto del protocolo WebSocket el concepto dado por Alexander Schulze en su artículo jWebSocket instead of XHR and Comet en el año 2010.

Para esta investigación se realizó un estudio de los fundamentos metodológicos para el desarrollo de implementaciones WebSocket del lado del servidor. Que no son más que un grupo de librerías que de una forma u otra sean capaces de gestionar el intercambio de datos entre el cliente y el servidor utilizando el protocolo WebSocket. Las implementaciones del lado del servidor WebSocket existentes se pueden encontrar dentro de las siguientes categorías, Librerías WebSocket, Contenedor de Servlet y Servidor de Aplicaciones.

jWebSocket es una tecnología orientada al desarrollo de aplicaciones basadas en el protocolo WebSocket que gocen de altos niveles de velocidad, escalabilidad y seguridad. Consiste en un marco de trabajo desarrollado sobre el lenguaje de programación Java, está compuesto por motores para el manejo de las conexiones mediante el protocolo WebSocket. En la presente investigación se analizaron las principales pautas y acciones que caracterizan al proceso de implementación de un motor para jWebSocket. Es por esto que se define como motor WebSocket para esta investigación, una interfaz que garantiza la comunicación entre cualquier implementación WebSocket del lado del servidor y el marco de trabajo jWebSocket. Los motores manejan las conexiones físicas de los clientes a través de los llamados Conectores que representan una abstracción de alto nivel de la conexión con los clientes. Un motor puede ser utilizado en cualquier momento sin necesidad de realizar ningún cambio sobre la aplicación ya que comparten una interfaz común. Los motores son configurables y sólo puede usarse uno a la vez, aunque pueden existir múltiples instancias del servidor jWebSocket ejecutándose por diferentes puertos. Cada uno de estos motores constituye una integración de jWebSocket con otro sistema embebido, el concepto Integración para jWebSocket se refiere a la posibilidad de reutilizar un Servidor de Aplicaciones o un Servidor Web existente mediante una interfaz denominada Motor WebSocket. (Schulze, 2012)

1.2 Análisis de soluciones existentes

Han sido muchos los intentos de oficializar el protocolo WebSocket. Decenas de borradores para la especificación de este protocolo se han sometido al análisis por la comunidad, para su aprobación y sugerencias. En la actualidad ya se alcanzó la versión final del protocolo y finalmente los desarrolladores pueden comenzar a desarrollar sus tecnologías sobre una versión estable del mismo.

Por su parte los servidores web y servidores de aplicaciones de la plataforma Java como Jetty, JBoss Netty, WebSphere, Oracle Application Server, GlassFish y otros, comienzan a ofrecer soporte al protocolo WebSocket. Estos servidores son aceptados por las empresas que desarrollan para la plataforma Java y tienen un alto respaldo por la comunidad. Ya a finales del año 2010 el término WebSocket comienza a ser altamente difundido en la comunidad Java. Esta revolución tecnológica favorece y ratifica a la plataforma Java convirtiéndola en la más preparada para soportar el desarrollo de aplicaciones basadas en WebSocket.

El marco de trabajo jWebSocket, en aras de aprovechar los beneficios de estos servidores, los integra como núcleo y reutiliza de ellos la gestión del protocolo, manejo de las conexiones, la transmisión de mensajes y el consumo de memoria, entre otras funcionalidades. Reutilizar dichas implementaciones, le permite al marco de trabajo jWebSocket ratificarse como un servidor extensible, altamente configurable y robusto.

Un requerimiento fundamental para una aplicación web es poder ser expandida fácil y aceleradamente usando WebSocket. Añadir soporte WebSocket para cualquier aplicación ya desarrollada es conveniente no sólo dentro de la lógica del negocio, sino también en cualquier ambiente de un servidor existente.

Una solución apropiada sería una librería que los desarrolladores puedan embeber en sus soluciones. Dado que las funciones de autenticación y acceso a bases de datos en la mayoría de los casos ya están implementadas, el mayor peso recaería en crear interfaces de comunicación, claras y fáciles de embeber. De esta manera se garantiza que el marco de trabajo jWebSocket esté abierto a un proceso de migración gradual.

Debido a lo novedoso que es el protocolo WebSocket, en la actualidad no existen integraciones entre servidores que proveen interfaces para la comunicación mediante este protocolo. La mayoría de las implementaciones WebSocket que existentes del lado del servidor, de una forma u otra están limitadas por el intercambio rudimentario de paquetes a bajo nivel, lo que provoca improductividad en los equipos de desarrollo que pretenden utilizar estas tecnologías. Cuando esto sucede, la solución más óptima es integrar un marco de trabajo especializado en el desarrollo de aplicaciones mediante el protocolo WebSocket a la aplicación existente, garantizando que este provea funciones adicionales e interfaces a la lógica existente en la aplicación subyacente. Por ejemplo, si se trabaja con un Servidor Web como *Apache*[21] y este no requiere acceso a la lógica del negocio del lado del servidor, entonces un servidor WebSocket paralelo sería un enfoque práctico para esta situación. Por tanto, el marco de trabajo debería usarse en su propio Servidor WebSocket o integrado en uno existente, que es más apropiado para determinados requisitos. Esto es útil cuando el marco de trabajo provee una determinada cantidad de características estándares del lado del servidor. Entre ellas se pueden mencionar el intercambio de ficheros, RPCs (*Remote Procedure Calls*, Llamadas a Procedimientos Remotos), servicios de correo y objetos compartidos. (Framework Approach for WebSockets, 2011)

El marco de trabajo jWebSocket considera todas las implementaciones WebSocket del lado del servidor como posibles integraciones. Para realizar una integración de jWebSocket con cualquier tecnología existente, básicamente se requiere implementar una interfaz, que está diseñada para modelar el servidor de bajo nivel que encapsula todo el proceso de comunicación mediante el protocolo WebSocket.

Los primeros pasos para integrar contenedores de Servlet al marco de trabajo jWebSocket fueron dados por Alexander Schulze, líder del proyecto jWebSocket. La primera integración de jWebSocket se realizó con el Servidor Web Apache Tomcat en enero de 2010, se trataba de un Servlet que permitía embeber el servidor jWebSocket dentro del servidor web Tomcat para reutilizar de este el soporte al protocolo WebSocket, de esta manera se permitía la comunicación WebSocket y HTTP por el mismo puerto.

El primer motor que se desarrolló recibe el nombre de TCPEngine y el mismo utiliza un modelo de conexión en el que se le asigna un hilo para cada cliente conectado. Posteriormente el marco de trabajo jWebSocket incluyó soporte para múltiples implementaciones del protocolo WebSocket. El motor NettyEngine, por ejemplo, fue creado para jWebSocket para reutilizar su modelo de conexión basado en Java NIO, dicho modelo en teoría permite soportar miles de conexiones concurrentes sin afectar la escalabilidad del servidor, además beneficia a jWebSocket de soporte SSL. Posteriormente se implementó el motor de Jetty en su versión 8.0.0.M1, este fue el segundo motor que se creó para jWebSocket, dicha integración también incluía soporte SSL. A partir de este momento ya era posible realizar conexiones a través de los puertos estándares 80 y 443.

Paralelo a esto, el servidor de aplicaciones GlassFish utiliza el marco de trabajo Grizzly para la gestión de las conexiones cliente/servidor, Grizzly implementa soporte para la última versión del protocolo WebSocket y propone un modelo de conexión basado en NIO, lo que lo hace un candidato ideal para ser integrado al marco de trabajo jWebSocket.

1.3 Metodología usada para el desarrollo de la solución

En una aplicación informática, el proceso de desarrollo debe estar regido y orientado por una metodología de desarrollo de software, que guíe los procesos y permita tener un registro detallado del avance de la investigación. Las metodologías pueden ser robustas o ágiles. Las metodologías robustas o pesadas están concebidas para guiar el proceso de desarrollo de los software de gran envergadura, cuando un proyecto requiere de gran cantidad de documentación, vaya a ser realizado en un tiempo considerablemente largo y existe la posibilidad de que pase por las manos de varios equipos de trabajo. Por su parte, las metodologías ágiles intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en los clientes y los resultados. Se basan en promover iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto, logrando que se minimicen los riesgos desarrollando software en corto tiempo. (Metodologías Tradicionales Vs. Metodologías Ágiles, 2011).

Proceso Unificado de Desarrollo - RUP

RUP es un proceso formal que provee un acercamiento disciplinado para asignar

tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales, respetando cronograma y presupuesto. Fue desarrollado por Rational Software, y está integrado con toda la suite de herramientas Rational. Puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte. Es guiado por casos de uso y centrado en la arquitectura, iterativo e incremental y utiliza UML como lenguaje de notación.

Consta de 4 fases principales:

- ✓ **Inicio:** el objetivo en esta etapa es determinar la visión del proyecto.
- ✓ **Elaboración:** en esta etapa el objetivo es determinar la arquitectura óptima.
- ✓ **Construcción:** en esta etapa el objetivo es llegar a obtener la capacidad operacional inicial.
- ✓ **Transición:** el objetivo es llegar a obtener el despliegue del proyecto.

(Figueroa, y otros, 2011)

XP (Extreme Programming) Programación Extrema

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (Wells, 2009)

SCRUM

SCRUM es un proceso ágil y liviano que sirve para administrar y controlar el desarrollo de software. El desarrollo se realiza de forma iterativa e incremental (una iteración es un ciclo corto de construcción repetitivo). Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nuevas funcionalidades. SCRUM se enfoca en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión.

Sus principales características son: desarrollo mediante iteraciones y reuniones sistemáticas a lo largo del proyecto. (Schwaber, y otros, 2011)

SXP

SXP está compuesta por las metodologías SCRUM y XP, ofreciendo una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva. Esta metodología fomenta el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo, ayudando al líder del proyecto a tener un mejor control del mismo.

SXP consta de 4 fases principales:

- ✓ **Planificación-Definición:** donde se establece la visión, se fijan las expectativas y se realiza el aseguramiento del financiamiento del proyecto.
- ✓ **Desarrollo:** es donde se realiza la implementación del sistema hasta que esté listo para ser entregado.
- ✓ **Entrega:** es la puesta en marcha.
- ✓ **Mantenimiento:** es la fase donde se realiza el soporte para el cliente.

De cada una de estas fases se realizan numerosas actividades tales como el levantamiento de requisitos, la priorización de la Lista de Reserva del Producto, definición de las Historias de Usuario, Diseño, Implementación, Pruebas, entre otras; de donde se generan artefactos para documentar todo el proceso. Las entregas son frecuentes, y existe una refactorización continua, lo que permite mejorar el diseño cada vez que se le añada una nueva funcionalidad.

SXP está especialmente indicada para proyectos con pequeños equipos de trabajo, un constante cambio de requisitos o requisitos imprecisos, donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una alta flexibilidad. Fomenta el trabajo en equipo, con un objetivo claro, permitiendo el seguimiento y control de las tareas a realizar.

(SXP, Metodología Ágil para el Desarrollo de Software, 2010).

Debido a las grandes ventajas que proporcionan las metodologías ágiles, además de que el proyecto está formado por un equipo de trabajo pequeño, el cliente no tiene bien definido algunos requisitos y el desarrollo está orientado una entrega

rápida de resultados, se propone para esta investigación el uso de la metodología SXP.

SXP es la metodología seleccionada para la realización de la solución porque es ideal para proyectos pequeños y porque incorpora lo mejor de las metodologías XP y SCRUM. Además la metodología SXP es la recomendada por la UCI para proyectos de pequeña envergadura como la solución que se pretende desarrollar en la presente investigación.

1.4 Herramientas y Tecnologías usadas para el desarrollo de la solución

1.4.1 Lenguajes de programación y modelado

Lenguaje de Modelado Unificado - UML

UML (Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. (Rumbaugh, y otros, 2007)

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Se utiliza en la presente investigación el Lenguaje Modelado Unificado UML porque es lo suficientemente expresivo como para modelar sistemas complejos con muchas relaciones.

Lenguaje de programación Java

Es un lenguaje de programación orientado a objetos, moderno y de alto nivel, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir muchos errores, como la manipulación directa de punteros o memoria. Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL. (Gosling, y otros, 2005) Actualmente este lenguaje es el más utilizado para el desarrollo de aplicaciones por las ventajas que brindan sus características, las cuales serán expuestas a continuación.

- Orientado a objetos: Java fue diseñado como un lenguaje orientado a objetos que permite agrupar en estructuras encapsuladas los datos y los métodos. Ofreciendo proyectos más fáciles de gestionar y manejar, mejorando su calidad y reduciendo el número de proyectos fallidos.

- Independencia de la plataforma: ofrece programas escritos en el lenguaje Java, que dan la posibilidad de ejecutarse de igual manera en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, “write once, runanywhere”.

- Recolector de basura: permite una fácil creación y eliminación de objetos, mayor seguridad y puede que más rápida que en C++. El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java Runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste.

- Entorno de funcionamiento: El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un

mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.

El marco de trabajo jWebSocket utiliza el lenguaje de programación java porque posee una curva de aprendizaje muy rápida, agiliza el proceso de desarrollo de cualquier aplicación substancialmente, es un lenguaje multiplataforma y de código abierto. El uso de Java le permite al marco de trabajo la reutilización de múltiples implementaciones WebSocket como Motores propios del mismo. Java permite a los desarrolladores aprovechar la flexibilidad de la Programación Orientada a Objetos en el diseño de sus aplicaciones. La comunidad de Java es una de las más grandes del mundo, por lo tanto, el uso de este lenguaje le proporciona al marco de trabajo más usabilidad y documentación disponible. Todo esto hace posible que jWebSocket hoy goce de buena seguridad, estabilidad, escalabilidad y robustez.

Lenguaje de programación JavaScript

JavaScript es un lenguaje utilizado para realizar acciones dentro del ámbito de una página Web, permitiendo el desarrollo de interfaces de usuario mejoradas y páginas Web dinámicas. Es un lenguaje de programación interpretado, multiplataforma, orientado a eventos. Estrictamente no es un lenguaje orientado a objetos (solo maneja scripts), ya que carece de los conceptos como herencia y métodos que tienen lenguajes como C++ y Java, pero es posible definir un objeto dentro de la página Web y sobre ese objeto definir a su vez diferentes eventos que producirán la aplicación o salida deseada ofreciendo la posibilidad de crear aplicaciones “on-.line” o modificar páginas Web en tiempo real, por ejemplo, cambiar el aspecto de la página Web. Actualmente, todos los navegadores incluyen JavaScript y es uno de los lenguajes más populares para la Web. (Paradigmas de la Programación: JavaScript y Python, Agosto 2010)

1.4.2 Otros Lenguajes

HTML

Definiéndolo de forma sencilla, "HTML es lo que se utiliza para crear todas las páginas Web de Internet". Más concretamente, HTML es el lenguaje con el que se "escriben" la mayoría de páginas Web.

Los diseñadores utilizan el lenguaje HTML para crear sus páginas Web, los

programas que utilizan los diseñadores generan páginas escritas en HTML y los navegadores que utilizamos los usuarios muestran las páginas Web después de leer su contenido HTML.

Aunque HTML es un lenguaje que utilizan los ordenadores y los programas de diseño, es muy fácil de aprender y escribir por parte de las personas. En realidad, HTML son las siglas de HyperTextMarkupLanguage y más adelante se verá el significado de cada una de estas palabras.

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado World Wide WebConsortium , más conocido como W3C. Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de forma muy similar en cualquier navegador de cualquier sistema operativo.

El propio W3C define el lenguaje HTML como "un lenguaje reconocido universalmente y que permite publicar información de forma global". Desde su creación, el lenguaje HTML ha pasado de ser un lenguaje utilizado exclusivamente para crear documentos electrónicos a ser un lenguaje que se utiliza en muchas aplicaciones electrónicas como buscadores, tiendas online y banca electrónica. (Eguíluz Pérez, 2009)

1.4.3 Marcos de Trabajo del lado del Servidor

Marco de trabajo jWebSocket

jWebsocket es un marco de trabajo de código abierto para el desarrollo de aplicaciones web estacionarias y móviles basado en Java en el lado del servidor y JavaScript en el lado del cliente. jWebsocket establece un modelo de token. Los tokens son datos abstractos que, a través de una estructura jerárquica y una API, proporcionan métodos de acceso a los contenidos. Con el objetivo de realizar una abstracción en la manipulación de los diferentes formatos, el marco de trabajo convierte los paquetes de datos entrantes y salientes en tokens. El cliente nativo soporta el intercambio de paquetes en los formatos JSON, XML y CSV, que en entornos específicos se pueden utilizar sin la necesidad de manejarlos a través de

tokens. El cliente jWebsocket tiene una arquitectura de complementos que permite aumentar con facilidad sus funcionalidades.

El servidor jWebsocket está diseñado para funcionar como servidor de comunicaciones o como servidor web, brindando total flexibilidad. En la primera opción jWebsocket proporciona un archivo *.jar*, ofreciendo la ventaja de ejecutarse fácilmente desde una línea de comandos e integrarse a la biblioteca de una aplicación existente de Java. En la actualidad hay algunos servidores que ya soportan Websockets y otros que no, por lo que jWebsocket se integra a servidores como Tomcat o Apache para lograr una comunicación Websockets. En caso de que los servidores soporten de manera nativa Websockets, como el caso de Jetty o GlassFish, se incluyen las funciones de comunicación del marco de trabajo jWebsocket, pero los motores internos se apagan y el anfitrión se utiliza. Esto asegura que no haya mecanismos de seguridad adicionales.

jWebsocket como servidor WebSocket proporciona un conjunto importante de funcionalidades y su arquitectura extensible mediante complementos permite añadir fácilmente características adicionales a un sistema independiente. Por otra parte los administradores pueden configurar el servidor exactamente como sea necesario y dejar a un lado todos los módulos que no necesiten. En un clúster los componentes se pueden utilizar como servicios, por lo que jWebsocket perfectamente es compatible con SOA (*Service Oriented Architectures*) en un entorno totalmente basado en eventos. Estas características muestran la fortaleza y flexibilidad del marco de trabajo para el desarrollo de aplicaciones web estacionarias y móviles, multiplataforma, multisectorial y compatible con todos los navegadores.(Framework Approach for WebSockets, 2011)

La presente investigación no sólo utiliza jWebSocket, sino que lo mejora añadiéndole soporte para múltiples motores. El hecho de crear un nuevo motor para jWebSocket, implica un aumento considerable de la comunidad de desarrolladores, mayor documentación al respecto y una mayor usabilidad. Además le proporciona una herramienta poderosa a otros desarrolladores para realizar aplicaciones basadas en WebSocket utilizando las ventajas del marco de trabajo jWebSocket. Un nuevo motor para jWebSocket garantizaría incluir jWebSocket en una nueva plataforma, pudiendo migrar las aplicaciones web al uso del protocolo

WebSocket con el menor cambio posible.

Servidor de Aplicaciones GlassFish

GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación.

Existen varias definiciones de GlassFish, entre ellas se puede destacar la definición dada por Alexis Moussine-Pouchkine, para quien GlassFish no sólo es un Servidor de Aplicaciones, sino también, una comunidad activa de desarrolladores. La comunidad de GlassFish se dedica a crear y mejorar muchos componentes necesarios, entre ellos se pueden mencionar módulos para tecnologías JCP incluidas en JavaEE5, entre los más conocidos se encuentran JSP Y JSF. GlassFish también posee muchas herramientas como Hudson (para la integración Continua) y una infraestructura muy útil como Grizzly, el marco de trabajo basado en el servidor NIO. La comunidad de GlassFish también mantiene un repositorio Maven para todos estos componentes. (Moussine-Pouchkine, y otros, 2007)

Para esta investigación, basándose en los conceptos mencionados anteriormente, se asume como definición de GlassFish, un Servidor de Aplicaciones distribuido bajo una licencia dual, a través de la licencia CDDL y la GNU GPL, totalmente gratuito y de código libre. Su versión comercial es denominada Oracle GlassFish Enterprise Server (antes se denominaba Sun GlassFish Enterprise Server). GlassFish soporta todas las especificaciones definidas por la API de Java EE tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. También provee especificaciones para otros componentes de JavaEE, entre ellos se pueden mencionar EJB (Enterprise Java Beans), Servlets, Portlets, JSP, entre otras tecnologías. Todo esto permite a los desarrolladores contar con una poderosa herramienta para crear aplicaciones empresariales escalables, portables y que se integren fácilmente con las tecnologías existentes.

GlassFish utiliza un derivado de Apache Tomcat para funcionar como Contenedor de Servlet sirviendo contenido Web a través de un componente adicional llamado Grizzly que utiliza Java New I/O (NIO) para la comunicación cliente-servidor a bajo nivel, garantizando gran seguridad, velocidad y escalabilidad en sus aplicaciones.

Los desarrolladores de aplicaciones se enfocan en la creación de programas según la especificación Java EE sin preocuparse por qué Servidor de Aplicaciones utilizar. Cualquier aplicación que se desarrolle de acuerdo esta especificación puede ser desplegada en cualquier Servidor de Aplicaciones Java EE. Para la presente investigación se seleccionó el Servidor de Aplicaciones GlassFish debido a que este reutiliza el marco de trabajo Grizzly como base tecnológica para garantizar grandes niveles de escalabilidad y disponibilidad sobre la gestión de conexiones cliente-servidor. Grizzly a su vez, es utilizado por una gran cantidad de sistemas y proyectos, tal es el caso de *JXTA*[22], *Sailfin*[23], *Jersey*[24], *RestLet*[25], entre otros. Grizzly posee una implementación WebSocket de bajo nivel, que permite gestionar conexiones e intercambio de mensajes por este protocolo, por tanto, al reutilizar Grizzly estamos aprovechando todo su soporte para esta tecnología.

Se decide integrar GlassFish con jWebSocket porque es un servidor de aplicaciones de código abierto, de fácil instalación para nuevos usuarios, posee un soporte completo con Java EE 5 y un soporte básico para aplicaciones basadas en WebSocket, posee una integración total con Netbeans. GlassFish para la comunidad cuenta con una documentación muy avanzada sobre uso, administración y desarrollo lo que garantizaría un aumento considerable de la usabilidad de jWebSocket. El uso de jWebSocket dentro de un servidor GlassFish permite crear aplicaciones basadas en WebSocket aprovechando todas las ventajas que brinda jWebSocket como servidor de aplicaciones WebSocket.

1.4.4 Servidores Web.

Servidor Web Grizzly.

Grizzly es un marco de trabajo que permite la comunicación a través de múltiples protocolos como HTTP y UDP, posee una gran rapidez para desarrollar aplicaciones que requieran una comunicación cliente-servidor a bajo nivel. Está basado en NIO (New Input/Output) que no es más que una colección de APIs de alto rendimiento para proveer acceso a operaciones de entrada y salida a bajo nivel en cualquier tipo de sistemas operativos a través de puertos.

Grizzly puede ser extendido en diferentes lugares, desde la administración de un flujo de bytes a bajo nivel hasta un manejo de hilos avanzado. (Arcand, 2007)

El marco de trabajo Grizzly es actualmente utilizado por muchos proyectos:

- Internamente lo usan:
  - Tango (WSIT) JRuby on Rails
  - Alaska (Open ESJ)
  - GlassFish
  - Phobos (Integración con NetBeans)
- Externamente lo usan:
  - AsyncWeb
  - Jetty
  - Brane Corporation/Oracle
  - 4Homedia
  - Ning

El hecho de que jWebSocket utilice el marco de trabajo Grizzly como motor para la integración con GlassFish, provee a los usuarios un producto con un alto rendimiento y escalabilidad en ambientes donde se necesite soportar múltiples clientes conectados de manera concurrente. Además, este motor le permite a jWebSocket servir Páginas Web y conexiones WebSocket utilizando puertos seguros para los servidores proxy.

Servidor Web Apache HTTP Server.

Es un servidor Web multiplataforma, con licencia de software libre, altamente utilizado para la publicación de aplicaciones en la Web. Apache HTTP Server presenta muchas características y funciones que lo califica como un servidor robusto y rápido de código abierto.

Apache es altamente configurable, posee bases de datos de autenticación y negociado de contenido, tiene amplia aceptación en la red, es uno de los servidores web más usados. La mayoría de sus vulnerabilidades de seguridad han sido descubiertas y resueltas.

Este servidor es utilizado en el nodo de la Facultad Regional de Artemisa para la publicación de las aplicaciones y sitios web.

1.4.5 Herramientas para la construcción de proyectos

Maven

Maven es una herramienta software para la gestión y construcción de proyectos

Java. Su funcionalidad es parecida a Apache Ant de manera que permite compilar, ejecutar pruebas o realizar distribuciones pero con la diferencia que trata de forma automática las dependencias del proyecto. Una de las más importantes características es su actualización en línea mediante servidores repositorios. Maven es capaz de descargar nuevas actualizaciones de las bibliotecas de las que depende un proyecto dado y de igual manera subir una nueva distribución a un repositorio de versiones, dejándola al acceso de todos los usuarios.

Maven posee licencia de software libre y es utilizado desde el comienzo por el proyecto jWebSocket.

1.4.6 Herramientas CASE

Se puede definir como herramientas CASE al conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software.

CASE se define también como:

- Conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

- La sigla genérica para una serie de programas y una filosofía de desarrollo de software que ayuda a automatizar el ciclo de vida de desarrollo de los sistemas.

- Una innovación en la organización, un concepto avanzado en la evolución de tecnología con un potencial efecto profundo en la organización. Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales. (INSTITUTO NACIONAL DE ESTADISTICA E INFORMATICA, 1999)

Visual Paradigm

Es una herramienta CASE para el diseño basado en UML, diseñada para ayudar en el proceso de desarrollo de software. Soporta los estándares más usados de la industria como UML, SysML, BPMN y XMI. Ofrece además un set completo de herramientas de desarrollo que todos los equipos necesitan tales como:

- Captura de requerimientos

- Planificación del software
- Planificación de las pruebas
- Modelado de clases
- Modelado de datos (VP, 2012)

Visual Paradigm multiplataforma, es capaz de integrarse con el entorno de desarrollo NetBeans y Eclipse. Además posee licencia gratuita para usos no comerciales y licencia comercial para uso comercial. La UCI posee licencia de esta herramienta para uso comercial.

Muchas empresas se han extendido a la adquisición de herramientas CASE, con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema, desde el principio, hasta el final y así incrementar su posición en el mercado competitivo. La Universidad de las Ciencias Informáticas ha tenido en cuenta otras de las herramientas CASE existentes en el mundo para el modelado de software como Rational Rose, pero como la política de desarrollo de la Universidad es crear aplicaciones completamente bajo software libre, se tomó la decisión de seleccionar como herramienta CASE a Visual Paradigm. Esta, a pesar de no ser libre, es multiplataforma y cuenta con una licencia comercial poseída por la Universidad con fines productivos y educacionales.

1.4.7 Herramientas de Control de Versiones

Subversion

Subversion es un sistema de control de versiones libre y de código fuente abierto. Es decir, Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de "máquina del tiempo".

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo

conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único, si se ha hecho un cambio incorrecto a los datos simplemente se deshace ese cambio. (Ben Collins-Sussman, 2004)

Subversion posee licencia de software libre y es utilizado actualmente por todos los centros de desarrollo de la UCI, en especial por el proyecto jWebSocket de la Facultad Regional de Artemisa desde su creación.

1.4.8 Clientes de Control de Versiones

TortoiseSVN

Es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. TortoiseSVN maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio.

RapidSVN

Es una plataforma de interfaz gráfica de usuario, para el sistema de revisión de Subversion. Este proyecto también incluye un cliente de Subversion C++ API. RapidSVN está licenciado bajo la v3 de GNU General Public License.

Utiliza las mejores características de los clientes de otras arquitecturas de control de versiones. Si bien es bastante fácil para los nuevos usuarios de Subversion trabajar con él, también debe ser lo suficientemente potente como para que los usuarios con experiencia sean aún más productivos.

Se caracteriza por ser:

- ✓ Simple: Proporciona una interfaz fácil de usar para las características de Subversion.
- ✓ Eficiente: Simple para los principiantes pero lo suficientemente flexible como para aumentar la productividad para los usuarios de Subversion.

- ✓ Portátil: Se ejecuta en cualquier plataforma: Linux, Windows, Mac OS / X, Solaris.
- ✓ Rápido: Completamente escrito en C++.

(RapidSVN, 2011)

Esta herramienta se utiliza por el proyecto porque posee una interfaz gráfica amigable, sencilla, eficiente, fácil para principiantes, pero robusta para desarrolladores, además es portable, multiplataforma, rápido y soporta múltiples idiomas.

1.4.9 Entorno Integrado de Desarrollo – IDE

NetBeans

NetBeans IDE es una herramienta desarrollada por Sun Microsystems. Está completamente escrito en Java, por lo que puede ser utilizado desde cualquier sistema operativo compatible con la máquina virtual de Java. Permite el desarrollo de aplicaciones de escritorio, web y móviles. Es un producto libre y gratuito sin restricciones de uso. Es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Su misión consiste en evitar tareas repetitivas, facilitar la escritura correcta de código, disminuir el tiempo de depuración e incrementar la productividad del desarrollador. Cuenta con un depurador, perfilador de integración, herramientas para refactorizaciones, completamiento de código y control de versiones de archivos. (NetBeans, 2011)

Existen otros IDE que brindan potencialidades para el desarrollo de aplicaciones Java como Eclipse SDK. Sin embargo, para el desarrollo de la solución propuesta se selecciona como entorno integrado de desarrollo a NetBeans IDE debido a que se tiene una mayor experiencia y familiarización con esta herramienta. Además su versión 7.0.1 introduce un soporte para el desarrollo con la especificación JavaSE7 (Java Standard Edition) con las características de JDK7 (Java Development Kit). Esta versión también ofrece una integración mejorada con el servidor Oracle WebLogic, así como soporte para Oracle Database y GlassFish3.1. Otros puntos destacados incluyen soporte para Maven3 y HTML5, así como mejoras en el editor de Java. (NetBeans, 2011)

Conclusiones del capítulo

En este capítulo se realizó un estudio de las tendencias actuales en cuanto al desarrollo de aplicaciones en tiempo real para la Web y los marcos de trabajo que permiten el desarrollo de este tipo de aplicaciones. Se analizaron y se seleccionaron las principales herramientas, lenguajes y metodologías de desarrollo a utilizar para proveer a jWebSocket, la integración con otras implementaciones WebSocket del lado del servidor.

CAPÍTULO 2: CARACTERÍSTICAS, ANÁLISIS Y DISEÑO DEL SISTEMA

Introducción

En el presente capítulo se describe el funcionamiento y la estructura del motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket, destacando sus características distintivas. Se realiza el modelado de las historias de usuario del negocio con el objetivo de comprender su contexto. Además se detallan brevemente los artefactos de la metodología de desarrollo SXP propuesta en el capítulo anterior. Describiéndose los requisitos funcionales y no funcionales que comprenden la solución, así como las historias de usuario y las tareas de ingeniería asociadas a las mismas.

2.1. Propuesta de Solución

En la actualidad el surgimiento del protocolo WebSocket ha cambiado la forma de realizar aplicaciones, imponiendo un nuevo paradigma de desarrollo de software que ofrece un sinnúmero de beneficios. Por tanto, la comunidad de desarrolladores web y empresas que se dedican al negocio del software se han motivado a crear soluciones para dar soporte a este protocolo del lado del servidor. Múltiples librerías para diferentes lenguajes han sido desarrolladas con el objetivo de brindar un soporte cada día más completo a este protocolo. Entre los resultados más aceptados por la comunidad se tienen GlassFish, Oracle, entre otras han logrado crear soluciones escalables y de alto rendimiento para dar soporte a esta nueva tecnología.

La presente solución consiste en desarrollar un motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket. El motor deberá permitir básicamente manejar las conexiones WebSocket desde y hacia el marco de trabajo jWebSocket utilizando el proyecto Grizzly embebido en el servidor de aplicaciones GlassFish. De esta forma se delega el tratamiento del soporte a múltiples conexiones al marco de trabajo Grizzly.

El hecho de que jWebSocket incluya en su núcleo un motor basado en la implementación que brinda GlassFish para soportar conexiones WebSocket, le brinda al marco de trabajo diversos beneficios. GlassFish utiliza el marco de trabajo

Grizzly para realizar conexiones de tipo cliente-servidor a bajo nivel, basándose en la API de Java NIO. Java NIO se especializa en el uso de protocolos como HTTP, UDP, WebSocket, entre otros, por tanto, sería una herramienta muy útil para el marco de trabajo jWebSocket. Tanto Grizzly como GlassFish poseen una comunidad de desarrollo altamente activa, lo que garantizaría mayores niveles de usabilidad de jWebSocket. Hoy el marco de trabajo jWebSocket no permite realizar conexiones WebSocket y HTTP por el mismo puerto, los puertos que utiliza no son estándares y resultan inseguros para algunos servidores proxy. Con el uso de Grizzly se logrará aprovechar puertos estándares como el 80 y el 443. Estos puertos están permitidos y asegurados por los servidores proxy, por lo que no deberá existir desconfianza para los usuarios que decidan utilizar jWebSocket. Con este motor se garantizará un total soporte a futuros cambios en el protocolo WebSocket, teniendo en cuenta que Grizzly es utilizado por una gran cantidad de desarrolladores en el mundo y lo mantienen en constante desarrollo.

Con este motor se podrán crear aplicaciones web basadas en el protocolo WebSocket que gocen de todas las ventajas que brinda el estándar de Java EE, unido a la gran cantidad de herramientas y complementos que ofrece el marco de trabajo jWebSocket. Por tanto, con esta propuesta de solución se espera un aumento de los niveles de seguridad y usabilidad para el marco de trabajo jWebSocket.

2.3. Planificación del proyecto por Roles.

Algo que diga como: Para el proyecto de desarrollo del motor de GlassFish se planificaron distintos roles en el proyecto. Esto se hizo con el objetivo de lograr mayor eficiencia, control y organización de las responsabilidades a enfrentar en el desarrollo. En la **Tabla 1** mostrada a continuación se describen los roles definidos, sus principales responsabilidades y los nombres de los encargados de cada rol.

Tabla 1: Listado de roles y sus principales responsabilidades.

Rol	Responsabilidad	Nombre
Líder del Proyecto	Debe asegurar que el proyecto se está llevando a cabo de acuerdo con las prácticas y que todo funciona según lo planeado. Su principal trabajo es remover impedimentos y reducir riesgos del	Alexander Schulze

	producto. Coordina y facilita las reuniones. Asegura que se consiguen los objetivos de cada iteración.	
Gerente	Es el responsable de tomar las decisiones finales, acerca de estándares y convenciones a seguir durante el proyecto. Participa en la selección de objetivos y requerimientos. Tiene la responsabilidad de controlar el progreso y trabaja junto con el Jefe de Proyecto en la reducción de la Lista de Reserva del Producto.	Victor Antonio Barzana Crespo
Especialista	Es necesario que conozca a fondo el proceso para el desarrollo de software. Es una especialización que está activa, el miembro del grupo de trabajo que la desempeña siempre está ejecutándola y alcanzando un grado mayor de conocimientos en el tema.	Alexander Schulze
Consultor	Es un miembro externo del equipo con un conocimiento especifico en algún tema necesario para el proyecto, en el que puedan surgir problemas, además aportan ideas y experiencias para el beneficio del sistema en desarrollo. Esta es una especialización menos activa, quien la ejecuta funciona en este rol por un corto período de tiempo.	Alexander Schulze
Cliente	Participa en las tareas que involucran la lista de reserva del producto.	Alexander Schulze
Programador	Elabora el código de las nuevas funcionalidades a implementar. Escribe las pruebas unitarias. Debe existir una comunicación y coordinación adecuada entre los programadores y el resto del equipo.	Victor Antonio Barzana Crespo
Analista	Escribe las historias de usuario y las pruebas funcionales para validar su implementación.	Victor Antonio Barzana Crespo
Diseñador	Encargado del diseño del sistema y de los prototipos de interfaces, son los máximos responsables de la realización del diseño de las metáforas y supervisan el proceso de construcción.	Victor Antonio Barzana Crespo
Encargado de Pruebas	Es el encargado de ayudar al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.	Victor Antonio Barzana Crespo
Arquitecto	Se vincula con el analista y el diseñador ya que su trabajo tiene que ver con la estructura y el diseño del sistema. Ayuda en el diseño de las metáforas.	Victor Antonio Barzana

		Crespo

2.4. Modelo de Dominio.

Dentro de las actividades más importantes definidas en la metodología SXP se encuentra la definición del Modelo de Historias de Usuario del Negocio, en el cual se hace una detallada descripción del negocio en cuestión. Pero si dicho negocio no está bien definido entre los clientes y los ejecutores del proyecto; entonces es generado el llamado Modelo de Dominio.

A continuación en la **Figura # 1** se presenta el diagrama de dominio para la solución que se propone:

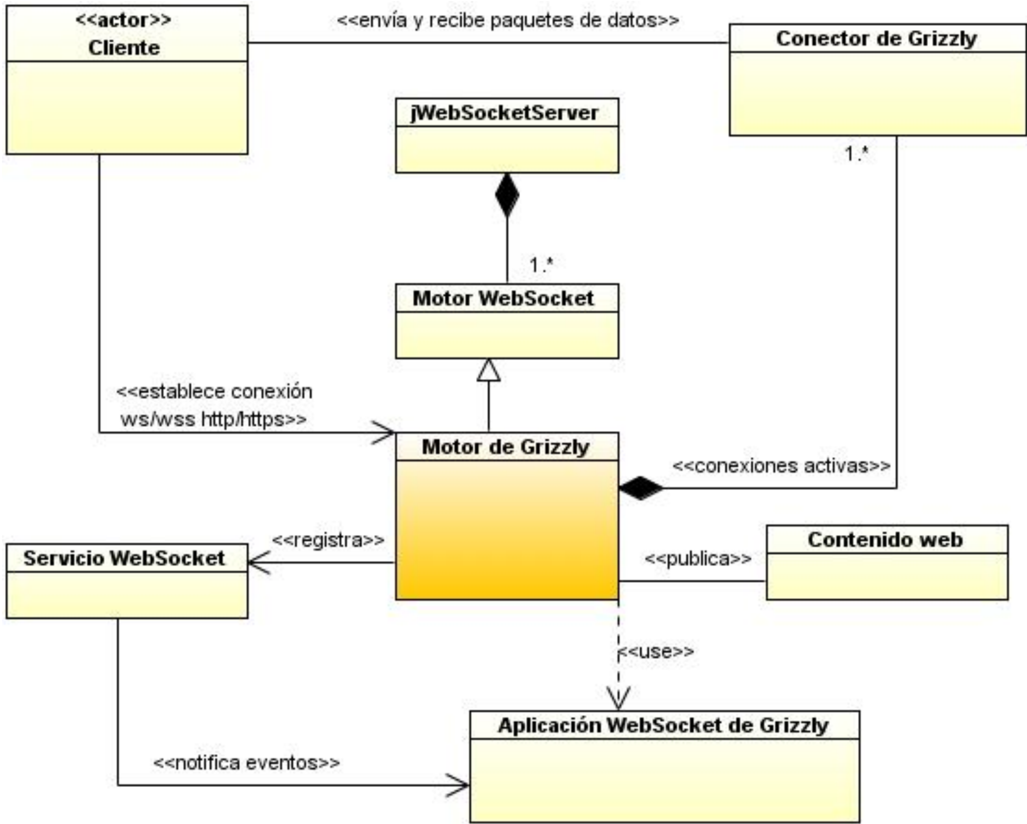


Figura # 1: Modelo de dominio de la solución.

En el diagrama se puede apreciar la estructura para el motor del marco de trabajo Grizzly para jWebSocket. Publique un Servidor HTTP para servir páginas web. A este servidor se podrán registrar múltiples Servicios o Escuchadores WebSocket, que serán los encargados de notificar cualquier acción que se realice en el protocolo WebSocket a las aplicaciones WebSocket. Estas estarán encargadas de

notificar los eventos WebSocket provenientes del servidor de GlassFish y reenviarlos hasta el marco de trabajo jWebSocket de manera transparente para los desarrolladores. Los llamados Conectores de Grizzly son una representación de alto nivel de la conexión con los clientes, el motor de Grizzly deberá mantener tantos conectores activos como Grizzly sea capaz de soportar. La solución debe permitir que el servidor de jWebSocket se ejecute utilizando el motor Grizzly, este debe permitir conexiones HTTP y WebSocket. Para esto se crea el servidor HTTP y se publican los demos de jWebSocket para que el usuario pueda acceder a estos desde una dirección URL y probar el funcionamiento correcto de este motor.

2.5. Lista de Reserva del Producto (LRP).

Recoge en una lista priorizada todo el trabajo a desarrollar en el proyecto, enfocado al desarrollo óptimo de una solución para el problema que se plantea. Esta lista puede crecer y modificarse a medida que se obtienen más conocimientos acerca del producto y del cliente. Con la restricción de que sólo puede cambiarse entre iteraciones. El objetivo es asegurar que el producto definido al terminar la lista es el más correcto, útil y competitivo posible y para esto la lista debe acompañar los cambios en el entorno y el producto. Puede estar conformada por requerimientos técnicos y del negocio, funciones, errores a reparar, defectos, mejoras y actualizaciones tecnológicas requeridas. Como se puede apreciar en la Tabla #2, la lista de reserva del producto está dividida en dos secciones. En la primera sección se definen los requisitos funcionales por niveles de importancia y en la segunda sección se definen todos los requisitos funcionales con los que deberá cumplir la solución propuesta.

Tabla # 2: Lista de reserva del producto.

Prioridad	Ítem *	Descripción	Estimación	Estimado por
Muy Alta				
	1	Identificar petición para aplicación WebSocket.	1 día	Analista
	2	Abrir conexión.	2 días	Analista
	3	Enviar paquetes WebSocket.	2 días	Analista

	4	Procesar paquetes entrantes desde GlassFish.	2 días	Analista
	5	Procesar fragmentos de paquetes entrantes.	2 días	Analista
	6	Comprobar el estado de la conexión (ping).	2 días	Analista
	7	Dar respuesta a comprobación de conexión (pong).	2 días	Analista
	8	Cerrar conexión.	2 días	Analista
	9	Cargar configuración del motor.	5 días	Analista
	10	Iniciar el motor de GlassFish.	5 días	Analista
	11	Notificar a todos los Componentes (PlugIns) del arranque del motor.	2 días	Analista
	12	Detener el motor de GlassFish.	5 días	Analista
	13	Notificar a todos los conectores que se detuvo el motor.	3 días	Analista
Alta				
	14	Unificar los puertos de GlassFish y jWebSocket (ws/wss y http/https).	3 días	Analista
	15	Definir el tiempo de espera de los clientes conectados.	1 día	Analista
	16	Definir el tamaño máximo de cada paquete.	1 día	Analista
Media				
	17	Inicializar la aplicación de ejemplo.	1 día	Analista
	18	Procesar peticiones WebSocket de la aplicación de ejemplo.	4 días	Analista
	19	Procesar peticiones HTTP de la aplicación de ejemplo.	4 días	Analista

	20	Definir el patrón de URL para la aplicación de ejemplo.	1 día	Analista
Baja				
RNF (Requisitos No Funcionales)				
	21	Garantizar la integridad y consistencia de los datos durante el intercambio de información entre el cliente y el servidor.		
	22	Se debe realizar la aplicación de forma versionable que permita darle mantenimientos al sistema a fin de aumentar las funcionalidades y/o corregir los errores del mismo a través de versiones posteriores.		
	23	Se documentará la aplicación con diferentes manuales con el objetivo de brindar una ayuda al usuario y un soporte para los desarrolladores.		
	24	El lenguaje de programación a utilizar para el desarrollo del módulo es Java y el IDE NetBeans 7.0.1.		
	25	La metodología de desarrollo a seguir es SXP y para la modelación se utilizará la herramienta Visual Paradigm 3.4.		
	26	Para el correcto funcionamiento de la aplicación es necesaria la versión 2.1.6 de Grizzly-		

		WebSockets y la versión 3.1.1 del servidor de aplicaciones GlassFish, cualquier versión del servidor de jWebSocket y la versión 7 de OpenJDK.		
	27	Todos los textos y mensajes en pantalla aparecerán en idioma inglés.		
	28	Los requisitos mínimos de hardware para el correcto funcionamiento de la aplicación son: 512 MB de memoria RAM, microprocesador Pentium IV, tarjeta red Fast Ethernet.		
	29	El código de la aplicación será liberado bajo la Licencia Pública General Reducida de GNU (LGPL, Lesser General Public License).		
	30	Para el desarrollo de la aplicación se han establecido pautas para la codificación que permitan mantener un código uniforme y legible.		

2.6. Historias de Usuario y Tareas de Ingeniería.

Las historias de usuario en la metodología de desarrollo SXP son las que describen las tareas que el sistema debe hacer, cuestión que depende en gran medida de las especificaciones realizadas por el cliente. Se escriben con un lenguaje natural y con palabras concisas para no exceder su tamaño en unas pocas líneas de texto. Van a ser la guía para la construcción posterior de las pruebas Funcionales comprobando de esta manera la correcta implementación de las historias de

usuario que engloban las funcionalidades que debe tener el motor.

A continuación se muestran una serie de tablas pertenecientes a cada historia de usuario donde se presentan los detalles de cada una de ellas:

Tabla #3: Historia de Usuario (Crear conector de Grizzly-GlassFish).

Historia de Usuario	
Número: HU_1	**Nombre Historia de Usuario:** Crear conector de Grizzly-GlassFish.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Victor Antonio Barzana Crespo	**Iteración Asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 3 semanas
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3 semanas
Descripción: La presente historia de usuario tiene como objetivo desarrollar un conector genérico para Grizzly-WebSocket que será el encargado de establecer la conexión entre cada uno de los clientes de jWebSocket. Cada uno de los motores WebSocket controla sus clientes a través de conectores activos en el servidor, por tanto, desarrollar un conector garantiza tener el control total, envío y recibo de datos entre los clientes y el servidor. Para esto es necesario identificar petición para aplicación WebSocket, abrir la conexión, enviar paquetes WebSocket, procesar los paquetes entrantes desde GlassFish, procesar los fragmentos de los paquetes entrantes, comprobar el estado de la conexión (ping), dar respuesta a la comprobación de la conexión (pong) y cerrar la conexión.	
Observaciones: Ninguna.	
Prototipo de interface: Ninguna.	

Para el desarrollo de la historia de usuario anterior se diseñaron un grupo de tareas de ingeniería. En las tablas 4, 5, 6, 7, 8, 9, 10 y 11 se muestran los detalles que definen cada tarea de ingeniería para esta historia de usuario.

Tabla #4: Identificar petición para aplicación WebSocket.

Tarea de Ingeniería	
Número Tarea: 1.1	**Número Historia de Usuario:** HU_1

Nombre Tarea: Identificar petición para aplicación WebSocket.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 día
Fecha Inicio: 3/10/2011	**Fecha Fin:** 4/10/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Identificar si la petición entrante es un estrechón de manos (handshake) WebSocket, para sino, manejar la petición como HTTP.	

Tabla #5: Abrir conexión.

Tarea de Ingeniería		
Número Tarea: 1.2	**Número Historia de Usuario:** HU_1	
Nombre Tarea: Abrir conexión.		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 2 días
Fecha Inicio: 4/10/2011		**Fecha Fin:** 6/10/2011
Programador Responsable: Victor Antonio Barzana Crespo		
Descripción: Abrir la conexión desde un cliente hasta GlassFish y lanzar el método onOpen del servidor de jWebSocket.		

Tabla #6: Enviar paquetes WebSocket.

Tarea de Ingeniería		
Número Tarea: 1.3	**Número Historia de Usuario:** HU_1	
Nombre Tarea: Enviar paquetes WebSocket.		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 2 días
Fecha Inicio: 6/10/2011		**Fecha Fin:** 10/10/2011
Programador Responsable: Victor Antonio Barzana Crespo		
Descripción: Permitir el envío de paquetes WebSocket a través de la implementación WebSocket de GlassFish.		

Tabla #7: Procesar paquetes entrantes desde GlassFish.

Tarea de Ingeniería	
Número Tarea: 1.4	**Número Historia de Usuario:** HU_1

Nombre Tarea: Procesar paquetes entrantes desde GlassFish.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2 días
Fecha Inicio: 10/10/2011	**Fecha Fin:** 12/10/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Recibir todos los paquetes WebSocket que llegan desde un cliente hasta el servidor de GlassFish y redireccionarlos a jWebSocket.	

Tabla #8: Procesar fragmentos de paquetes entrantes.

Tarea de Ingeniería		
Número Tarea: 1.5	**Número Historia de Usuario:** HU_1	
Nombre Tarea: Procesar fragmentos de paquetes entrantes.		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 2 días
Fecha Inicio: 12/10/2011		**Fecha Fin:** 14/10/2011
Programador Responsable: Victor Antonio Barzana Crespo		
Descripción: Recibir todos los fragmentos de paquetes WebSocket que llegan desde un cliente hasta el servidor de GlassFish y redireccionarlos a jWebSocket (Sólo se ejecuta cuando hay fragmentación de paquetes).		

Tabla #9: Comprobar el estado de la conexión (ping).

Tarea de Ingeniería		
Número Tarea: 1.6	**Número Historia de Usuario:** HU_1	
Nombre Tarea: Comprobar el estado de la conexión (ping).		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 2 días
Fecha Inicio: 17/10/2011		**Fecha Fin:** 19/10/2011
Programador Responsable: Victor Antonio Barzana Crespo		
Descripción: Recibir peticiones de los clientes para saber si la conexión con el servidor está activa o no, lanzar el evento onPing del servidor de jWebSocket.		

Tabla #10: Dar respuesta a comprobación de conexión (pong).

Tarea de Ingeniería

Número Tarea: 1.7	**Número Historia de Usuario:** HU_1
Nombre Tarea: Dar respuesta a comprobación de conexión (pong).	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2 días
Fecha Inicio: 18/10/2011	**Fecha Fin:** 20/10/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Enviar respuesta desde el servidor de jWebSocket para un cliente a quien le interese saber que la conexión está activa, lanzar el evento onPong del servidor de jWebSocket.	

Tabla #11: Cerrar conexión.

Tarea de Ingeniería	
Número Tarea: 1.8	**Número Historia de Usuario:** HU_1
Nombre Tarea: Cerrar conexión.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2 días
Fecha Inicio: 20/10/2011	**Fecha Fin:** 24/10/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Esperar a que un cliente cierre la conexión, detener todos los hilos relacionados con el mismo, lanzar el evento onClose del servidor de jWebSocket.	

Tabla #12: Historia de Usuario 2 (Crear motor de Grizzly-GlassFish).

Historia de Usuario	
Número: HU_2	**Nombre Historia de Usuario:** Crear motor de Grizzly-GlassFish.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Victor Antonio Barzana Crespo	**Iteración Asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 4 semanas
Riesgo en Desarrollo: Alto	**Puntos Reales:** 4 semanas
Descripción: La presente historia de usuario tiene como objetivo crear un motor independiente de GlassFish que utilice las librerías Grizzly-WebSockets que garanticen la comunicación bidireccional entre los clientes y el servidor de	

jWebSocket. Para esto es necesario cargar la configuración del motor, iniciar el motor de GlassFish, notificar a todos los conectores el arranque del motor, detener el motor de GlassFish y notificar a todos los conectores que se detuvo el motor.
Observaciones: Ninguna.
Prototipo de interface: Ninguna.

Para el desarrollo de la historia de usuario anterior se diseñaron un grupo de tareas de ingeniería. En las tablas 13, 14,15, 16, 17 y 18 se muestran los detalles que definen cada tarea de ingeniería para esta historia de usuario.

Tabla #13: Cargar configuración del motor.

Tarea de Ingeniería	
Número Tarea: 2.1	**Número Historia de Usuario:** HU_2
Nombre Tarea: Cargar configuración del motor.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2 días
Fecha Inicio: 24/10/2011	**Fecha Fin:** 26/10/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Cargar la configuración definida para el motor de GlassFish dentro de la configuración de jWebSocket en el fichero jWebSocket.xml.	

Tabla #14: Iniciar el motor de GlassFish.

Tarea de Ingeniería	
Número Tarea: 2.2	**Número Historia de Usuario:** HU_2
Nombre Tarea: Iniciar el motor de GlassFish.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 5 días
Fecha Inicio: 26/10/2011	**Fecha Fin:** 1/11/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Crear un servidor HTTP y añadirle escuchadores WebSocket, utilizar la configuración definida en el motor de GlassFish dentro del fichero	

jWebSocket.xml.

Tabla #15: Notificar a todos los Componentes (PlugIns) del arranque del motor.

Tarea de Ingeniería		
Número Tarea: 2.3	**Número Historia de Usuario:** HU_2	
Nombre Tarea: Notificar a todos los Componentes (PlugIns) del arranque del motor.		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 2 días
Fecha Inicio: 1/11/2011		**Fecha Fin:** 3/11/2011
Programador Responsable: Victor Antonio Barzana Crespo		
Descripción: Recorrer la lista de Componentes y ejecutar en una cadena el método engineStarted dentro de cada uno de ellos.		

Tabla #16: Detener el motor de GlassFish.

Tarea de Ingeniería		
Número Tarea: 2.4	**Número Historia de Usuario:** HU_2	
Nombre Tarea: Detener el motor de GlassFish.		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 5 días
Fecha Inicio: 3/11/2011		**Fecha Fin:** 9/11/2011
Programador Responsable: Victor Antonio Barzana Crespo		
Descripción: Finalizar los hilos de ejecución asignados a cada uno de los conectores activos en el servidor de jWebSocket.		

Tabla #17: Notificar a todos los conectores que se detuvo el motor.

Tarea de Ingeniería		
Número Tarea: 2.5	**Número Historia de Usuario:** HU_2	
Nombre Tarea: Notificar a todos los conectores que se detuvo el motor.		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 3 días
Fecha Inicio: 9/11/2011		**Fecha Fin:** 14/11/2011
Programador Responsable: Victor Antonio Barzana Crespo		

Descripción: Notificar a todos clientes de que el motor GlassFish se ha detenido, enviar una razón a los clientes de que se cerró el servidor (closeReason). Lanzar el método engineStopped dentro de los Componentes (PlugIns) de jWebSocket.

Tabla #18: Historia de Usuario 3 (Integrar configuración de ambos servidores).

Historia de Usuario	
Número: HU_3	**Nombre Historia de Usuario:** Integrar configuración.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Victor Antonio Barzana Crespo	**Iteración Asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 1 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 1 semanas
Descripción: : La presente historia de usuario tiene como objetivo crear una integración de la configuración del servidor jWebSocket con el servidor GlassFish para lograr un único fichero de configuración centralizado donde se definan todas las configuraciones correspondientes a ambos servidores y al motor de GlassFish. Para esto es necesario unificar los puertos de GlassFish y jWebSocket (ws/wss y http/https), definir el tiempo de espera de los clientes conectados y definir el tamaño máximo que tendrá cada paquete.	
Observaciones: Ninguna.	
Prototipo de interface: Ninguna.	

Para el desarrollo de la historia de usuario anterior se diseñaron un grupo de tareas de ingeniería. En las tablas 19, 20 y 21 se muestran los detalles que definen cada tarea de ingeniería para esta historia de usuario.

Tabla #19: Unificar los puertos de GlassFish y jWebSocket (ws/wss y http/https).

Tarea de Ingeniería	
Número Tarea: 3.1	**Número Historia de Usuario:** HU_3
Nombre Tarea: Unificar los puertos de GlassFish y jWebSocket (ws/wss y	

http/https).	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 3 días
Fecha Inicio: 14/11/2011	**Fecha Fin:** 17/11/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Definir el puerto 80 para las conexiones ws y http, utilizar el puerto 443 por defecto para las conexiones wss y https, puertos estándares y seguros para los servidores proxy.	

Tabla #20: Definir el tiempo de espera de los clientes conectados.

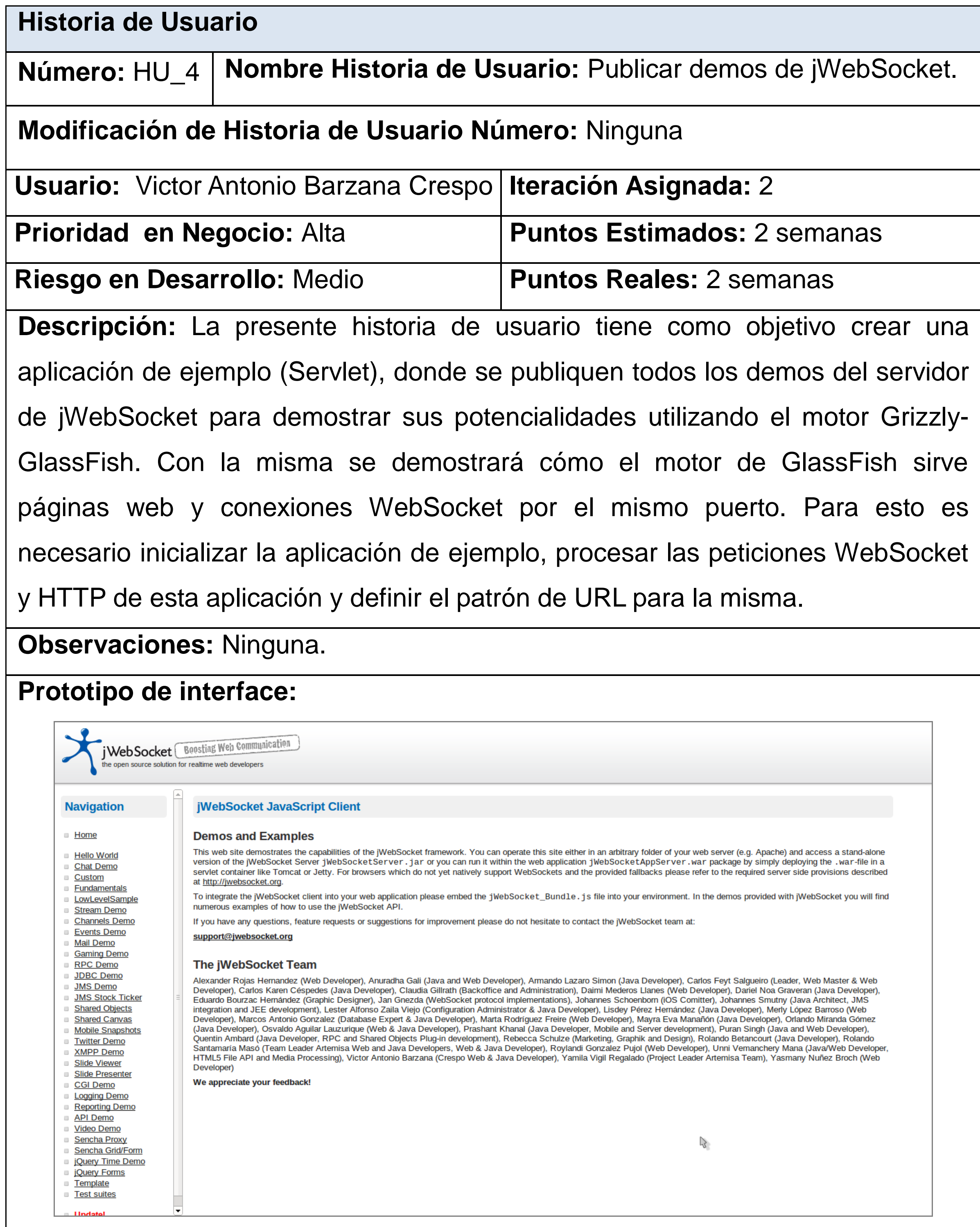
Tarea de Ingeniería	
Número Tarea: 3.2	**Número Historia de Usuario:** HU_3
Nombre Tarea: Definir el tiempo de espera de los clientes conectados.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 día
Fecha Inicio: 17/11/2011	**Fecha Fin:** 18/11/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Por motivos de seguridad se define un tiempo de vida límite para las conexiones WebSocket , en caso de que se exceda este tiempo se le cierra la conexión al cliente y se le notifica de la ocurrencia de un “timeout”.	

Tabla #21: Definir el tamaño máximo de cada paquete.

Tarea de Ingeniería	
Número Tarea: 3.3	**Número Historia de Usuario:** HU_3
Nombre Tarea: Definir el tamaño máximo de cada paquete.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 día
Fecha Inicio: 17/11/2011	**Fecha Fin:** 18/11/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Se define el tamaño máximo para los paquetes WebSocket, estos no deben exceder el número de bytes definido en esta parte de la configuración del motor de GlassFish.	

Tabla #22: Historia de Usuario 4 (Publicar demos de jWebSocket).

Historia de Usuario	
Número: HU_4	**Nombre Historia de Usuario:** Publicar demos de jWebSocket.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Victor Antonio Barzana Crespo	**Iteración Asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2 semanas
Descripción: La presente historia de usuario tiene como objetivo crear una aplicación de ejemplo (Servlet), donde se publiquen todos los demos del servidor de jWebSocket para demostrar sus potencialidades utilizando el motor Grizzly-GlassFish. Con la misma se demostrará cómo el motor de GlassFish sirve páginas web y conexiones WebSocket por el mismo puerto. Para esto es necesario inicializar la aplicación de ejemplo, procesar las peticiones WebSocket y HTTP de esta aplicación y definir el patrón de URL para la misma.	
Observaciones: Ninguna.	
Prototipo de interface:	



Para el desarrollo de la historia de usuario anterior se diseñaron un grupo de tareas de ingeniería. En las tablas 23, 24, 25 y 26 se muestran los detalles que definen cada tarea de ingeniería para esta historia de usuario.

Tabla #23: Inicializar la aplicación de ejemplo.

Tarea de Ingeniería	
Número Tarea: 4.1	**Número Historia de Usuario:** HU_4
Nombre Tarea: Inicializar la aplicación de ejemplo.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 día
Fecha Inicio: 21/11/2011	**Fecha Fin:** 22/11/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Garantizar que al ejecutar el servlet se publiquen todos los demos de jWebSocket que están incluidos en el paquete del cliente para probar el funcionamiento del motor de GlassFish.	

Tabla #24: Procesar peticiones WebSocket de la aplicación de ejemplo.

Tarea de Ingeniería	
Número Tarea: 4.2	**Número Historia de Usuario:** HU_4
Nombre Tarea: Procesar peticiones WebSocket de la aplicación de ejemplo.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 4 días
Fecha Inicio: 22/11/2011	**Fecha Fin:** 28/11/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Garantizar que el servlet reciba conexiones WebSocket desde los demos de jWebSocket y las envíe hasta el marco de trabajo de jWebSocket.	

Tabla #25: Procesar peticiones HTTP de la aplicación de ejemplo.

Tarea de Ingeniería	
Número Tarea: 4.3	**Número Historia de Usuario:** HU_4
Nombre Tarea: Procesar peticiones HTTP de la aplicación de ejemplo.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 4 días
Fecha Inicio: 28/11/2011	**Fecha Fin:** 2/12/2011
Programador Responsable: Victor Antonio Barzana Crespo	
Descripción: Servir la página de los demos de jWebSocket por HTTP, garantizando peticiones GET y POST desde esta aplicación.	

Tabla #26: Definir el patrón de URL para la aplicación de ejemplo.

Tarea de Ingeniería		
Número Tarea: 4.4	**Número Historia de Usuario:** HU_4	
Nombre Tarea: Definir el patrón de URL para la aplicación de ejemplo.		
Tipo de Tarea : Desarrollo		**Puntos Estimados:** 1 día
Fecha Inicio: 5/12/2011		**Fecha Fin:** 6/12/2011
Programador Responsable: Victor Antonio Barzana Crespo		
Descripción: Definir la URL que va a tener la aplicación de ejemplo para publicar los demos de jWebSocket, el patrón tendría una definición como se muestra a continuación: http://localhost:80/jWebSocketGrizzlyDemos.		

2.7. Plan de Releases

El plan de releases o iteraciones permite realizar las entregas intermedias y la entrega final del motor de GlassFish para el marco de trabajo jWebSocket. Tiene como entrada la relación de Historias de Usuario definidas previamente para resolver los requerimientos. Para colocar una historia en cada iteración se tiene en cuenta la prioridad que definió el cliente para dicha historia. Como resultado de la priorización de historias de usuario se llegó a la siguiente planificación:

Tabla #27: Plan de Releases.

Release	**Descripción de la iteración**	**Orden de la HU a implementar**	**Duración total**
2	El objetivo principal de esta iteración es liberar un motor de jWebSocket para GlassFish basado en la librería Grizzly-WebSockets que permita integrar aplicaciones hechas con jWebSocket dentro de proyectos hechos con GlassFish. Además en esta iteración se van a	HU_1 HU_2 HU_3 HU_4	10 semanas

	desarrollar aplicaciones de ejemplo para probar el funcionamiento correcto del motor implementado, así como lograr ejecutar todos los demos de jWebSocket en una aplicación de GlassFish.		

2.8. Descripción de la Arquitectura

Para desarrollar el motor de GlassFish para jWebSocket se seleccionó una Arquitectura basada en componentes. Este proceso de construcción de una pieza de software con componentes ya existentes, da origen al principio de reutilización del software, mediante el cual se promueve que los componentes sean implementados de una forma que permita su utilización funcional sobre diferentes sistemas en el futuro. Se debe entonces, para terminar de definir la arquitectura basada en componentes, saber qué es un componente de software. Un componente de software se define típicamente como algo que puede ser utilizado como una caja negra, en donde se tiene de manera externa una especificación general, la cual es independiente de la especificación interna. Como se puede apreciar en la Figura #2, la estructura del motor de GlassFish para jWebSocket está distribuida en diferentes componentes.

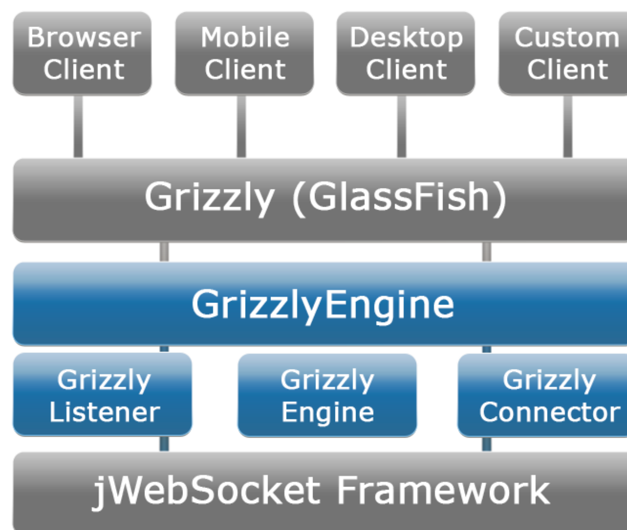


Figura # 2: Diagrama de componentes del motor de GlassFish para jWebSocket.

La arquitectura del motor cuenta con tres componentes fundamentales:

Grizzly (GlassFish): El servidor de GlassFish que es el encargado de procesar las peticiones de cualquier cliente, tanto HTTP como WebSocket.

GrizzlyEngine: El motor de GlassFish para jWebSocket, este se encarga de procesar los mensajes que llegan al servidor de GlassFish y enviarlos al marco de trabajo jWebSocket, así como enviar mensajes desde el marco de trabajo jWebSocket a través del servidor de GlassFish.

jWebSocket Framework: El marco de trabajo jWebSocket que implementa toda una lógica de negocio para desarrollar aplicaciones web basadas en WebSocket. Este se basa en el motor Grizzly para procesar y enviar los mensajes WebSocket.

2.9. Diseño con Metáforas

Debido a que SXP está basada en XP, y dicha metodología define un término llamado metáfora, lo cual según Martin Fowler es una historia compartida que describe como debería funcionar el sistema.

El Diseño con metáforas es sencillamente el diseño de la solución más simple que pueda funcionar y ser implementado en un momento dado del proyecto; lo cual genera el artefacto conocido como Modelo de Diseño, que a su vez está compuesto por un diagrama de paquetes como se puede apreciar en la Figura #3, donde se expone dicho diseño.

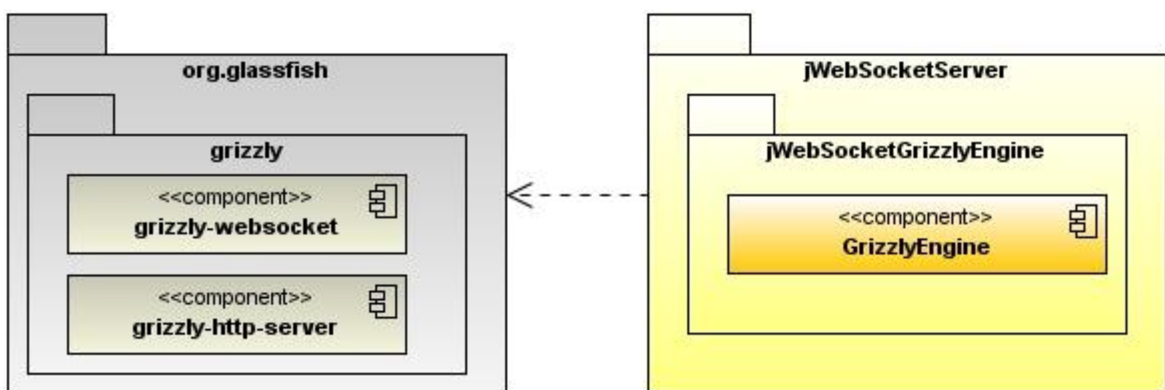


Figura #3: Diagrama de paquetes.

Para la realización del motor Grizzly, es necesario incluir sus dependencias, estas se encuentran en el paquete del Servidor de Aplicaciones GlassFish (***org.glassfish***), el paquete ***Grizzly*** cuenta con los componentes principales de este

marco de trabajo para lograr el correcto funcionamiento de la aplicación.

El servidor de jWebSocket se encuentra dentro del paquete ***jWebSocketServer***, dentro de él se encuentran las implementaciones de todos sus motores con sus respectivos nombres, por ejemplo, el motor Grizzly estará dentro del paquete ***jWebSocketGrizzlyEngine***.

2.10. Diagrama de Componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos de software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, entre otros. Para realizar la presente solución, se modeló el sistema en componentes como se muestra en la Figura #4 con el objetivo de lograr una mayor separación de las responsabilidades y realizar una mejor organización de la solución propuesta.

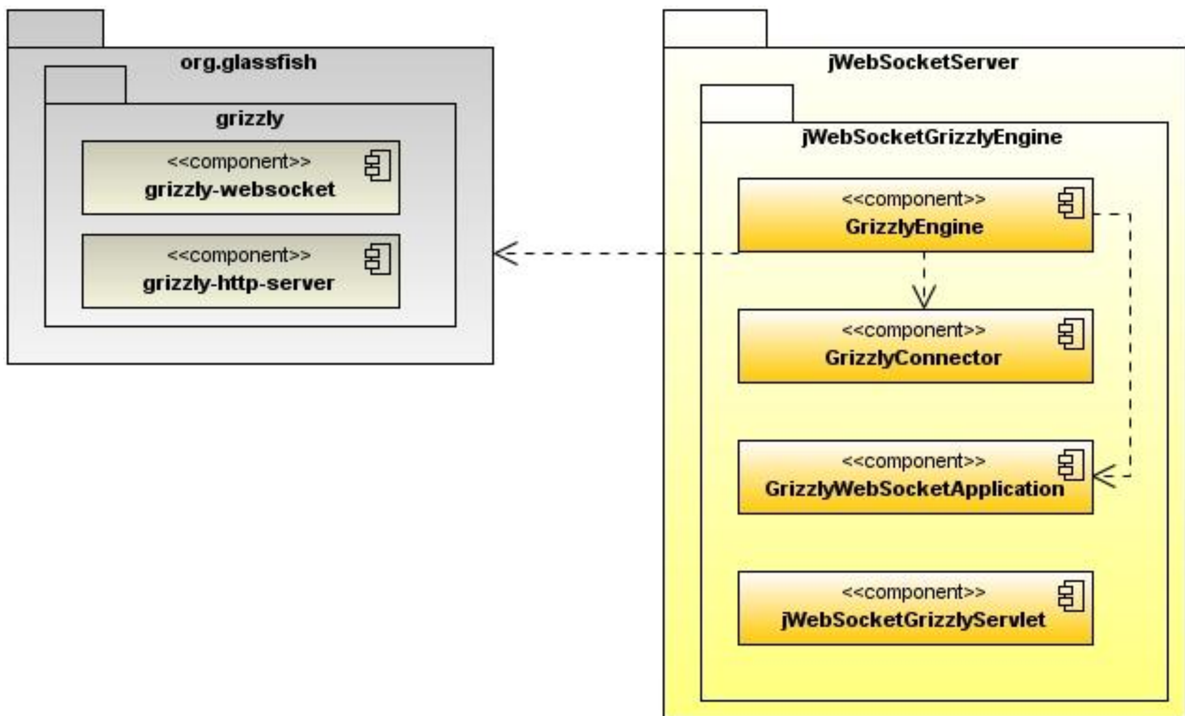


Figura # 4: Diagrama de componentes del motor Grizzly.

Para la creación del motor Grizzly para jWebSocket se utilizaron las librerías para crear servidores WebSocket y HTTP embebidas en el servidor de aplicaciones GlassFish. Por tanto, el paquete perteneciente a GlassFish contiene los

componentes de los que depende este motor. A a continuación se describen detalladamente las dependencias de Grizzly:

- **grizzly-websocket:** Contiene las librerías principales para establecer una comunicación WebSocket entre el cliente y el servidor de aplicaciones GlassFish.
- **grizzly-http-server:** Este componente permite crear un Servidor Web para publicar páginas web, generalmente utiliza el puerto 80 para lograr este objetivo.

Los componentes creados en el motor estarán agrupados en un solo paquete, denominado jWebSocketGrizzlyEngine, a continuación se describe brevemente la funcionalidad de cada uno de ellos:

- **GrizzlyEngine:** Este componente, como su nombre lo indica, es el Motor de Grizzly para jWebSocket, se encarga de crear los servidores HTTP y WebSocket, sirve páginas Web y conexiones WebSocket por el mismo puerto, es altamente configurable y permite tantos usuarios conectados como permita GlassFish.
- **GrizzlyConnector:** Representa una abstracción de alto nivel de la conexión que se realiza a un nivel muy bajo con los clientes, cada cliente representa un conector activo en el servidor y mantiene el flujo de vida hasta que el servidor lo elimine por algún motivo o el cliente cierre la conexión.
- **GrizzlyWebSocketApplication:** Este componente permite observar el comportamiento del servidor de GlassFish para recibir y enviar información a través del protocolo WebSocket. Este componente es utilizado por el motor para controlar todo el flujo de mensajes que entran al servidor GlassFish, procesarlos dentro de jWebSocket y enviar mensajes si es necesario.
- **jWebSocketGrizzlyServlet:** Este componente está compuesto por un grupo de aplicaciones de ejemplo basadas en WebSocket que son publicadas por el Servidor Web, una vez que se ejecuta el motor Grizzly para probar el funcionamiento de conexiones HTTP y WebSocket por el mismo puerto.

Conclusiones del capítulo

En este capítulo se definieron las características y funcionalidades principales del motor de Grizzly para jWebSocket que se pretende desarrollar. Quedaron aprobados los requisitos funcionales y no funcionales necesarios para obtener un motor eficiente, seguro y escalable. Se describieron las historias de usuario y tareas de ingeniería que se deben implementar así como el plan de releases para establecer el cronograma de trabajo. También se realizó un análisis de la arquitectura a utilizar y un modelado del diagrama de paquetes y de componentes que permiten una mejor compresión de la solución propuesta.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DEL SISTEMA

Introducción

En el presente capítulo se documenta la implementación de la solución propuesta en el capítulo anterior, generándose el diagrama de despliegue del sistema desarrollado. Además se exponen los casos de pruebas o test funcionales a los que fue sometido el motor en cada una de las iteraciones. El cumplimiento de estos casos de pruebas fue el hito para avanzar hacia la próxima iteración. En este capítulo además de las pruebas se dan a conocer los resultados obtenidos hasta el momento.

3.1. Diagrama de Despliegue

El diagrama de despliegue es un artefacto que modela la arquitectura en tiempo de ejecución de un sistema. En él se indica la situación física de los componentes lógicos desarrollados, lo que significa que sitúa el software en el hardware que lo contiene. A continuación se muestra el diagrama de despliegue, que representa la distribución física del sistema en términos de cómo se distribuirán las funcionalidades entre los nodos, donde cada nodo representa un recurso de cómputo, siendo estos procesadores o dispositivos hardware que se necesitan para el despliegue del sistema. El diagrama que se aprecia en la Figura #5 muestra los recursos necesarios para realizar el despliegue del motor de GlassFish para jWebSocket.

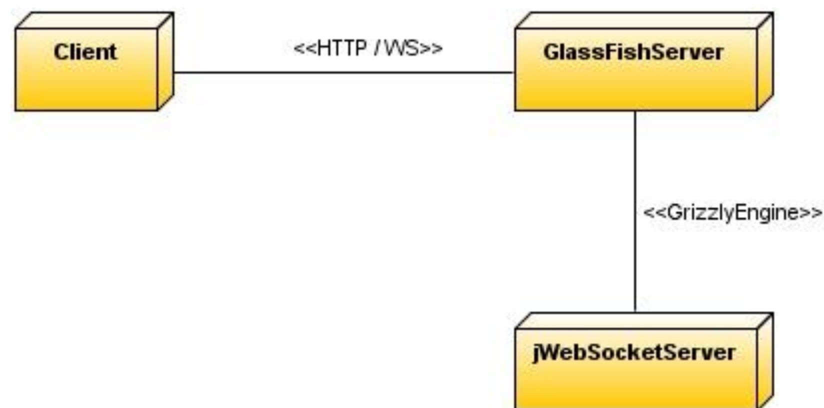


Figura #5: Diagrama de despliegue de la solución propuesta.

Se puede apreciar un Servidor de Aplicaciones GlassFish instalado en una PC servidora, esta brinda soporte de HTTP y WebSocket a múltiples clientes, una vez

que estos se conectan, pasan a ser clientes directos del servidor de jWebSocket a través del motor Grizzly. Dicho cliente puede ser un teléfono, una computadora o cualquier otro dispositivo que tenga soporte para realizar conexiones WebSocket o HTTP con un servidor en una red.

3.2. Estrategia de Validación

Para realizar las pruebas al motor de GlassFish para jWebSocket fue necesario dividir el ciclo de prueba en varias etapas. La primera etapa consistió en combinar mecanismos de prueba de caja negra y pruebas funcionales, entre una gran diversidad de pruebas de Caja Negra se utilizó la estrategia prueba de Arquitectura cliente/servidor. Para realizar este tipo de prueba, es necesario tener en cuenta las siguientes categorías:

- **Pruebas de servidor:** Permite probar las funciones de coordinación y manejo de datos del servidor. Desempeño del servidor (tiempo de respuesta y procesamiento total de los datos).
- **Pruebas de comunicación de red:** Se utiliza para verificar comunicación entre los nodos, el paso de mensajes, transacciones y tráfico de la red se realice sin errores.

A pesar de haber seleccionado una Arquitectura Basada en Componentes, se seleccionó una estrategia de pruebas basada en el modelo cliente-servidor porque para probar el correcto funcionamiento del motor de Grizzly, es necesario conectar una gran cantidad de usuarios al servidor de jWebSocket a través de GlassFish.

Otra de las etapas que se lleva a cabo es el proceso de certificación de calidad de software, realizado por el grupo de calidad del Centro de Desarrollo de la Facultad Regional "Mártires de Artemisa". Dicho grupo es el encargado de certificar la funcionalidad, estandarización y limpieza del código así como los artefactos documentales generados teniendo en cuenta la metodología de desarrollo SXP seleccionada en la presente investigación.

Por último, se realiza una valoración por parte del cliente que permite la evaluación del motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket. Comprobando de esta manera la capacidad de integración de la solución desarrollada con jWebSocket a nivel internacional, así como la usabilidad

real para los clientes potenciales y desarrolladores de esta comunidad.

A continuación se realiza una descripción más detallada de cada una de las etapas de la estrategia de validación trazada a las cuales fue sometido el motor desarrollado.

Casos de Pruebas

Las pruebas funcionales son definidas por el cliente y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al cliente. La utilización de estas, proporcionan grandes ventajas, permitiendo a los programadores medir la calidad de su trabajo y garantizar la entrega de un producto con calidad y en correspondencia con las necesidades del cliente.

Para lograr un mayor nivel de seguridad y usabilidad, se trazó como estrategia de validación realizar los casos de pruebas funcionales a una aplicación demostrativa, ya que esta permite probar las funcionalidades del motor de GlassFish para jWebSocket.

Debido a que las historias de usuario se encuentran altamente desglosadas y que estas a su vez no pueden ser probadas de manera independiente porque representan requerimientos aislados o de bajo nivel, se han agrupado para poder aplicarle las pruebas Funcionales.

La estrategia de pruebas seleccionada se apoya en la estrategia de pruebas de la Arquitectura Cliente-Servidor, debido a que las funcionalidades del motor de GlassFish deben ser probadas utilizando conexiones de múltiples clientes, de esta manera se estarían realizando comprobaciones del servidor y de la comunicación de red. Para la realización de las pruebas se utilizó del lado del cliente la herramienta Jasmine para JavaScript, esta permite definir una serie de casos de prueba y muestra los resultados obtenidos después de la ejecución de las mismas.

Tabla #28: Caso de prueba Funcionales (jws-01-01).

Caso de Prueba Funcionales	
Código Caso de Prueba: [jws-01-01]	**Nombre Historia de Usuario:** Crear conector de Grizzly-GlassFish.

Nombre de la persona que realiza la prueba: Victor Antonio Barzana Crespo.
Descripción de la Prueba: Esta prueba permite probar y validar que desde una aplicación cliente se pueden establecer y cerrar múltiples conexiones a través del motor Grizzly con el motor de jWebSocket y que este a su vez notifica a los escuchadores que ofrecerán soporte al cliente entrante.
Condiciones de Ejecución: El servidor Grizzly debe ser ejecutado previamente para que escuche conexiones entrantes, además, el servidor de jWebSocket debe estar inicializado con el motor de Grizzly-GlassFish. El cliente debe tener un mecanismo para establecer la conexión WebSocket, casi siempre es un cliente WebSocket cualquiera, tanto un navegador Web, como una aplicación Java o C#, desde un teléfono, o cualquier otro dispositivo con soporte WebSocket.
Entrada / Pasos de ejecución: 1- El cliente abre la aplicación de pruebas de jWebSocket: http://localhost:80/jWebSocketGrizzlyDemos/. 2- El cliente selecciona de la lista de demos, la aplicación de prueba del motor de jWebSocket, marca las opciones “Open connections for admin and guest” y “Load”, después manda a ejecutar la prueba haciendo clic en el botón Programmatic para que se realicen las pruebas automatizadas. 3- El Servidor de Aplicaciones GlassFish recibe una serie de conexiones concurrentes y notifica a sus escuchadores de clientes WebSocket conectados. 4- El motor Grizzly es notificado y automáticamente envía la información al servidor de jWebSocket. 5- El servidor de jWebSocket instancia un conector por cada cliente entrante para persistir la conexión a bajo nivel y queda establecida la conexión. 6- El cliente es notificado con un mensaje “Hello from jWebSocketServer”, el flujo de la aplicación de prueba muestra un resultado de ejecución satisfactorio. 7- El cliente cierra las conexiones automáticamente con el servidor de jWebSocket.

8- El servidor recibe la notificación de cierre de los conectores activos y cierra la conexión, borra las instancias de los conectores y notifica a las aplicaciones por cada conector que se cierre.
Resultado Esperado: El cliente recibe un mensaje de respuesta desde el servidor conteniendo un mensaje de tipo Welcome con la siguiente información "Hello from jWebSocketServer".
Evaluación de la Prueba: Satisfactoria

Tabla #29: Caso de prueba Funcionales (jws-01-02).

Caso de Prueba Funcionales	
Código Caso de Prueba: [jws-01-02]	**Nombre Historia de Usuario:** Crear conector de Grizzly-GlassFish.
Nombre de la persona que realiza la prueba: Victor Antonio Barzana Crespo.	
Descripción de la Prueba: Esta prueba permite probar y validar que desde una aplicación cliente se pueden enviar y recibir mensajes desde un cliente hasta el servidor de jWebSocket.	
Condiciones de Ejecución: El servidor Grizzly debe ser ejecutado previamente para que escuche conexiones entrantes, además, el servidor de jWebSocket debe estar inicializado con el motor de Grizzly-GlassFish. El cliente debe tener un mecanismo para establecer la conexión WebSocket, casi siempre es un cliente WebSocket cualquiera, tanto un navegador Web, como una aplicación Java o C#, desde un teléfono, o cualquier otro dispositivo con soporte WebSocket.	
Entrada / Pasos de ejecución: 1- El cliente selecciona la opción "Streaming PlugIn" en la aplicación de prueba y hace clic en ejecutar prueba. 2- El Servidor de Aplicaciones GlassFish recibe un mensaje del cliente con una petición de envío de cierta información. 3- El motor Grizzly es notificado y automáticamente envía la información al servidor de jWebSocket. 4- El servidor de jWebSocket procesa el mensaje entrante y comienza a	

enviar las respuestas correspondientes a la petición realizada. 5- El cliente recibe los mensajes enviados por el servidor de jWebSocket a través del motor de Grizzly.
Resultado Esperado: El cliente recibe una serie de mensajes con la información requerida por la aplicación de prueba. La aplicación muestra al usuario un resultado satisfactorio y la cantidad de pruebas ejecutadas.
Evaluación de la Prueba: Satisfactoria

Tabla #30: Caso de prueba Funcionales (jws-02-01).

Caso de Prueba Funcionales	
Código Caso de Prueba: [jws-02-01]	**Nombre Historia de Usuario:** Crear motor de Grizzly-GlassFish.
Nombre de la persona que realiza la prueba: Victor Antonio Barzana Crespo.	
Descripción de la Prueba: Esta prueba permite probar y validar que el servidor de jWebSocket se ejecuta correctamente. Y publique los demos de jWebSocket a través del puerto definido en la configuración. Para esto es necesario cargar la configuración del motor Grizzly, iniciar el motor de GlassFish, detener el motor de GlassFish y notificar a todos los conectores que se detuvo el motor.	
Condiciones de Ejecución: El servidor Grizzly debe ser ejecutado previamente para que escuche conexiones entrantes, además, el servidor de jWebSocket debe estar inicializado con el motor de Grizzly-GlassFish. El cliente debe tener un mecanismo para establecer la conexión WebSocket, casi siempre es un cliente WebSocket cualquiera, tanto un navegador Web, como una aplicación Java o C#, desde un teléfono, o cualquier otro dispositivo con soporte WebSocket.	
Entrada / Pasos de ejecución: 1- El desarrollador abre la aplicación servidora y la ejecuta. 2- El servidor carga la configuración de jWebSocket. 3- El servidor de jWebSocket arranca el motor Grizzly para GlassFish por el puerto especificado en la configuración del motor. 4- El motor Grizzly publica los demos de jWebSocket en la URL especificada en la configuración, por defecto es	

http://localhost:80/jWebSocketGrizzlyDemos/. 5- El motor Grizzly registra los escuchadores WebSocket del lado del servidor y ejecuta la aplicación GrizzlyWebSocketApplication que es la encargada de escuchar las conexiones WebSocket e instanciar los conectores. 6- El servidor muestra los logs de que se inicializó correctamente el servicio por los puertos especificados en la configuración. 7- El servidor de jWebSocket inicializa todas sus aplicaciones, componentes y filtros definidos en la configuración. 8- El motor Lanza el método engineStarted dentro de cada una de las apliaciones.
Resultado Esperado: El servidor de jWebSocket muestra un log de inicialización satisfactoria “jWebSocketServer startup complete”.
Evaluación de la Prueba: Satisfactoria

Tabla #31: Caso de prueba Funcionales (jws-03-01).

Caso de Prueba Funcionales	
Código Caso de Prueba: [jws-03-01]	**Nombre Historia de Usuario:** Integrar configuración
Nombre de la persona que realiza la prueba: Victor Antonio Barzana Crespo.	
Descripción de la Prueba: Esta prueba permite probar que la carga del fichero de configuración sea correcta, para esto se ejecutan pruebas a todos los componentes de jWebSocket para saber si el servidor funciona correctamente y no existe fallo de carga en ningún fichero, a esta prueba también se le denomina prueba de carga y rendimiento del servidor de jWebSocket utilizando el motor Grizzly.	
Condiciones de Ejecución: El servidor Grizzly debe ser ejecutado previamente para que escuche conexiones entrantes, además, el servidor de jWebSocket debe estar inicializado con el motor de Grizzly-GlassFish. El cliente debe tener un mecanismo para establecer la conexión WebSocket, casi siempre es un cliente WebSocket cualquiera, tanto un navegador Web, como una aplicación Java o C#, desde un teléfono, o cualquier otro dispositivo con soporte	

WebSocket.
Entrada / Pasos de ejecución: 1- El servidor inicia utilizando el motor de Grizzly. 2- El fichero de configuración es cargado correctamente por el motor y no aparece ningún log en la consola de errores. 3- Se instancian todas las aplicaciones, filtros y escuchadores definidos en el fichero de configuración. 4- El servidor muestra un mensaje de “jWebSocketServer startup complete”. 5- El cliente accede a la aplicación de pruebas con un navegador web o cualquier cliente WebSocket. 6- El cliente selecciona todas las opciones de la lista, teniendo en cuenta que cada una de ellas representa una aplicación que debió ser cargada correctamente en la configuración. 7- El cliente comienza a ejecutar las pruebas haciendo clic en el botón Programmatic y calcula el tiempo de inicio de las pruebas. 8- El servidor ejecuta y procesa todas las conexiones y las pruebas de rendimiento realizadas a petición del cliente. 9- El servidor envía mensajes de respuesta, satisfactorios por cada prueba ejecutada al cliente. 10- El cliente calcula el tiempo de respuesta de la aplicación y lo muestra al usuario. 11- El cliente muestra un estado de la ejecución de las pruebas al usuario, rellenando en color verde las pruebas satisfactorias y en rojo las pruebas insatisfactorias.
Resultado Esperado: El cliente recibe una serie de mensajes con la información requerida por la aplicación de prueba. La aplicación muestra al usuario un resultado satisfactorio y la cantidad de pruebas ejecutadas.
Evaluación de la Prueba: Satisfactoria

Tabla #32: Caso de prueba Funcionales (jws-04-01).

Caso de Prueba Funcionales

<table>
<tr><td>Código Caso de Prueba: [jws-04-01]</td><td>Nombre Historia de Usuario: Publicar los demos de jWebSocket.</td></tr>
<tr><td colspan="2">Nombre de la persona que realiza la prueba: Victor Antonio Barzana Crespo.</td></tr>
<tr><td colspan="2">Descripción de la Prueba: Esta prueba permite probar el correcto funcionamiento de los demos de jWebSocket que hayan sido previamente publicados por el Servidor Web instanciado por el motor de jWebSocket.</td></tr>
<tr><td colspan="2">Condiciones de Ejecución: El servidor Grizzly debe ser ejecutado previamente para que escuche conexiones entrantes, además, el servidor de jWebSocket debe estar inicializado con el motor de Grizzly-GlassFish.
El cliente debe tener un mecanismo para establecer la conexión WebSocket, casi siempre es un cliente WebSocket cualquiera, tanto un navegador Web, como una aplicación Java o C#, desde un teléfono, o cualquier otro dispositivo con soporte WebSocket.</td></tr>
<tr><td colspan="2">Entrada / Pasos de ejecución:
1- El servidor de jWebSocket inicia utilizando el motor de Grizzly.
2- El fichero de configuración es cargado correctamente por el motor y no aparece ningún log en la consola de errores.
3- El motor Grizzly publica el Servidor Web para brindar contenido HTTP por el puerto definido en la configuración
4- El motor publica los demos de jWebSocket en la URL definida por la configuración del servidor.
5- Se instancian todas las aplicaciones, filtros y escuchadores definidos en el fichero de configuración.
6- El servidor muestra un mensaje de “Grizzly server started at host 127.0.0.1 and port 80”.
7- El cliente accede a la url donde se encuentran publicados los demos de jWebSocket para comprobar el correcto funcionamiento.
8- El cliente accede a la página de pruebas para ejecutar las pruebas de jWebSocket.
9- El La ejecución de las pruebas es satisfactoria, establece conexiones y brinda servicio WebSocket por el mismo puerto 80.</td></tr>
<tr><td colspan="2">Resultado Esperado: El cliente accede a la URL donde están publicados los</td></tr>
</table>

demos de jWebSocket y establece una conexión WebSocket y HTTP por el mismo puerto. La aplicación de pruebas muestra al usuario resultados satisfactorios.
Evaluación de la Prueba: Satisfactoria

Certificación de Calidad de Software

El grupo de calidad del Centro de Desarrollo de la Facultad Regional “Mártires de Artemisa” es el encargado de certificar la calidad de los artefactos documentales generados teniendo en cuenta la metodología de desarrollo SXP seleccionada en la presente investigación. Este proceso fue llevado a cabo por la asesora de calidad del Centro, Ing. Maidel Ojeda Castro, teniendo en cuenta los siguientes artefactos:

- Plantilla Modelo de Historia de Usuario del Negocio.
- Plantilla lista de reserva del producto (LRP).
- Plantilla de Historia de Usuario.
- Plantilla de Arquitectura de Software SXP.
- Plantilla Tarea de Ingeniería.
- Plantilla de Releases.
- Plantilla Estándar de Código.
- Plantilla Caso de Prueba Funcionales.
- Plantilla Manual de Usuario.
- Plantilla Manual de Instalación.
- Plantilla Manual de Desarrollo.

Por otra parte, se puso a disposición del Ing. Domma Moreno Dager, asesor de tecnología del Centro, el código fuente del motor desarrollado. Este fue sometido a un proceso de revisión donde se hizo énfasis en la calidad, estandarización y limpieza del código así como en el correcto funcionamiento de la librería teniendo en cuenta los requisitos funcionales definidos por el cliente.

Valoración del cliente

Con el objetivo de comprobar el funcionamiento del motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket, la presente investigación fue sometida al criterio de Alexander Schulze, líder internacional de la comunidad jWebSocket y arquitecto principal del proyecto. Para esto fueron incorporados al repositorio internacional del proyecto jWebSocket el código fuente y los manuales de Desarrollador y Administración en idioma inglés que garantizan un mayor entendimiento del motor. De esta forma el cliente certifica la capacidad de integración de la solución desarrollada con el marco de trabajo jWebSocket a nivel internacional, así como la usabilidad real para los clientes potenciales y desarrolladores de esta comunidad. En la realización de este proceso fueron señaladas algunas imprecisiones y dificultades al motor, las cuales fueron solucionadas y revisadas nuevamente por el cliente. Finalmente fue emitida su certificación de calidad.

3.3. Resultados Obtenidos

El proceso de validación al cual fue sometido el motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket presentó los siguientes resultados:

- Los casos de pruebas Funcionales realizados fueron satisfactorios, garantizando el correcto funcionamiento de la librería desarrollada y el cumplimiento de los requisitos funcionales definidos por el cliente.
- El grupo de calidad del Centro de Desarrollo de la Facultad Regional “Mártires de Artemisa” emitió un aval que certifica la funcionalidad y estandarización de la librería desarrollada así como la calidad de la documentación que permite a la solución ser usable y generalizada para otros usuarios.
- La valoración del cliente certificó satisfactoriamente el motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket, asegurando de esta forma la correcta integración de esta solución al proyecto internacional jWebSocket.

Por el resultado satisfactorio obtenido en todos los casos de pruebas realizados al motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket, queda disponible en la versión 1.0 Beta de este Marco de Trabajo. Se obtuvo un sistema que cumple con todas las especificaciones para permitir extender las funcionalidades del servidor de jWebSocket con un motor altamente configurable y robusto basado en el Marco de Trabajo Grizzly, en el **Anexo 1** podrá apreciar una muestra de la aplicación que ejecuta las pruebas mostrando los resultados satisfactorios.

3.4. Funcionalidades Obtenidas

Entre las principales funcionales que posee el motor de Grizzly para jWebSocket en su versión 1.0 se pueden encontrar:

- ✓ Permite conectar y desconectar cualquier cliente al servidor de jWebSocket.
- ✓ Permite enviar paquetes WebSocket a una ultra alta velocidad, del cliente hacia el servidor y viceversa.
- ✓ Permite a jWebSocket procesar paquetes entrantes desde cualquier cliente conectado al Servidor de Aplicaciones GlassFish.
- ✓ Permite procesar paquetes fragmentados y comprobaciones del estado de la conexión (Ping/Pong).
- ✓ Permite publicar un Servidor Web para servir páginas web y conexiones WebSocket por el mismo puerto.
- ✓ Permite aprovechar los puertos 80 y 443 que son seguros y estándares para servidores proxy.
- ✓ Incluye demos de prueba que se publican automáticamente por el Servidor Web, una vez que se ejecuta el servidor de jWebSocket.

3.5. Aporte Social y Económico

El motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket ya se encuentra funcionando y disponible para su descarga desde el sitio oficial de jWebSocket, se encuentra a prueba en esta primera versión, sin embargo, para el desarrollo de aplicaciones Web empresariales representa un innovación tecnológica que aumenta la productividad y eficiencia de las empresas o equipos de desarrollo que utilizan el marco de trabajo jWebSocket para desarrollar

sus aplicaciones. El uso de este motor, permite entre otras ventajas, la fácil migración de aplicaciones Web empresariales creadas sobre el Servidor de Aplicaciones GlassFish que deseen utilizar el marco de trabajo jWebSocket sin tener que realizar cambios en la lógica de sus aplicaciones.

El ahorro de tiempo y mejores y eficientes prácticas de desarrollo se traducen directamente en impactos económicos positivos para las empresas.

En el ámbito social, el motor Grizzly se incluye de forma nativa dentro del marco de trabajo jWebSocket, el cual posee licencia de software libre, permitiendo así que este motor pueda ser utilizado sin costo alguno por los desarrolladores interesados. Esta herramienta que hoy se encuentra disponible para su descarga dentro de las fuentes del servidor de jWebSocket ha sido desarrollada en la UCI y por tanto la universidad con toda seguridad recibirá beneficios de brindar soporte técnico directo a otras empresas y capacitación de los estudiantes, profesores y trabajadores de esta institución.

Conclusiones del capítulo

En este capítulo se realizó el diagrama de despliegue que indica la situación física de los componentes lógicos desarrollados. Además se realizaron casos de pruebas que confirmaron la confiabilidad y la calidad del motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket. De esta manera los casos de prueba mostraron que las funciones del motor son operativas y que se produce un resultado correcto. Demostrándose así que el motor se encuentra en óptimas condiciones para ser integrado al marco de trabajo jWebSocket.

CONCLUSIONES

Con la realización de la presente investigación se dio respuesta a las preguntas científicas planteadas, obteniéndose de manera general las siguientes conclusiones:

- ✓ La realización de la fundamentación teórico-metodológica permitió conceptualizar el proceso de integración de servidores de aplicaciones Java, específicamente en el marco de trabajo jWebSocket.
- ✓ Actualmente jWebSocket considera todas las implementaciones WebSocket del lado del servidor como posibles integraciones, aprovechando de ellas todas sus ventajas.
- ✓ Se desarrolló un motor de GlassFish para jWebSocket utilizando la librería Grizzly-WebSockets que permite el intercambio de paquetes a bajo nivel usando el protocolo WebSocket. Esto permite una mayor seguridad y usabilidad en las aplicaciones WebSocket que hoy se desarrollan con jWebSocket.
- ✓ Se comprobó la capacidad y potencialidad del motor desarrollado para el marco de trabajo jWebSocket a través de la estrategia de validación trazada para la presente investigación, garantizando que su código fuente y documentación poseen la calidad requerida, además puede ser integrado al entorno de desarrollo de jWebSocket y utilizado por su comunidad.

RECOMENDACIONES

Para versiones posteriores del motor para la integración del servidor GlassFish en el marco de trabajo jWebSocket se recomienda:

- Crear una configuración adicional para que al ejecutar el motor se publique por cada componente un sitio web correspondiente al mismo.
- Garantizar la integración del motor con la nueva herramienta de despliegue Grizzly Deployer para garantizar que se pueda publicar y desplegar cualquier contenido web referente a la tecnología Java EE.

BIBLIOGRAFÍA

1. **Arcand, Jeanfrancois. 2007.** weblogs.java.net. [En línea] 2007. http://weblogs.java.net/blog/jfarcand/archive/20070207_Grizzly.pdf.
2. **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato. 2004.** Control de versiones con Subversion. [En línea] 2004. [Citado el: 22 de 02 de 2012.] http://svnbook.red-bean.com/nightly/es/svn-ch-1-sect-1.html.
3. **Eguíluz Pérez, Javier. 2009.** *Introducción a XHTML.* 2009.
4. **Figueroa, Roberth G, Solís, Camilo J y Cabrera, Armando A. 2011.** Entorno Virtual de Aprendizaje. [En línea] 2011. [Citado el: 13 de 12 de 2011.] http://eva.uci.cu/mod/resource/view.php?id=9303&subdir=/Metodologias/RUP .
5. *Framework Approach for WebSockets.* **Schulze, Alexander. 2011.** Web Technologies & Internet Applications (WebTech 2011) : http://dl.globalstf.org/index.php?page=shop.product_details&flypage=flypage_images.tpl&product_id=528&category_id=42&option=com_virtuemart&Itemid=4&vmcchk=1&Itemid=4, 2011.
6. **Gosling, James, y otros. 2005.** *The Java language specification.* s.l. : Addison Wesley, 2005. 3ra Edition.
7. **INSTITUTO NACIONAL DE ESTADISTICA E INFORMATICA, PERU. 1999.** Herramientas CASE. [En línea] 1999. http://www.inei.gob.pe/biblioineipub/bancopub/Inf/Lib5103/Libro.pdf.
8. *jWebSocket instead of XHR and Comet.* **Schulze, Alexander. 2010.** 06 de 2010, Mobile Developer. http://jwebsocket.org/press/xhr_insteadof_comet.htm.
9. **Kirkpatrick, Marshall. 2009.** Read Write Web. [En línea] 22 de 9 de 2009. [Citado el: 21 de 02 de 2012.] http://www.readwriteweb.com/archives/explaining_the_real-time_web_in_100_words_or_less.php.

10. **Moussine-Pouchkine, Alexis, Pelegri-Llopart, Eduardo y Yoshida, Yutaka. 2007.** *The GlassFish Community Delivering a JavaEE Application Server.* 2007. http://www.slideshare.net/alexismp/glassfish-article-september-07.
11. **NetBeans. 2011.** *NetBeans.* [En línea] Oracle Corporation, 2011. http://netbeans.org/features/index.html.
12. *Paradigmas de la Programación: JavaScript y Python.* **Ana Lilia, Careaga Mercadillo. Agosto 2010.** s.l. : Instituto Tecnológico de Teléfonos de México S.C., Agosto 2010.
13. **RapidSVN. 2011.** RapidSVN. [En línea] 2011. [Citado el: 13 de 12 de 2011.] http://www.rapidsvn.org/.
14. **Riola, Jose Carlos Carvajal. 2008.** *HERRAMIENTAS Y MODELO DE DESARROLLO PARA APLICACIONES.* 2008. Tesis de Maestría. http://upcommons.upc.edu/pfc/bitstream/2099.1/5608/1/50015.pdf.
15. **Rumbaugh, James y Jacobson, Ivar. 2007.** *El Lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : ADDISON-WESLEY, 2007. 9788478290871.
16. **Schulze, Alexander. 2012.** *WebSockets- Boosting Web Communication.* s.l. : Addison Wesley Verlag, 2012. http://books.google.es/books?id=PUvUXwAACAAJ&dq=alexander+schulze+websocket&hl=es&sa=X&ei=hklUT7eaGuHd0QHVr7z1DQ&ved=0CDIQ6AEwAA.
17. **Schwaber, Ken y Beedle, Mike. 2011.** *Agile Software Development with SCRUM.* s.l. : Prentice Hall, 2011. 0130676349.
18. *SXP, Metodología Ágil para el Desarrollo de Software.* **Peñalver, G, Meneses, A y García, S. 2010.** Chile : 1er congreso Iberoamericano de Ingeniería de Proyectos, 2010.
19. **Templ, Josef. 2003.** *The KITE Application Server Architecture.* 2003. Tesis. http://www.software-templ.com/papers/KiteArchitecture.pdf.
20. **2011**The WebSocket protocol*The WebSocket*

*protocol*http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17

21. **VP, Visual Paradigm. 2012.** Visual Paradigm for UML. [En línea] 2012. [Citado el: 22 de 02 de 2012.] http://www.visual-paradigm.com/product/vpuml/.

22. **Wells, Don. 2009.** Extreme Programming: A gentle introduction. *Extreme Programming.* [En línea] 28 de 07 de 2009. [Citado el: 16 de 02 de 2012.] http://www.extremeprogramming.org/.

ANEXOS

Anexo 1:

Resultado satisfactorio de la ejecución de pruebas al motor de Grizzly para jWebSocket utilizando el Marco de Trabajo para pruebas desde Javascript Jasmine.

jWebSocket - Test and Benchmark - Mozilla Firefox

File Edit View History Bookmarks Tools Help

jWebSocket - Test and Benchmark

localhost/germanyClient/test/runTests.htm?{"action":"runFullTestSuite","args":{"openConns":true,"streamingPlugIn":true,"channelsPlugIn":true,"rpcPlu

RPC PlugIn
Automated API PlugIn
File System PlugIn
System PlugIn
Load
Close connections for admin and guest
IOC Container
Events Sample Application

Programmatic Benchmark

Jasmine 1.0.2 revision 1298837858 Show passed skipped

67 specs, 0 failures in 6.912s Finished at Sat Apr 07 2012 12:42:32 GMT-0400 (CDT) run all

jWebSocket Test Suite run
Performing test suite: jws.tests.load... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
jws.tests.load: Trying to establish 20 concurrent connections... run
Performing test suite: jws.tests.system... run

GLOSARIO DE TÉRMINOS

---

[1] ***HTTP:*** El propósito del protocolo HTTP es permitir la transferencia de archivos (principalmente, en formato HTML) entre un navegador (el cliente) y un servidor web localizado mediante una cadena de caracteres denominada dirección URL.
[2] ***XHR*:** Un objeto XMLHttpRequest es una instancia de una API que nos permite la transferencia de datos en formato XML desde cualquier lenguaje del navegador (JavaScript, JScrip, VBScript) a los del servidor (PHP, Perl, ASp, Java) e inversamente.
[3] ***Ajax:*** Acrónimo de JavaScript y XML asíncronos. Técnica de desarrollo web para crear aplicaciones interactivas mediante la combinación de tecnologías ya existentes tales como: HTML/XHTML, CSS, DOM y JavaScript.
[4] ***Comet:*** Con el término Comet se busca es describir el intercambio de información existente entre un cliente y un servidor donde este último sea quien inicie la comunicación. Otro término que describe este mecanismo es "Reverse Ajax" (Ajax Reverso ó Inverso).
[5] ***JavaScript*:** Javascript es un lenguaje con muchas posibilidades, utilizado para crear pequeños programas que luego son insertados en una página web y en programas más grandes, orientados a objetos mucho más complejos.
[6] ***WebSocket:*** Permite que las aplicaciones web cliente puedan abrir desde JavaScript conexiones de red bidireccionales con cualquier servidor mediante un protocolo de red. La W3C se encarga de definir la API para JavaScript y la IETF de elaborar el protocolo de red.
[7] ***Full-duplex:*** Cualidad de los elementos que permiten la entrada y salida de datos de forma simultánea.
[8] ***TCP:*** El fin de TCP es proveer un flujo de bytes confiable de extremo a extremo sobre una internet no confiable. TCP puede adaptarse dinámicamente a las propiedades de la internet y manejar fallas de muchas clases.
[9] ***Handshake*:** Es un término del idioma inglés que se refiere a la conversación protocolar que realizan los sistemas en arquitecturas cliente/servidor antes de establecer una conexión.
[10] ***Flash*:** Es una tecnología para crear animaciones gráficas vectoriales independientes del navegador y que necesitan poco ancho de banda para mostrarse en los sitios web.
[11] ***Servlets:*** Los servlets son programas que funcionan como los CGIs convencionales atendiendo peticiones de un cliente teniendo al servidor como el encargado, pero escritos en Java y con la ventaja de explotar todas las bondades de java
[12] ***Java Server Pages (JSP):*** *JSP es un acrónimo de Java Server Pages (Páginas de Servidor Java). Es una tecnología orientada a crear páginas web con programación en Java.*
[13] ***Enterprise Java Beans (EJB):*** *Un "Java Bean" es un componente utilizado en Java que permite agrupar funcionalidades para formar parte de una aplicación.*
[14] **WebSphere:** Servidor de aplicaciones de IBM.
[15] ***Oracle Application Server:*** Servidor de aplicaciones de código abierto de Oracle.
[16] ***GlassFish*:** Servidor de aplicaciones de código abierto de Oracle.
[17] ***API:*** Application Programming Interface (Interfaz de programación de aplicaciones), constituye un conjunto de rutinas, procedimientos, funciones y herramientas que una determinada biblioteca pone a disposición para que sean utilizados por otro software como una capa de abstracción.
[18] ***Netty:*** es un marco de trabajo cliente-servidor basado en NIO que permite el desarrollo fácil y rápido de aplicaciones en red tales como servidores de protocolo cliente servidor.
[19] ***Jetty:*** Provee un servidor HTTP, un cliente HTTP y un Contenedor de Servlet.
[20] ***Java New I/O API (NIO):*** *New I/O, usualmente llamado NIO, es una colección de APIs Java Programming Language que ofrecen ventajas para operaciones I/O.*
[21] ***Apache*:** Servidor Web de código abierto para proveer servicios HTTP.
[22] ***JXTA*:** (Juxtapose) es una plataforma (peer-to-peer) de código abierto creada por Sun Microsystems en el año 2001. Está definida como un conjunto de protocolos basados en XML.
[23] ***Sailfin*:** es un proyecto que añade nuevas funcionalidades, como el servlet de Session Initiation Protocol (SIP).
[24] ***Jersey:*** Sub proyecto de GlassFish para medir la calidad de producción, JAX-RS (JSR 311).
[25] ***RestLet:*** Marco de trabajo de código abierto usado por Java SE, Java EE, OSGi, Android y GWT.