Universidad de las Ciencias Informáticas

Facultad 6

Título: Solución Informática para el Centro Nacional de Balance Alimentario: Implementación de un componente de software para la transferencia de datos a través de Internet.

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

Autores: Lissete González Gallo
Reynaldo Alvarez Luna

Tutores: Msc. Yanet Villanueva Armenteros
Msc. Longendri Aguilera Mendoza
Ing. Ana Lupe Delgado Montero

Consultante: Ing. Andrés Ballester Marsal

Mayo, 2008

"Si quieres comprender la palabra felicidad, tienes que entenderla como recompensa y no como fin."

Antoine de Saint-Exúpery

DECLARACIÓN DE AUTORÍA

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

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

Reynaldo Alvarez Luna

_____

Firma del autor

Lissete Gonzalez Gallo

_____

Firma del autor

Msc. Yanet Villanueva Armenteros

_____

Firma del tutor

Msc. Longendri Aguilera Mendoza

_____

Firma del tutor

Ing. Ana Lupe Delgado Montero

_____

Firma del tutor

DATOS DE CONTACTO

Tutores:
Msc. Yanet Villanueva Armenteros
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: villanueva@uci.cu

Ing. Ana Lupe Delgado Montero
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: aldelgado@uci.cu

Msc. Longendri Aguilera Mendoza
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: loge@uci.cu

Consultante:
Ing. Andrés Ballester Marsal
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: aballester@uci.cu

AGRADECIMIENTOS

A Fidel y la Revolución por la creación de esta Universidad…

A todas las personas que han contribuido con nuestra formación profesional y para la vida, profesores, amigos y familia, les agradecemos sinceramente, con el mayor deseo de estar a la altura de las enseñanzas y principios que compartieron con nosotros.

A Longendri, Ana Lupe, Yanet y a los profesores del proyecto, que nos ayudaron mucho profesionalmente con sus críticas y consejos constantes.

A Tania por su revisiones ortográficas, por ser tan compresiva y paciente.

A Martica, Yamila y Mari, nuestras analistas….

A los buenos muchachos del proyecto, que los admiro, en especial a Yosvany.

A la FEU, que me ha enseñado lo que no se aprende en el aula…

DEDICATORIA

A mis padres, que han dado siempre lo mejor por mi futuro…
A mi niña que es mi inspiración constante… te amo
A mi abuela del alma, de la que siempre seré su ombligo…
A mi abuelo que siempre confió en este momento, donde quiera que esté…
A mi hermana, que le deseo lo mejor de la vida…
A toda mi familia, que siempre me han dado ese apoyo necesario… a todos.
A Rosado que ha sido y será mi hermano y un poco mi consejero…
A mis amigos que tan especialmente saben estar cuando se necesitan…Peña, Eslavy y Pedro.
A mis profesores y compañeros de la FEU.

Reynaldo

A mi mamita y mi papito lindo, a quienes adoro y por siempre serán mi vida….
A mi hermano, a quien siempre tengo presente y quiero mucho…
A mi nene lindo Fily, que es mi amor y mi felicidad…
A mi suegrito, por brindarme siempre su apoyo.
A mis tías, primas y en general a mi familia por preocuparse por mi y brindarme su apoyo…
A mis amigas preciosas, Moniquilla, Lieny, Sally, Aiditrini, Yadira y Aliekna, para quienes guardo un lugarcito en mi corazón.
A mi compañero de tesis Rey, a quien siempre recordaré por ayudarme tanto y aconsejarme.
A todas mis amistades por estar allí cuando los necesito, en especial Maikel y Yadir.
A mis profesores y compañeros a lo largo de todos estos años.

Lissete

RESUMEN

La presente investigación surge en el marco de trabajo del Proyecto: “Diseño Organizacional y de Solución Informática para la captura, procesamiento, análisis y validación de los indicadores productivos y de gestión de la Misión Alimentación para la toma de decisiones en el marco del Centro Nacional de Balance Alimentario” aprobado en la Séptima Comisión Mixta Cuba-Venezuela del Convenio de colaboración Cuba-Venezuela; más conocido como “Solución Informática para el Centro Nacional de Balance Alimentario”. En esta se realizará la implementación de un componente de software para la transferencia de datos a través de internet entre aplicaciones que siguen una arquitectura Cliente-Servidor. La seguridad estará garantizada mediante el uso de técnicas de cifrado y túneles seguros. Su utilización, abstrae a los desarrolladores de las aplicaciones clientes y servidor del conocimiento de las tecnologías utilizadas en la transferencia de datos, brindándole un procedimiento sencillo para la integración en sus aplicaciones.

PALABRAS CLAVE

Componente, sistemas distribuidos, transferencia de datos, seguridad, cliente, servidor, CNBA

TABLA DE CONTENIDOS

TABLA DE ILUSTRACIONES

INTRODUCCIÓN

En la República Bolivariana de Venezuela se han implementado numerosos proyectos y reformas en función de alcanzar la seguridad alimentaria del pueblo, siguiendo el precepto de la Organización para la Alimentación y la Agricultura (FAO), que plantea que: "Existe seguridad alimentaria cuando todas las personas tienen en todo momento acceso físico y económico a suficientes alimentos inocuos y nutritivos para satisfacer sus necesidades alimentarias". [1]

La seguridad alimentaria implica el cumplimiento de las siguientes condiciones:

- Una oferta y disponibilidad de alimentos adecuada.
- La estabilidad de la oferta sin fluctuaciones ni escasez en función de la estación del año.
- El acceso a los alimentos o la capacidad para adquirirlos.
- La buena calidad e inocuidad de los alimentos.

Es decir, las políticas gubernamentales, las medidas de control, los procesos que se siguen, pretenden alcanzar el que todo alimento que llega al consumidor, sea un alimento "seguro", libre de contaminaciones que supongan una amenaza para la salud. [2]

En la República Bolivariana de Venezuela, alcanzar la seguridad alimentaria ha implicado un arduo trabajo. La Misión más sensible creada por el Gobierno Revolucionario creció, dejó de ser Misión Mercal para convertirse en Misión Alimentación. La Misión Mercal se encargaba solo de llevar el alimento a la población; y la Misión Alimentación le incorporó a ésta, el desarrollo de programas para reordenar los hábitos alimenticios del ciudadano venezolano propiciando una alimentación balanceada, adaptada a las costumbres de cada región, avanzando hacia la disminución en los índices de subnutrición.

En el 2006 el Ministerio de la Alimentación propone el surgimiento del Centro Nacional de Balance Alimentario (CNBA) como la estrategia bandera de la Misión Alimentación; cuyo objetivo sería integrar todos los organismos que tuvieran estrecha relación con el tema de la alimentación. Actualmente el CNBA sigue integrando estos ministerios y entre los objetivos fundamentales planteados se encuentran: garantizar el consumo equilibrado de alimentos de acuerdo a las necesidades nutricionales de la población y lograr el autoabastecimiento territorial disminuyendo progresivamente las importaciones de alimentos e insumos.

El Comandante Hugo Rafael Chávez Frías el 9 de enero del 2008 evidenció la importancia que tiene el CNBA para lograr la seguridad alimentaria en la nación cuando planteaba en una de sus reflexiones “que aunque existan fallas en la producción, en la importación, en el sistema de almacenaje, y en la distribución de alimentos es muy importante la planificación que se haga”. [3]

El CNBA se encarga de generar balances nacionales y alertas tempranas para garantizar el abastecimiento, conjugando producción, inventarios, importaciones, exportaciones y niveles de reserva. Actualmente el CNBA avanza en la automatización de sus procesos y se ha producido la necesidad de una solución tecnológica que permita la recolección de la información proveniente de los distintos entes involucrados.

Los entes se clasifican en privados (empresas, organizaciones, asociaciones, cooperativas con tipo de propiedad privada) y gubernamentales (ministerios con sus dependencias).

Los entes privados suministran a los analistas del CNBA:

- información de Producción Nacional
- información de Importaciones-Exportaciones
- información de Inventarios

Los entes gubernamentales suministran a los analistas del CNBA:

- información de Producción Nacional
- información de Acceso Económico
- información de Permisología
- información de Importaciones-Exportaciones
- información de Consumo Social Priorizado
- información de Inventarios

La información suministrada por los entes involucrados se encuentra en formato duro o digital, y es enviada a los analistas del CNBA vía correo electrónico o postal, o por entrega formal. La mayoría de los entes no disponen de sistemas para generar la información requerida por el CNBA, por lo que generalmente la información suministrada no satisface la información requerida; se envían datos innecesarios o no se envían algunos de los necesarios. Los analistas del CNBA precisan buscar la

información por vías donde no está asegurada la veracidad de los datos y realizar un conjunto de transformaciones, lo que puede provocar que los análisis no sean lo suficientemente profundos. Vale destacar que actualmente este proceso se realiza de forma manual y genera gran volumen de información que debe estar disponible en cualquier momento para realizar análisis estadísticos más precisos; y teniendo en cuenta que parte de los análisis que se realizan en el marco del CNBA utilizan como patrón de referencia datos pasados, se hace necesario disponer de un medio de almacenamiento con capacidad suficiente que permita mantener un historial de los datos.

Otra situación desfavorable es la protección inadecuada de las vías de entrega de la información. Los entes gubernamentales y privados que reportan información al CNBA tienen acceso a Internet y utilizan esta vía para la entrega de la información a través de correo electrónico.

Para la solución del problema de la recogida de la información de los diferentes entes, en la concepción del Proyecto Técnico, y en el análisis de la Solución Informática del CNBA, se ha definido el desarrollo de una aplicación de escritorio que envía la información a través de internet al servidor del CNBA, la que es procesada por la aplicación de integración y almacenada en la base de datos (repositorio central), siguiendo una arquitectura de tres capas como se muestra en la Fig. 1

Fig. 1 Distribución lógica de la Solución del CNBA

La aplicación de escritorio estará ubicada en cada uno de los entes responsables de suministrar los datos al CNBA que tengan la información digitalizada. La misma se encargará de extraer los datos relevantes, transformarlos al formato establecido y enviarlos a la Capa Media. Esta Aplicación de Escritorio es considerada como un cliente pesado ya que tiene parte de la lógica de negocio y realiza parte del procesamiento de los datos. En la Capa Media, ubicada en el CNBA, existirá una Aplicación de Integración donde estará centralizada la lógica de negocio, para validar el formato de los datos recibidos y almacenar a los mismos en el repositorio central, también ubicado en el CNBA.
La protección de la información digitalizada que es enviada a los analistas del CNBA procedentes de

los diferentes entes, es clave dentro de la Solución Informática del CNBA, por ello se plantea como **problema de la investigación:** ¿Cómo lograr la transferencia segura de datos entre las aplicaciones de escritorio y la aplicación de integración del CNBA?

El **objeto de estudio** es el proceso de transferencia de datos a través de internet; y el **campo de acción**, las tecnologías para la comunicación entre aplicaciones de forma segura a través de Internet.

Para dar solución al problema científico declarado, esta investigación tiene como **objetivo general** implementar un componente de software para la transferencia segura de datos entre la aplicación de escritorio y la aplicación de integración del CNBA.

A partir de este objetivo general se trazaron los siguientes **objetivos específicos:**

- especificar los requisitos del componente de software.
- diseñar un componente de software capaz de realizar de forma segura la transferencia de datos entre las aplicaciones clientes y la aplicación de integración del CNBA.
- implementar el componente de software diseñado.

Los que se cumplirán a través de las siguientes **tareas de la investigación:**

- selección de las tecnologías para la comunicación entre dos aplicaciones a través de Internet a partir de las existentes;
- selección de los mecanismos de seguridad a utilizar en el proceso de transferencia de datos entre dos aplicaciones;
- análisis y descripción del sistema modelado para la Solución Informática del CNBA;
- definición de los requisitos funcionales y no funcionales del componente;
- descripción de los casos de uso del sistema relacionados con el componente;
- selección de los patrones de diseño;
- confección del diagrama de clases del diseño;
- confección de los diagramas de interacción del diseño;
- implementación del componente;
- realización de pruebas de sistema;

Aportes prácticos esperados de la investigación

El resultado de la presente investigación será un componente de software para la transferencia de datos a través de Internet entre las aplicaciones (clientes y servidor) del proyecto MINPPAL-CNBA. Su diseño posibilitará la reutilización en la solución de otros problemas.

A continuación se describe de manera resumida el contenido que se expone en cada capítulo:

Capítulo 1: Fundamentación Teórica. Este capítulo comprende el estudio del estado del arte acerca de los elementos fundamentales a tener en cuenta para la solución del problema planteado, se expone una breve explicación de las tendencias actuales que existen sobre el proceso de transferencia de datos entre dos aplicaciones a través de internet y los mecanismos de seguridad existentes. Se argumenta la selección de las tecnologías y metodologías que se usarán en ésta investigación para dar solución al problema científico.

Capítulo 2: Diseño del componente: Este capítulo describe la propuesta de solución para el problema científico de esta investigación, mostrando los procesos del negocio mediante el Modelo de Dominio. Se presenta el Diagrama de Casos de Uso del Sistema con las especificaciones de los casos de uso correspondientes. Además se describen los requisitos funcionales y no funcionales del componente, los patrones de diseño empleados y los diagramas de clases y de interacción del diseño. Así como la distribución física de los nodos a través del diagrama de despliegue.

Capítulo 3: Implementación del componente. Este capítulo describe la implementación del componente. Se presenta el diagrama de componentes y fragmentos relevantes del código; además de la descripción de pruebas exploratorias, funcionales y de sistema realizadas.

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA

En este capítulo se exponen los principales resultados del estudio del proceso de transferencia de datos a través de Internet y sus tendencias actuales; también se hace referencia a las principales tecnologías y mecanismos de seguridad existentes; y se incluye una descripción de las características de un componente de software y los patrones a analizar para su diseño. Se establece la metodología de desarrollo, las herramientas y técnicas que se utilizan en el transcurso de la investigación, además de una fundamentación de las pruebas a realizar para la validación del resultado.

1.1 COMPONENTE DE SOFTWARE

Un componente de software es una unidad de composición con interfaces contractualmente especificadas y explícitas sólo con dependencias dentro de un contexto. Un componente de software puede ser desplegado independientemente y es sujeto a la composición de terceros. [4]

Características claves para que un elemento pueda ser catalogado como componente:

Identificable: Debe tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación.

Auto contenido: Un componente no debe requerir de la utilización de otros para finiquitar la función para la cual fue diseñado.

Puede ser reemplazado por otro componente: Se puede reemplazar por nuevas versiones u otro componente que lo reemplace y mejore.

Con acceso solamente a través de su interfaz: Debe asegurar que estas no cambiarán a lo largo de su implementación.

Sus servicios no varían: Las funcionalidades ofrecidas en su interfaz no deben variar, pero su implementación sí.

Bien documentado: Un componente debe estar correctamente documentado para facilitar su búsqueda, actualización o integración con otros.

Es genérico: Sus servicios deben servir para varias aplicaciones.

Reutilizado dinámicamente: Puede ser cargado en tiempo de ejecución en una aplicación.

Independiente de la plataforma: Hardware, Software, Sistema Operativo.

Durante el estudio realizado no se ha encontrado ningún software que permita la transferencia de datos y que a su vez sea reutilizable, la mayoría de las soluciones son en correspondencia al problema particular que resuelven. Las tecnologías que se utilizan en la transferencia de datos, de manera general pueden resultar complejas para los desarrolladores, por ello se propone la implementación de un componente de software que pueda ser utilizado sin conocer los detalles de su funcionamiento, y que constituya una solución general y aplicable a futuros proyectos.

1.2 TRANSFERENCIA DE DATOS A TRAVÉS DE INTERNET

La transmisión de datos ha sido históricamente uno de los problemas más estudiados por el hombre en la búsqueda del perfeccionamiento de la comunicación entre diversas fuentes. La aparición de internet incrementó las expectativas y los esfuerzos en el estudio del tema referido, ya que, además de ser internet un medio de almacenaje/recuperación de información y conocimiento de extensión universal, está concitando numerosas aplicaciones basadas fundamentalmente en su capacidad bidireccional de manejar la información.

Los avances en internet han permitido el desarrollo de nuevas vías de negocio y se han perfeccionado los sistemas empresariales, fundamentalmente en aquellas empresas e instituciones que necesitan de información externa. Su contribución está vinculada fundamentalmente al control de la logística y la planificación.

Vale destacar que internet se concibe como una red completamente abierta, por ello insegura, aspecto reconocido y destacado por las autoridades líderes en el manejo de la red como una característica inherente de la misma. Se han puesto de manifiesto también vulnerabilidades y faltas de seguridad importantes en diferentes sistemas; por ello, el reto para los desarrolladores de estas aplicaciones consiste en asegurar la fidelidad de la información a transmitir y evitar la intromisión de terceros en la comunicación.

Actualmente las compañías que se interesan por implementar sistemas de comunicación a través de internet, establecen los siguientes requerimientos:

- Seguridad: protección de las comunes y crecientes amenazas de internet.

- Control: control total de toda la información que se envía – quién, qué, cuándo, dónde – y la habilidad de auditar todas las actividades en todo momento.
- Integración: la transferencia de información habitualmente se realiza entre diferentes entidades, por ello es importante que sea capaz de permitir a otros conectarse de manera eficiente y segura.

1.2.1 Middleware

Se conoce como Middleware a la capa de software intermedio que consiste en un conjunto de servicios que permite la ejecución de múltiples procesos en una o más computadoras a través de la red [5]. Esta capa de software permite gestionar la comunicación entre plataformas heterogéneas utilizando los diferentes mecanismos existentes para entornos distribuidos.

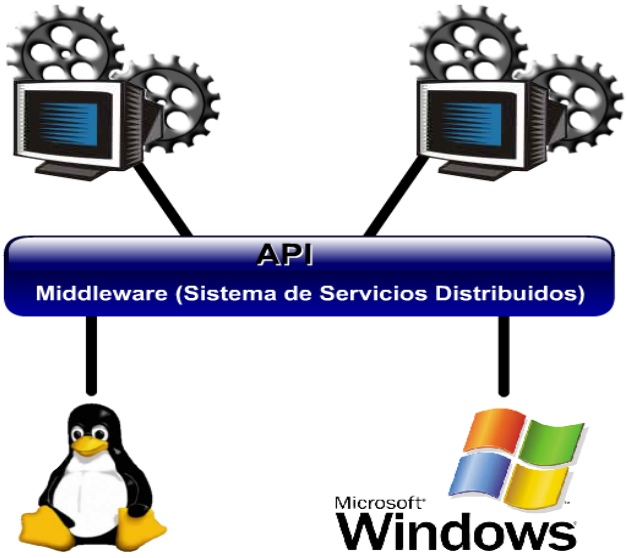


Fig. 2 Ubicación de los Middleware en las capas de un sistema

Existen dos tipos:

1. Software intermedio general: Servicios generales que requieren todos los clientes y servidores; por ejemplo: software para las comunicaciones que usan el TCP/IP, software parte del sistema operativo que, por ejemplo, almacena los archivos distribuidos, software de autenticación, el software intermedio de mensajes de clientes a servidores y viceversa.
2. Software intermedio de servicios: Software asociado a un servicio en particular, por ejemplo: software que permite a dos BD conectarse a una red cliente/servidor (ODBC: Conectividad abierta de BD), software de objetos distribuidos, software intermedio asociado a productos de

seguridad específicas (conexiones seguras: sockets).

Se caracterizan fundamentalmente por:

- independizar el servicio de su implantación, del sistema operativo y de los protocolos de comunicaciones.
- permitir la convivencia de distintos servicios en un mismo sistema.
- permitir la transparencia en el sistema.
- ser un modelo orientado a objetos.

Para la solución del problema, el componente de software a implementar se clasificaría como “software intermedio de servicios”, relacionado con las tecnologías de objetos distribuidos y la implementación de conexiones seguras en un entorno cliente/servidor.

1.2.2 Protocolos de Comunicación sobre Internet

La familia de protocolos de internet es un conjunto de protocolos de red en la que se basa internet y que permiten la transmisión de datos entre redes de computadoras. En ocasiones se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), que fueron los dos primeros en definirse, y que son los más utilizados de la familia. [6]

Se pueden mencionar además el protocolo HTTP (Hyper Text Transfer Protocol), que se utiliza para acceder a las páginas Web, además de otros como el ARP (Address Resolution Protocol) para la resolución de direcciones, el FTP (File Transfer Protocol) para transferencia de archivos, y el SMTP (Simple Mail Transfer Protocol) y el POP (Post Office Protocol) para correo electrónico, TELNET (TELecommunication NETwork) para acceder a equipos remotos, entre otros.

En el protocolo TCP/IP la capa de aplicación es la capa a la que acceden de forma directa la mayoría de las aplicaciones que usan internet. En ella se reciben los datos, que son pasados a las capas inferiores para que sean enviados a su destino. A este nivel pertenecen protocolos tales como: HTTP, FTP, SSH (Secure Shell)[1], SMTP, entre otros.

Estos protocolos se encargan de la transmisión de datos básicamente y del control de los paquetes en la red, pero carecen de mecanismos de seguridad, por ello es necesario el estudio y la aplicación de

[1] Protocolo para acceder a una computadora remota en la red.

mecanismos y protocolos seguros para la transmisión de datos entre las aplicaciones, aspecto que será abordado más adelante en este capítulo.

1.3 TECNOLOGÍAS Y FRAMEWORKS

1.3.1 Java

El principal objetivo de los diseñadores de Java, y dado el gran crecimiento de las redes en los últimos años, fue desarrollar un lenguaje cuyas aplicaciones una vez compiladas pudiesen ser inmediatamente ejecutables en cualquier máquina y sobre cualquier sistema operativo. [7]

Los diseñadores de Java trataron de mantener las facilidades básicas del lenguaje en un mínimo y proporcionar un gran número de extras con las librerías de clases.

Es un lenguaje de programación seguro, que no puede acceder a los recursos del sistema de manera incontrolada. Por este motivo se eliminó la posibilidad de manipular la memoria mediante el uso de punteros y la capacidad de transformación de números en direcciones de memoria.

El lenguaje de programación Java ha sido totalmente mejorado, ampliado y probado por una comunidad activa de unos cuatro millones de desarrolladores de software. [8]

La tecnología Java, es una tecnología madura, extremadamente eficaz y sorprendentemente versátil, se ha convertido en un recurso inestimable ya que permite a los desarrolladores:

- Desarrollar software en una plataforma [2] y ejecutarlo en prácticamente cualquier otra.
- Crear programas para que funcionen en un navegador web y en servicios web.
- Desarrollar aplicaciones para servidores como foros en línea, tiendas, encuestas, procesamiento de formularios HTML, entre otras.
- Combinar aplicaciones o servicios basados en la tecnología Java para crear servicios o aplicaciones totalmente personalizados.
- Desarrollar potentes y eficientes aplicaciones para teléfonos móviles, procesadores remotos, productos de consumo de bajo coste y prácticamente cualquier dispositivo digital.
- Crear aplicaciones en entornos distribuidos utilizando tecnologías como RMI y CORBA, que son abordadas en este capítulo.

[2] Hardware, software o sistema operativo

Además de las características expuestas anteriormente, se selecciona la tecnología java en el desarrollo para lograr un mejor acoplamiento con el resto de la solución (las aplicaciones de escritorio e integración) que se ha desarrollado usando java.

1.3.2 Frameworks

Conceptualmente “un framework es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación” [9]. Son soluciones que implementan patrones y ayudan a desarrollar funciones dentro de las aplicaciones a un nivel superior. Tienen la funcionalidad de abstraer complejas implementaciones y hacer el desarrollo más ágil, ya que proveen soluciones a problemas comunes. Son como un conjunto de librerías que usan las aplicaciones. Constituyen elementos a tener en cuenta a la hora de modelar una arquitectura pues hacen que se cubran en gran medida los objetivos con los que debe cumplir la misma.

Para la solución del problema de esta investigación es importante la utilización de frameworks, pues se está hablando de la comunicación entre aplicaciones, por lo que es importante el total entendimiento entre las mismas y la mejor forma de lograrlo es implementando, a través del uso de patrones y estándares que establecen los frameworks.

Existen numerosos frameworks sobre la plataforma J2EE. Algunos se dedican solo a una parte del desarrollo o a un tipo de aplicación específica: Java Server Faces (JSF) para las interfaces, Struts para el trabajo con aplicaciones Web, entre otros.

En la implementación del componente de software se usará Spring, por ser el más integrado y uno de los más reconocidos para el desarrollo de aplicaciones empresariales. A continuación se exponen las principales ventajas que éste ofrece, entre ellas se encuentra el soporte de tecnologías de objetos distribuidos y servicios web.

1.3.3 Spring Framework

Spring Framework es una librería open source [3] que ofrece facilidades avanzadas para la organización de servicios en una aplicación Java [10]. Es considerado el principal framework basado en soluciones que requieren inyección de dependencia. Recientemente (a partir de Spring 2.0) ha logrado separar (fuera de la lógica del negocio) parte de la arquitectura, mediante el uso de anotaciones en documentos escritos en XML, lo que constituye una fortaleza en el diseño de componentes de software basados en servicios.

[3] Código abierto

Tiene además sus propios frameworks para el manejo de los aspectos esenciales a tener en cuenta en el desarrollo:

- Acegi: para la seguridad.
- Spring MVC: para el desarrollo web.
- Spring-AOP: para el desarrollo de Aspect Oriented Programming.
- Spring-WebServices: para el uso de servicios web.
- Swing: framework de inyección de dependencias.
- Spring-JDBC: framework de persistencia.
- Spring Web Flows: framework para definir web flows.

Spring Framework es recomendable para el desarrollo de aplicaciones, pues constituye un contenedor ligero que gestiona los objetos de las mismas, manteniendo el código limpio y claro. Los objetos gestionados no necesitan implementar o extender funcionalidades especiales, por cuanto es un mecanismo totalmente transparente. Además, los servicios que brinda Spring son fundamentales en cualquier aplicación y abstraen al programador de codificar cientos, e incluso, miles de líneas, esto hace que aumente la productividad en los proyectos actuales que lo utilicen.

Spring soporta varias estrategias remotas que son objeto de esta investigación para el desarrollo del componente, entre ellas se encuentran Hessian, Burlap y HTTP Invoker, además de soportar servicios web. Tiene la capacidad de exponer el mismo servicio bajo diferentes protocolos. Hace un uso consistente del estilo de configuración a partir de la creación de proxies con interfaces de servicio de Java y exportando entidades manejadas por el mismo.

1.3.3.1 Tecnologías remotas de Spring Framework

Hessian y Burlap

Hessian y Burlap son mecanismos de servicios desarrollados por Caucho Technology. Hessian es un protocolo binario y Burlap es la contraparte basada en XML [11], ambos pueden ser usados para hacer llamadas a procedimientos remotos sobre HTTP. Pueden ser usados para serializar cualquier tipo de objeto que se vaya a enviar a través de la web. Son buenas alternativas cuando solo se requiere hacer simples llamadas a métodos entre procesos de Java, sin importar las cuestiones de interoperabilidad.

Los servicios Hessian (o Burlap) son definidos como servlets usando clases (HessianServlet (o

BurlapServlet)) definidas en framework de Caucho. La aplicación de servicio para ser exportada debe ser implementada como una subclase de las anteriormente mencionadas o como una clase Java implementando la interfaz de servicio. En el cliente son proporcionadas clases factory que generan el proxy para una interfaz y dirección específica del servicio.

Hessian y Burlap usa sus propios algoritmos de serialización para los tipos primitivos, colecciones y otros. Dichos mecanismos de serialización tiene algunas limitaciones, cuando existe la necesidad de reconstruir objetos a través de reflection[4], Hessian y Burlap no son capaces de deserializarlos y convertirlos en instancias de variables. Además no detectan objetos con serialización propia, por ejemplo colecciones de Hibernate[5].

HTTP Invoker

HTTP Invoker es uno de los servicios remotos que son soportados por Spring Framework. El mismo emplea el modelo remoto para hacer llamadas remotas sobre HTTP y al mismo tiempo el paso de objetos de Java usando técnicas de serialización del lenguaje [12]. Esto hace que la implementación de un servicio remoto desde una clase de Java es más fácil y permite al desarrollador concentrarse en la interfaz de negocio del servicio remoto por encima de los detalles de la infraestructura del mismo. Se basa en la infraestructura de invocación de RMI, pero utiliza HTTP como protocolo de transporte.

HTTP Invoker está solo disponible en Spring, cliente y servidor deben estar basados en Spring. Es el tipo de servicio para llamadas remotas más simple de desarrollar en Java superando a RMI en cuanto al transporte sobre HTTP (evitando los cortafuegos).

1.3.4 Objetos distribuidos

En los sistemas Cliente/Servidor, un objeto distribuido es aquel que está gestionado por un servidor y sus clientes invocan sus métodos a través de un "método de invocación remota" [13]. El cliente invoca el método mediante un mensaje al servidor que gestiona el objeto; se ejecuta el método del objeto en el servidor y el resultado se devuelve al cliente en otro mensaje.

[4] A través de reflection es posible crear componentes totalmente genéricos, que funcionen con cualquier tipo de objeto (java.lang.reflect).
[5] Framework para persistencia de Java.

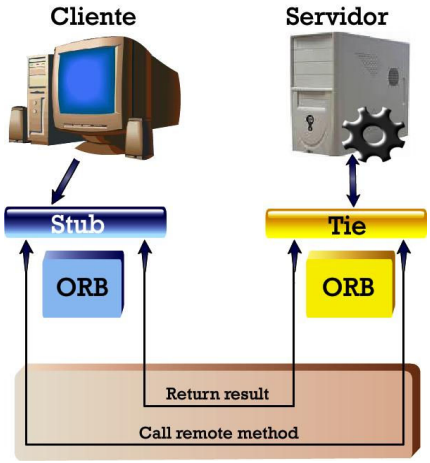


Fig. 3 Estructura general de invocación de métodos remotos

Las tres tecnologías más importantes y usadas en este ámbito son:

- CORBA
- RMI
- DCOM

A continuación se describen las principales características de cada una, exceptuando DCOM por ser una tecnología propietaria.

1.3.4.1 Common Object Request Broker Architecture (CORBA)

CORBA (arquitectura común de intermediarios en peticiones a objetos) es un estándar abierto para la distribución de objetos definido por Object Management Group (OMG). CORBA es la especificación de una arquitectura para tecnologías middleware que provee interoperabilidad entre clientes y servidores distribuidos en ambientes heterogéneos [14]. Habilita a los clientes para la invocación de métodos sobre objetos remotos de un servidor, independientemente del lenguaje utilizado y del lugar donde estén ubicados. La interacción entre el cliente y servidor es mediada por Object Request Brokers (ORBs) en ambas partes. Las funcionalidades de los objetos CORBA son definidas usando Interface Definition Language (IDL).

CORBA soporta la implementación de múltiples lenguajes: C, C++, Java, Ada95, COBOL así como algunos lenguajes script como: Perl, Python, Javascript.

De esta forma se logra una abstracción a la heterogeneidad que permite que su uso no sea nada

complejo. La forma de desarrollar con CORBA sigue una metodología concreta y fácil de seguir.

CORBA ha buscando un entorno heterogéneo, el cual constituye una visión abierta del mundo de la informática y en la cual hay cabida para diferentes sistemas y distintas filosofías, un mundo más rico que el que se puede lograr con un solo sistema alrededor del cual funcionan todas las aplicaciones.

Interface Definition Language (IDL)

El Lenguaje de Definición de Interfaces (IDL) ofrece la sintaxis necesaria para definir los procedimientos o métodos que son invocados remotamente [15], a través de IDL se definen las diversas estructuras que serán utilizadas en un ambiente CORBA.

Object Request Broker (ORB)

El ORB es la parte medular de un sistema CORBA, ya que es a través de éste que se comunican los diversos Stubs y Skeletons generados a través de IDL, es el ORB quien ofrece la conectividad en un sistema CORBA. [16]

1.3.4.2 Remote Invocation Method (RMI)

El sistema de Invocación Remota de Métodos (RMI) de Java permite, a un objeto que se está ejecutando en una Máquina Virtual Java (VM), llamar a métodos de otro objeto que está en otra VM diferente. Esta tecnología está asociada al lenguaje de programación Java, es decir, que permite la comunicación entre objetos creados en este lenguaje. [17]

Las aplicaciones RMI normalmente comprenden dos programas separados: un servidor y un cliente. Una aplicación servidor típica crea cierta cantidad de objetos remotos, hace accesibles unas referencias a dichos objetos remotos. Una aplicación cliente típica obtiene una referencia remota de uno o más objetos remotos en el servidor y llama a sus métodos. RMI proporciona el mecanismo por el que se comunican y se pasan información del cliente al servidor y viceversa.

1.3.5 Servicios web

Un servicio web (en inglés Web Service) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios Web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. [18]

Las organizaciones OASIS (Organization for the Advancement of Structured Information Standards) y

W3C (World Wide Web Consortium) son los comité responsables de la arquitectura y reglamentación de los servicios web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios web se ha creado el organismo WS-I (Web Services Interoperability Organization), encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

Los servicios web son muy prácticos, aportan gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad es muy importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más usada.

En lo adelante se describen las características principales de las tecnologías fundamentales para la publicación y el acceso a servicios Web en la plataforma Java. Destacando como elementos comunes en todos los casos, la independencia del servicio y la plataforma sobre la cual está desarrollado.

1.3.5.2 JAX-RPC y JAX-WS

JAX-RPC y JAX-WS son APIs (Application Program Interface) de Java basados en XML que hacen posible escribir aplicaciones Java que usen XML para hacer llamadas a procedimientos remotos (RPC). El segundo surge como una actualización-mejora del primero, el mismo integra otros APIs de Java como son JAXB para el mapeo de XML y la validación de esquemas, lo que elimina la doble definición del manejo de XML que utilizaba JAX-RPC. JAX-WS integra además la seguridad establecida en JSR (Web Services Security). [19]

Anteriormente se han abordado Java IDL y Java RMI como APIs de Java para hacer llamadas a procedimientos remotos. Todos estos APIs tienen en común un API para empaquetar/desempaquetar argumentos y para transmitir y recibir llamadas a procedimientos. La diferencia es que JAX-RPC y JAX-WS están basados en XML y está dirigido a servicios Web.

Ambos hacen sencillo el uso de servicios Web, y facilitan su desarrollo, especialmente usando la plataforma J2EE[6]. Un servicio Web basado en RPC básicamente es una colección de procedimientos que pueden ser llamados por un cliente remoto desde Internet. El propio servicio es una aplicación servidora desarrollada sobre un contenedor del lado del servidor que implementa procedimientos que está disponible para llamadas de clientes.

JAX-WS permite a los desarrolladores implementar servicios orientados a mensajes. La invocación a

[6] Java 2 Enterprise Edition. Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java.

un servicio Web es representada por un protocolo basado en XML como SOAP. La especificación de SOAP define la estructura de desarrollo y las convenciones para la representación de las invocaciones y respuestas de un servicio Web. Esas llamadas y respuestas son transmitidas como mensajes SOAP (archivos XML) sobre HTTP.

Los mensajes SOAP son complejos pero el API JAX-WS oculta esa complejidad al desarrollador. En el servidor se especifican las operaciones del servicio Web, mediante la definición de los métodos en una interfaz escrita en Java. El desarrollador escribe una o más clases que implementen la interfaz. El cliente crea un proxy (objeto local que representa el servicio) que simplifica la llamada a los métodos. Con JAX-WS, el desarrollador no genera o interpreta mensajes SOAP, su sistema de ejecución es el encargado de convertir las llamadas del API y las respuestas en mensajes SOAP.

1.3.6 Selección de las tecnologías para la transferencia

Una vez descritas las características fundamentales de las tecnologías candidatas para el desarrollo, se opta por la implementación de dos de las tecnologías planteadas, para ofrecer la posibilidad de escoger una de éstas, de acuerdo al entorno en que se implante el componente de software, según lo recomendado para cada tecnología:

RMI: por ser un mecanismo propio de java que garantiza una comunicación eficiente a través del uso de objetos remotos. Serializa los objetos durante el proceso de comunicación y tiene facilidades para la implementación de la seguridad a través de sockets. Se recomienda su utilización en entornos donde no haya problemas con el paso de firewalls.

HTTP Invoker: soportado por Spring, ofrece facilidades en la exportación de servicios remotos, el framework maneja internamente la lógica del servicio; el desarrollador solo debe preocuparse por la implementación del servicio. Similar a RMI utiliza el mecanismo de serialización de java, tiene como ventaja sobre el mismo que utiliza HTTP como protocolo para la comunicación, por lo que no tiene problemas con el paso de firewalls. La publicación se realiza a través de un servidor de aplicaciones web que se encarga de la seguridad y permite el acceso a través de HTTPS. Se recomienda su uso en entornos abiertos, donde se desconozcan las configuraciones de los firewall y el funcionamiento de las redes.

1.4 MECANISMOS DE SEGURIDAD

Las redes locales, y por extensión Internet, no fueron diseñados en principio para ser seguros. La mayoría de los software y los hardware creados hasta mediados de los años 90, no tuvieron la

seguridad como objetivo de primer orden.

En la actualidad los sistemas existentes en Internet manejan informaciones claves de empresas, instituciones e incluso del gobierno, por ello se hace necesario extremar las medidas de seguridad de toda la información manejada por estos sistemas.

La tecnología de encriptación de información destinada a pasar a través de la red ha evolucionado bastante, haciéndose popular el término VPN para hacer referencia a canales que encriptan la información de un modo más o menos transparente. [20]

Quizás la aplicación más antigua de la criptografía sea la de establecer canales de comunicaciones seguros entre dos interlocutores. Aunque se pueden emplear diferentes medios de envío para los mensajes a transmitir, todos ellos tienen en común el hecho de que ambos interlocutores carecen de control sobre el canal de comunicación. En general, se ha de considerar que los mensajes serán depositados en medios ajenos a los interlocutores y usualmente hostiles. Existen dos peligros fundamentales para un mensaje liberado en un medio hostil:

- **Acceso por agentes no autorizados**: asumiendo que intrusos pueden acceder a los mensajes transferidos, el sistema de protección debe centrarse en garantizar que el mensaje resulte ininteligible al atacante.
- **Alteraciones del mensaje**: este problema puede llegar a ser peor que el anterior, ya que si al recibir un mensaje alterado se reconoce igualmente, las consecuencias para la comunicación pueden ser nefastas. En este tipo de riesgo las alteraciones pueden aplicarse sobre el contenido del mensaje y en la cola de información acerca de su verdadera procedencia.

Ciertas aplicaciones estándares han recibido soluciones de encriptación también estándar. El caso de la web encriptada bajo Secure Socket Layer (HTTPS) junto con la industria de certificados digitales es el caso más conspicuo.

La seguridad en los sistemas desarrollados sobre internet, reconoce en la actualidad tres desafíos fundamentales, en los que se debe centrar los esfuerzos de los diseñadores de esta clase de sistema:

- Confidencialidad: Protección contra individuos no autorizados.
- Integridad: Protección contra la alteración o corrupción.
- Disponibilidad: Protección contra la interferencia con los procedimientos de acceso a los recursos.

1.4.1 Virtual Private Network

VPN (Virtual Private Network) es una red privada de datos que utiliza como medio de enlace la infraestructura de comunicación ya creada, manteniendo la privacidad con el uso de túneles, protocolos y procedimientos seguros. [21]

Este método permite enlazar dos o más redes de forma tal que simulan una única red privada. Así la comunicación entre computadoras se establece como si fuera punto a punto.

También un usuario remoto se puede conectar individualmente a una LAN (Local Area Network) utilizando una conexión VPN, y de esta manera utilizar aplicaciones y enviar datos de manera segura.

Las Redes Privadas Virtuales utilizan tecnología de túnel para la transmisión de datos mediante un proceso de encapsulación y en su defecto de encriptación, esto es importante a la hora de diferenciar Redes Privadas Virtuales y Redes Privadas, ya que esta última utiliza líneas telefónicas dedicadas a formar la red.

Una de las principales ventajas de una VPN es la seguridad. Los paquetes viajan a través de infraestructuras públicas (Internet) en forma encriptada y a través del túnel de manera que sea prácticamente ilegible para quien intercepte estos paquetes.

Esta tecnología es muy útil para establecer redes que se extienden sobre áreas geográficas extensas como ciudades y a veces hasta países y continentes. Por ejemplo en empresas que tienen oficinas remotas en puntos distantes, la idea de implementar una VPN haría reducir notablemente los costos de comunicación.

1.4.2 OpenVPN

OpenVPN es una solución de conectividad basada en software SSL y VPN, ofrece conectividad punto a punto con validación jerárquica de usuarios y host conectados remotamente, resulta una buena opción en tecnologías Wi-Fi (redes inalámbricas EEI 802.11) y soporta una amplia configuración. Está publicado bajo la licencia GPL (General Public License), de software libre. [22]

Hay dos tipos básicos de túneles que se pueden crear con OpenVPN:

- Túnel IP – se usa para encaminar tráfico IP punto a punto sin broadcast. Es más eficiente que un puente ethernet y más fácil de configurar.
- Puente ethernet -- se puede usar para encapsular tanto protocolos IP como no-IP. Este tipo de túnel es apropiado para aplicaciones que se comunican utilizando difusión (broadcast), tales

como la red de Windows y juegos de área local (LAN). Es más difícil de configurar.

Ninguna otra solución ofrece una mezcla semejante de seguridad a nivel empresarial: seguridad, usabilidad y riqueza de características. Es una solución multiplataforma que ha simplificado mucho la configuración de redes VPN y ha dejado atrás los tiempos de soluciones difíciles de configurar como IPsec. OpenVPN es una solución de conectividad más accesible para inexpertos en este tipo de tecnologías.

1.4.3 SSL y Certificados Digitales

El protocolo Secure Socket Layer (SSL) fue desarrollado por Netscape en 1996, para asegurar el transporte y enrutamiento de datos a través de capas de aplicaciones como: HTTP, LDAP o POP3. SSL es diseñado para usar TCP como capa de comunicación Para proporcionar una conexión segura y autenticada entre dos puntos a través de una red (por ejemplo en servicios cliente-servidor). Puede ser utilizado además para la protección de los datos en tránsito en cualquier servicio de red, se utiliza principalmente en el servidor HTTP y las aplicaciones cliente.

Actualmente la mayoría de los servidores HTTP disponibles soportan una sesión de SSL, y los navegadores en sus últimas versiones tienen SSL habilitado para el software del cliente [23], por lo que se ha convertido en el método elegido para asegurar las transmisiones de datos por internet.

Un certificado digital es un archivo electrónico que identifica de modo único a individuos y servidores. Los certificados digitales funcionan como una especie de pasaporte o credencial digital que autentica al servidor antes de establecer la sesión SSL. Por lo general, los certificados digitales están firmados por una tercera persona independiente y fiable para garantizar su validez.

SSL hace uso del sistema de codificación con dos claves: una pública y una privada, desarrollado por RSA[7] y proporciona comunicaciones seguras mediante la combinación de los siguientes dos elementos:

- Autenticación: el certificado digital va unido a un dominio específico y una autoridad certificadora (CA) realiza una cantidad de verificaciones para confirmar la identidad de la organización que solicita el certificado antes de emitirlo. De este modo, el certificado sólo puede instalarse en el dominio contra el cual ha sido autenticado, ofreciendo a los usuarios la seguridad que necesitan. (Ver Fig.4)

[7] RSA: sistema criptográfico con clave pública (RSA por las iniciales de sus creadores Ron Rivest, Adi Shamir y Len Adleman)

- Codificación: la codificación es el proceso de transformar la información para hacerla incomprensible para todos salvo el receptor al que va dirigida. Esto constituye la base de la integridad y la privacidad de los datos, necesarias para el comercio electrónico.

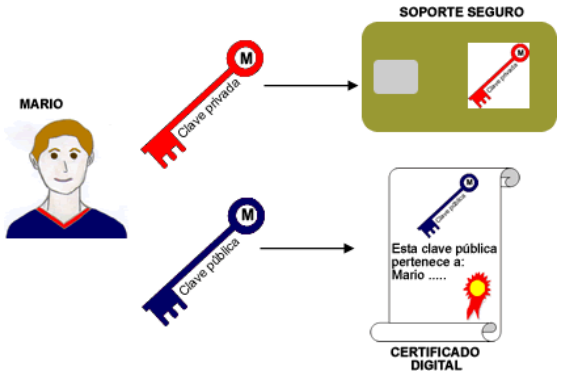


Fig. 4 Funcionamiento de los Certificados Digitales

Existen dos áreas amplias de aplicación para los certificados SSL:

- Aseguramiento de la comunicación entre el explorador y el servidor web: es actualmente la principal aplicación y se aplica con mayor frecuencia a los sitios web de comercio electrónico para garantizar la transferencia de información sobre pagos. El tipo de datos considerados sensibles se está ampliando actualmente desde los datos financieros para incluir toda la información personal identificable, incluidos los números de identidad y seguridad social, y, cada vez más, las direcciones de correo electrónico.

- Aseguramiento de la comunicación entre servidores: Cada vez recurren más empresas a certificados SSL para asegurar las comunicaciones entre servidores. Esta es un área de aplicación que ofrece a las empresas varias opciones para mejorar la seguridad de los datos y la privacidad de la red. Actualmente, asegurar la comunicación entre servidores de correo electrónico es la aplicación más usual, si bien también es posible asegurar sitios FTP, bases de datos y servidores de aplicaciones, entre otros.

1.4.4 ¿Por qué SSL?

En el entorno del problema se hace difícil la creación de una VPN pues el CNBA es una organización independiente de los entes que le suministrarán información, por lo que se plantea el uso de SSL. Este último es uno de los mecanismos más utilizados en la actualidad para el establecimiento de la seguridad y garantiza la autenticación de los usuarios a través del uso de los certificados digitales y la

encriptación de todo el tráfico entre las aplicaciones que lo utilizan.

1.5 METODOLOGÍA DE DESARROLLO

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo [24]. La metodología indica cómo hay que obtener los distintos productos parciales y finales. Entre los procesos de desarrollo más conocidos se tienen: Programación Extrema (Extreme Programming, XP), clasificado como método ligero, y Proceso Unificado de Desarrollo (Rational Unified Process, RUP), que se clasifica como método pesado.

1.5.1 Proceso Unificado de Desarrollo

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, es el proceso unificado de desarrollo de software que plantea quién hace qué, cuándo y cómo. Tiene como objetivo asegurar la producción de software de calidad dentro de plazos y presupuestos predecibles. Se actualiza constantemente para tener en cuenta las mejores prácticas y utiliza además UML como lenguaje de modelado.

El ciclo de vida de RUP se caracteriza por:

- **Estar dirigido por casos de uso:** Los casos de uso reflejan lo que los usuarios futuros necesitan, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo.
- **Estar centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo, por lo que describe los elementos del modelo más importantes para su construcción. El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML.
- **Ser iterativo e incremental:** RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, unos más que otros. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto.

La metodología RUP divide en 4 fases el desarrollo del software:

- Inicio: El objetivo es determinar la visión del proyecto.

- Elaboración: El objetivo es determinar la arquitectura óptima.
- Construcción: El objetivo es obtener la capacidad operacional inicial.
- Transición: El objetivo es obtener el release del proyecto.

1.5.2 Extreme Programming

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizada para proyectos de corto plazo y corto equipo. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. [25]

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo.

Lo fundamental en este tipo de metodología es:

- La comunicación entre los usuarios y los desarrolladores.
- La simplicidad al desarrollar y codificar los módulos del sistema.
- La retroalimentación concreta y frecuente del desarrollo, el cliente y los usuarios finales.

1.5.3 ¿Por qué RUP?

Además de lo expuesto anteriormente, se selecciona RUP como metodología de desarrollo, por su adaptabilidad al proyecto y las condiciones de desarrollo que garantiza el cumplimiento de las metas trazadas. Además por su buena concepción en cuanto a la organización del trabajo, dada por la distribución de las actividades según la fase de desarrollo y el establecimiento de roles para la ejecución de las mismas. Propone además, la generación de los artefactos necesarios para una buena documentación y el entendimiento entre los miembros del equipo.

1.5.4 Roles y Artefactos

RUP es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Un rol es una definición abstracta de un conjunto de actividades realizadas y de artefactos obtenidos. Los roles son realizados típicamente por un individuo, o un conjunto de individuos, trabajando juntos en equipo.

De acuerdo con las necesidades y características de ésta investigación los roles que se desarrollarán serán: diseñador, implementador y arquitecto de software, pertenecientes al grupo de desarrolladores definido por RUP.

El rol Arquitecto de Software es el responsable de la arquitectura de software, que incluye las decisiones técnicas claves que restringen el diseño global y la implementación para el proyecto.

A continuación se describen solo los artefactos que fueron generados por el arquitecto de software en esta investigación:

- **Modelo de diseño**: Es un modelo de objeto que describe la realización de casos de uso, y sirve como una abstracción del modelo de implementación y el código fuente.
- **Modelo de despliegue**: Un diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el sistema final, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes software (procesos y objetos que se ejecutan en ellos).
- **Modelo de implementación**: Representa la composición física de la implementación en términos de subsistemas de implementación, y elementos de implementación (directorios y archivos, incluyendo código fuente, datos y archivos ejecutables). Específicamente de este modelo se realizará el diagrama de componentes (usado para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación).

El rol Diseñador define las responsabilidades, las operaciones, los atributos, y las relaciones de una o varias clases y determina cómo serán ajustadas al ambiente de implementación. Además, el rol

diseñador puede tener la responsabilidad de unos o más paquetes de diseño, o de diseño de los subsistemas, incluyendo cualquier contenido por los paquetes o los subsistemas.

A continuación se describen solo los artefactos que fueron generados por el diseñador en esta investigación:

- **Clase del Diseño:** Se define como la descripción de un conjunto de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos y semántica.
- **Realización de caso de uso:** Describe como un caso de uso es comprendido dentro del modelo de diseño en términos de colaboración de objetos.
- **Paquete de Diseño:** Es una colección de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes. Este es usado para estructurar el modelo de diseño dividiéndolo en pequeñas partes.

El rol Implementador es responsable de desarrollar y de probar componentes de acuerdo con los estándares adoptados del proyecto para la integración en subsistemas más grandes. Cuando los componentes de prueba, tales como drivers o partes se deben crear para apoyar la prueba, el implementador es también responsable de desarrollar y de probar los componentes de prueba y los subsistemas correspondientes.

A continuación se describen solo los artefactos que fueron generados por el implementador en esta investigación:

- **Elementos de implementación:** son las partes físicas que componen la implementación, incluyendo archivos y directorios. Además de los archivos de código del software (código, binarios o ejecutables), archivos de datos y documentación.
- **Artefactos de instalación:** se refieren al software y a las instrucciones documentadas requeridas para instalar el producto.

El resto de los artefactos correspondientes a los roles descritos fue generado en conjunto con el equipo de trabajo del proyecto.

1.6 HERRAMIENTAS PARA EL MODELADO

El modelado de sistemas software es una técnica para tratar con la complejidad inherente a estos sistemas. El uso de modelos ayuda al ingeniero de software a "visualizar" el sistema a construir. Además, los modelos de un nivel de abstracción mayor pueden utilizarse para la comunicación con el

cliente. Por último, las herramientas de modelado pueden ayudar a verificar la corrección del modelo.

Algunas de las herramientas CASE [8] conocidas son el ArgoUML, Rational Rose, Visual Paradigm, Easy CASE, Xcase, CASE Studio 2, CASEWise entre otras. Dentro de las más utilizadas se encuentran el Rational Rose y el Visual Paradigm.

Rational Rose es la herramienta CASE desarrollada por los creadores de UML Booch, Rumbaugh y Jacobson, provee un modelado basado en UML y cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables.

Visual Paradigm por su parte, es una herramienta CASE que provee soporte para la generación de código, ingeniería inversa para Java. Se integra con Eclipse, Borland JBuilder y Oracle JDeveloper, para soportar las fases de implementación en el desarrollo de software. Tiene dentro de sus características que es portable y posee gran facilidad de uso. [26]

Utiliza UML como lenguaje de modelado y tiene como características:

- Es un producto de calidad.
- Soporta aplicaciones web.
- Se integra con las siguientes herramientas Java: Eclipse/IBM WebSphere, JBuilder, NetBeans IDE, Oracle JDeveloper, BEA Weblogic.
- Es fácil de instalar y actualizar.
- Logra compatibilidad entre ediciones.

Teniendo en cuenta las características antes mencionadas del Visual Paradigm, especialmente su buena integración con los IDE para el desarrollo de java, y la ventaja de que presenta una interfaz de usuario de fácil uso y que permite realizar los diagramas y artefactos que se generan durante el desarrollo del software además de realizar un control de las versiones durante todo el ciclo de trabajo, el colectivo de autores decide utilizarla como herramienta de modelado.

1.7 HERRAMIENTAS DE DESARROLLO

Las herramientas de desarrollo son un conjunto de programas usado por los programadores para escribir aplicaciones. La selección de las herramientas de desarrollo en la presente investigación se

[8] Ingeniería de Software Asistida por Computadoras.

concentra en aquellas que sean multiplataforma y sean distribuidas bajo licencias de tipo GNU/GPL[9].

1.7.1 Netbeans

El Netbeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE Netbeans [27]. Es un producto libre y gratuito, desarrollado por una comunidad extensa de desarrolladores.

1.7.2 Eclipse

Eclipse es una poderosa herramienta que permite integrar diferentes aplicaciones para construir un entorno integrado de desarrollo (IDE) [28]. Es un potente entorno de desarrollo de Java. Usa java como lenguaje de programación ya que soporta la programación orientada a objetos (POO) y la implementación de aplicaciones resulta mucho más sencilla. Mediante Eclipse se puede crear diversas aplicaciones como son sitios web, programas Java, C++ y Enterprise Java Beans. Su principal aplicación es JDT (Java Development Tool), herramienta para crear aplicaciones en Java. Otras aplicaciones pueden ser integradas a eclipse en forma de plugins, que son reconocidos automáticamente por Eclipse al iniciar el mismo.

Beneficios de la herramienta Eclipse:

- Es una herramienta open-source.
- Soporta la construcción de una variedad de herramientas para el desarrollo de aplicaciones.
- Soporta herramientas que manipulan diferentes tipos de archivos como por ejemplo Java, C, C++, EJB, HTML, GIF, entre otros.
- Corre en una gran cantidad de sistemas operativos incluyendo Windows y Linux.

Se escoge Eclipse como IDE debido a su facilidad de uso, además de considerarse más estable y rápido; su diseño global permite tener las herramientas que necesitan los programadores inmediatamente al alcance de sus manos.

1.7.4 GlassFish

GlassFish es un servidor open source de aplicaciones que implementa las tecnologías definidas en la

[9] Licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software.

plataforma Java EE 5 y permite ejecutar aplicaciones que siguen esta especificación. Tiene como base al servidor Sun Java System Application Server 9 PE de Sun Microsystems.

GlassFish cuenta con plugins para Eclipse y con excelente soporte en NetBeans, incluye las nuevas librerías de Web Services (JAX-WS 2.0) y es la base de las nuevas plataformas SOA en JAVA.

1.7.3 Apache Tomcat

Tomcat es un servidor web con soporte de servlets y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache. [29]

Tomcat puede funcionar como servidor web por sí mismo, es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

La publicación de servicios web u otros servicios remotos sobre HTTP requieren ser desplegados en un servidor de aplicaciones web, con este objetivo se propone la utilización de Apache Tomcat durante el desarrollo, por su amplia integración con Eclipse, además de llevar años de desarrollo y explotación que hace que sea una herramienta madura a diferencia de Glassfish.

1.7.5 Subversion

Subversion es un software de sistema de control de versiones. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como svn por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. [30]

Clientes

Existen varias interfaces a Subversion, ya sea programas individuales como interfaces que lo integran en entornos de desarrollo.

- TortoiseSVN. Provee integración con el explorador de Windows. Es la interfaz más popular en este sistema operativo.
- Subclipse. "Plugin" que integra Subversion al entorno Eclipse.

- Subversive. "Plugin" alternativo para Eclipse.
- KDESvn. Provee integración con el escritorio KDE

1.7.6 Herramientas seleccionadas para el desarrollo

Las herramientas seleccionadas para el desarrollo son Eclipse como IDE de desarrollo, Apache Tomcat como servidor web y Subversion (específicamente el plugin Subclipse) como controlador de versiones, que mantienen una excelente integración para brindar un ambiente de desarrollo óptimo.

1.8 PATRONES DE DISEÑO

Un patrón es una solución recurrente para un problema típico y en un contexto determinado. La solución se refiere a la respuesta al problema, por lo que ayuda a resolver las dificultades de diseño en problemas similares. Los patrones deberían comunicar soluciones de diseño a los desarrolladores y arquitectos que los leen y los utilizan. [31]

Los patrones se clasifican en 3 grandes categorías basadas en su propósito, se aborda en cada caso los que se aplican a esta investigación.

1.8.1 Patrones Creacionales

Los patrones creacionales tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es el de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados. Utilizamos de manera práctica el patrón Factory y el Singleton de este grupo de patrones creacionales.

1.8.1.1 El patrón Factory

Es utilizado para la selección y retorno de una instancia de una clase, de un número de clases similares basado en la información que se le indique a la clase Factory.

Este patrón propone crear objetos por medio de distintos métodos, a partir de algunos atributos, pero sin especificar una clase concreta. Generalmente estos métodos son estáticos y reciben como parámetros, los valores de los atributos necesarios para crear el objeto.

Los nombres de los métodos terminan siendo más descriptivos que un simple constructor sobrecargado. Aunque la ventaja principal de este patrón es que distintas subclases pueden sobrescribir un método fábrica para devolver una instancia de una clase nueva, que no existía cuando se escribió el método fábrica original. De esta manera se puede extender una librería de clases de

manera sencilla.

Se utiliza fundamentalmente en los siguientes casos:

- Frameworks y librerías extensibles.
- Clases que necesiten verificaciones extra antes de ser instanciadas.
- Clases que son construidas de distinta manera de acuerdo al uso.

1.8.1.2 El patrón Singleton

Es un patrón que asegura la creación de una única instancia de un objeto y solo es posible el acceso global a esa única instancia. Tiene como dos características fundamentales:

1. Restricción de acceso al constructor: con esto se logra que sea imposible crear nuevas instancias. Solo la propia clase puede crear la instancia.
2. Mecanismo de acceso a la instancia: el acceso a la instancia única se hace a través de un único punto bien definido, que es gestionado por la propia clase y que puede ser accedido desde cualquier parte del código en principio.

1.8.2 Patrones Estructurales

Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

1.8.2.1 El patrón proxy

Este patrón permite acceder a un recurso mediante un intermediario con varios propósitos, tales como diferir la carga, controlar el acceso y hacer caché.

Los recursos a los que se puede acceder mediante un intermediario pueden ser archivos, conexiones de red, objetos muy grandes, entre otros. El proxy permite acceder desde múltiples lugares a un mismo recurso o darle múltiples usos; cuando hacer copias es muy costoso o imposible, en lugar de eso, se crean múltiples proxies que hacen de intermediario entre el mismo recurso y sus diferentes clientes.

El proxy funciona reenviando todos los mensajes que recibe al objeto real. Debe tener una referencia a este objeto o una manera de buscarla. Dependiendo del escenario en que se lo use, el proxy puede incluir distintas clases comportamiento.

1.8.3 Patrones de Asignación de Responsabilidades

Los patrones de Asignación de Responsabilidades (conocidos como GRASP), tienen como objetivo esencial determinar las responsabilidades que tendrán cada clase de un sistema y su forma de interacción. Durante el diseño se han aplicado, porque los mismos propician la modularidad y consistencia del sistema así como la mayor independencia entre clases. Se describen a continuación de manera resumida los que se utilizarán en esta investigación:

1.8.3.1 Patrón Experto

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide.

1.8.3.2 Alta cohesión

El patrón consiste en asignar una responsabilidad de modo que la cohesión siga siendo alta. En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Las clases con baja cohesión a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos.

1.8.4 Inyección de dependencia

Inyección de dependencia es una de las implementaciones del concepto Inversión de Control. Desde la perspectiva de la Inversión de Control se reconoce que los objetos involucrados en la lógica de negocio de una aplicación dependen de otros objetos de negocio, objetos de acceso a datos y recursos compartidos. Todos ellos se ejecutan en un ambiente denominado Contenedor Ligero, el cual es responsable de establecer las dependencias entre los objetos.

Este patrón tiene las siguientes ventajas:

- La búsqueda de recursos es removida del código de la aplicación.
- No hay dependencia con respecto a una Interfaz de Programación de Aplicaciones provista por un contenedor específico.
- No requiere interfaces especiales.

1.9 PRUEBAS DE SISTEMA

Según las definiciones de la IEEE/ANSI, las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. [32]

Los tipos de pruebas que se consideraron para validar el componente de software incluyen: las pruebas de funcionalidad (seguridad y volumen) y soportabilidad (configuración).

Cada tipo de prueba tiene un objetivo específico y una técnica que lo soporte.

Prueba de seguridad: garantiza que los usuarios están restringidos a funciones específicas o su acceso está limitado únicamente a los datos que están autorizados a acceder. Solo aquellos usuarios autorizados a acceder al sistema son capaces de ejecutar las funciones del mismo. Además cumple objetivos específicos de seguridad de cada sistema.

Prueba de volumen: Consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.

Prueba de configuración: se enfoca en evaluar las diferentes variaciones de una aplicación integrada, contra sus requerimientos de configuración. Tiene como meta hacer que la aplicación falle en cumplir sus requerimientos de configuración, de manera que los defectos escondidos sean identificados, analizados, arreglados, y prevenidos en el futuro.

CONCLUSIONES

El estudio de las tendencias del proceso de transferencia de datos posibilitó la selección de las tecnologías de comunicación remota a utilizar en esta investigación; además quedó evidenciada la necesidad de un mecanismo de seguridad que garantice la transferencia de una manera segura. Se establecieron la metodología y herramientas para el modelado y desarrollo de este componente según las características del proyecto. Con el estudio de las pruebas funcionales y de soportabillidad, se seleccionaron las pruebas de seguridad, volumen y configuración, necesarias éstas para la validación del resultado.

CAPÍTULO 2
DISEÑO DEL COMPONENTE

En este capítulo se describe la propuesta de solución del sistema según el problema descrito. Para ello se muestran los procesos del negocio mediante el modelo del dominio y se destacan los elementos vinculados a esta investigación; además, se brinda una concepción general del sistema propuesto y se particulariza en las funcionalidades y características que tendrá el componente a desarrollar. Se presenta el diagrama de casos de uso del sistema con las correspondientes especificaciones de los casos de uso asociados al componente, el diagrama de despliegue. Se describen los patrones de diseño empleados así como los diagramas de clases y de interacción del diseño.

2.1 BREVE DESCRIPCIÓN DEL SISTEMA

Teniendo en cuenta que actualmente hay entes que pueden suministrar la información digitalizada y otros la presentan en copia dura, la solución informática permitirá que los datos digitales sean cargados mediante un Excel a través de una aplicación de escritorio y para los entes que presentan la información en copia dura se realizará una aplicación Web que les permitirá transcribir los datos requeridos. Vale destacar que estará asegurada la integración de ambas aplicaciones, permitiendo la recopilación y almacenamiento de los datos, ya que las informaciones recopiladas por cualquiera de las vías tributarán al correcto funcionamiento del CNBA.

El sistema a desarrollar a través de la presente investigación constituirá un componente de software para el envío seguro de la información, entre la aplicación de escritorio y el servidor del CNBA.

2.1.1 Descripción del negocio

RUP define en su primera fase de desarrollo la realización del modelo de negocio, con el objetivo de comprender el entorno del cliente y detectar las mejoras potenciales en los procesos de la organización. Cuando no es posible identificar claramente los procesos del negocio, RUP propone realizar un modelo de dominio, que es un subconjunto del modelo de negocio.

Un modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan los conceptos que existen o eventos que ocurren en el entorno en que trabaja el sistema. El modelo de dominio se describe específicamente mediante diagramas de clases,

utilizando el lenguaje de modelado UML.

2.1.2.1 Modelo de Dominio

El estudio del negocio realizado evidencia que los procesos que se llevan a cabo en el CNBA, no están bien definidos, por ello se procede a la realización del modelo de dominio.

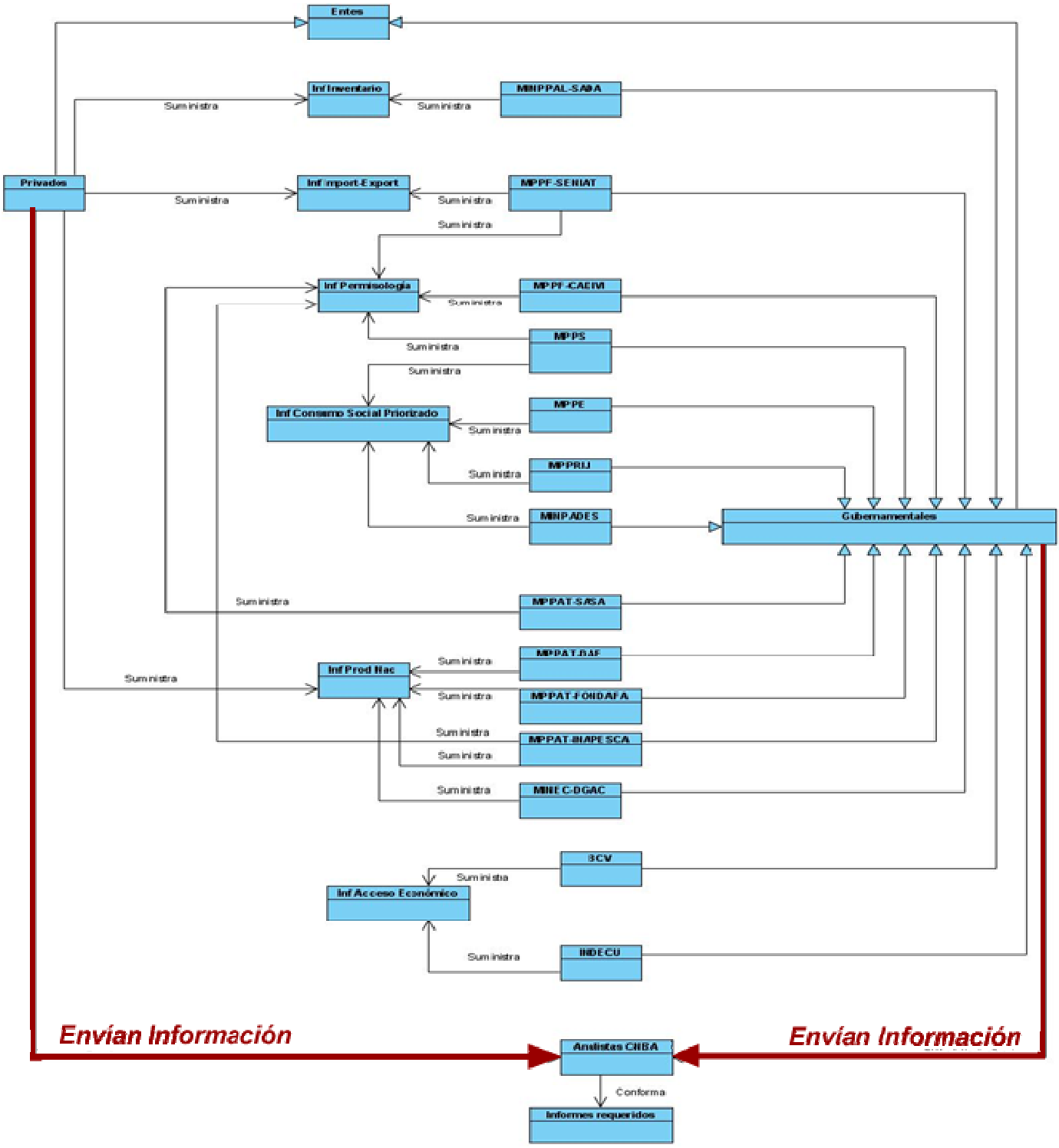


Fig. 5 Modelo del dominio

El proceso de envío de la información desde los entes gubernamentales y privados al CNBA es el centro de esta investigación. En la figura anterior se destacan los elementos del dominio (en rojo) relacionados directamente con el desarrollo del componente de software.

2.1.2 Modelación del sistema

2.1.2.1 Requisitos Funcionales y No Funcionales.

Los requisitos funcionales y no funcionales muestran las capacidades o condiciones que el sistema debe cumplir y las propiedades o cualidades que el producto debe tener, los cuales en la fase de construcción deben ser posibles de probar o verificar.

Requisitos Funcionales.

A continuación se muestran los requisitos funcionales de la aplicación de escritorio para la Solución Informática del CNBA.

R.1: Autenticar Usuario.
R.2: Cambiar Contraseña.
R.3: Convertir Volumen a TM.
R.4: Convertir unidad de precio a Bs/TM.
R.5: Certificar información.
- **R.5.1:** Capturar usuario.
- **R.5.2:** Capturar IP de salida.
- **R.5.3:** Chequear validez del IP de Salida.
- **R.5.4:** Enviar notificación.
- **R.5.5:** Capturar fecha.
- **R.5.6:** Mostrar usuario.
- **R.5.7:** Mostrar IP de salida.
- **R.5.8:** Mostrar fecha.

R.6: Imprimir Fichero.
R.7: Registrar Información de Certificación.
R.8: Configurar aplicación escritorio.
R.9: Cargar Información.
- **R.9.1:** Registrar fecha.
- **R.9.2:** Localizar fichero a cargar.
- **R.9.3:** Mostrar localización del fichero.

R.9.4: Validar fichero a cargar.

R.9.4.1: Validar fichero de Nivel de Ausencia y Escasez.

R.9.4.2: Validar fichero de Nivel de Inflación.

R.9.4.3: Validar fichero de Nivel de Acatamiento según BCV.

R.9.4.4: Validar fichero de Nivel de Acatamiento según INDECU.

R.9.4.5: Validar fichero de Nivel de Abastecimiento.

R.9.4.6: Validar fichero de Producción Nacional.

R.9.4.7: Validar fichero de Importaciones-Exportaciones.

R.9.4.8: Validar fichero de Permisología.

R.9.4.9: Validar fichero de Inventario.

R.9.5: Transformar fichero.

R.9.6: Cargar fichero.

R.9.7: Mostrar fichero cargado.

R.10: Visualizar el contenido del fichero.

R.11: Eliminar fichero.

R.12: Enviar fichero.

R.12.1: Controlar estado de envíos.

R.12.2: Notificar envío al servidor.

El componente de software de esta investigación cumplirá las funcionalidades descritas en los requisitos R.12.1 y R.12.2.

Requisitos No Funcionales.

Los requisitos no funcionales del sistema general se han adecuado a las cualidades particulares que debe tener el componente. Se detallan a continuación, agrupados según las diferentes categorías los requisitos no funcionales.

Apariencia o interfaz externa

El acceso al mismo y su configuración, no debe realizarse a través de interfaces gráficas de usuario.

Usabilidad

El componente puede ser acoplado a cualquier sistema de funcionamiento distribuido. Para su configuración y utilización es necesario tener conocimientos de Java.

Requerimientos de soporte

El componente debe estar documentado para garantizar el buen desempeño de los desarrolladores a la hora de interactuar con el mismo**.** (Ver manual del desarrollador en el anexo 5)

Debe realizar la configuración del cliente y el servidor para el proceso de envío de la información, dejando listos los elementos de red (IP, puerto de conexión), la tecnología a utilizar y los elementos de seguridad (ubicación de los certificados digitales).

Portabilidad

El sistema debe permitir ser ejecutado sobre los sistemas operativos Linux y Windows.

Seguridad

Los datos deben ser enviados cifrados y mediante el uso de túneles seguros a través de internet.

El sistema debe contar con protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

Se debe garantizar comunicaciones seguras entre los clientes y el servidor, encriptando todo el tráfico de información con la utilización de llaves negociadas, firma digital, algoritmos y protocolos.

Legales

Se estará usando para el desarrollo de la aplicación herramientas de software libre con licencia GNU/GPL.

Confiabilidad

El componente debe garantizar la integridad de los datos durante el envío. La información que fluirá en el mismo estará certificada y solo será la emitida por quienes la conforman, y manipulada por el personal autorizado.

Software

- Se deberá disponer para el uso del componente, del Sistema Operativo Windows 98 o superior, o cualquier distribución de Linux.
- Para el servidor de la aplicación el sistema operativo recomendado es Windows Server 2003 o superior o Linux.
- Se debe instalar TOMCAT como servidor web.
- Debe estar instalado el Java Runtime Environment (JRE) versión 1.6 o superior.

Hardware

Para el funcionamiento del componente se requiere de máquinas con los siguientes requisitos:

- Para las PC en las que se ejecutará la aplicación cliente: procesador Pentium 3 o superior, 256 Mb de RAM y al menos 1 Gb de capacidad del disco duro.
- El servidor debe tener las siguientes características: capacidad de disco duro superior a 80 GB, microprocesador Pentium IV superior a 2.0 GHz y como mínimo 1.0 GB de RAM.

Extensibilidad

Se debe lograr un diseño adaptable, con la capacidad de poder soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

Restricciones en el diseño y la implementación

El componente debe brindar la posibilidad de ser configurado según el cliente donde se instale. Su diseño debe ser flexible y adaptable a nuevos requisitos de tecnología.

El análisis y diseño de la aplicación será basado en la metodología RUP con el uso del lenguaje de modelado UML.

Se usará como herramienta CASE Visual Paradigm para el modelado de los artefactos que se generan en cada uno de los flujos de trabajo definidos por RUP.

Se usará como lenguaje de programación Java.

Se utilizará Eclipse (IDE de desarrollo) y Subversion (Control de versiones) durante el desarrollo.

En el desarrollo deben ser utilizados estándares como HTTP, HTML, XML, SOAP, UDDI.

Rendimiento

Al estar concebido para un ambiente distribuido, se trata de garantizar la rapidez de respuesta del componente ante las solicitudes de las aplicaciones y la velocidad de procesamiento de la información, para lo cual recibe los datos validados y los envía al servidor. De esta manera se logra una mayor velocidad de procesamiento y un mayor aprovechamiento de los recursos.

2.1.3 Diagrama de Casos de Uso del Sistema

El diagrama general de casos de uso describe la división del sistema en paquetes, así como la interacción de los actores con los casos de uso.

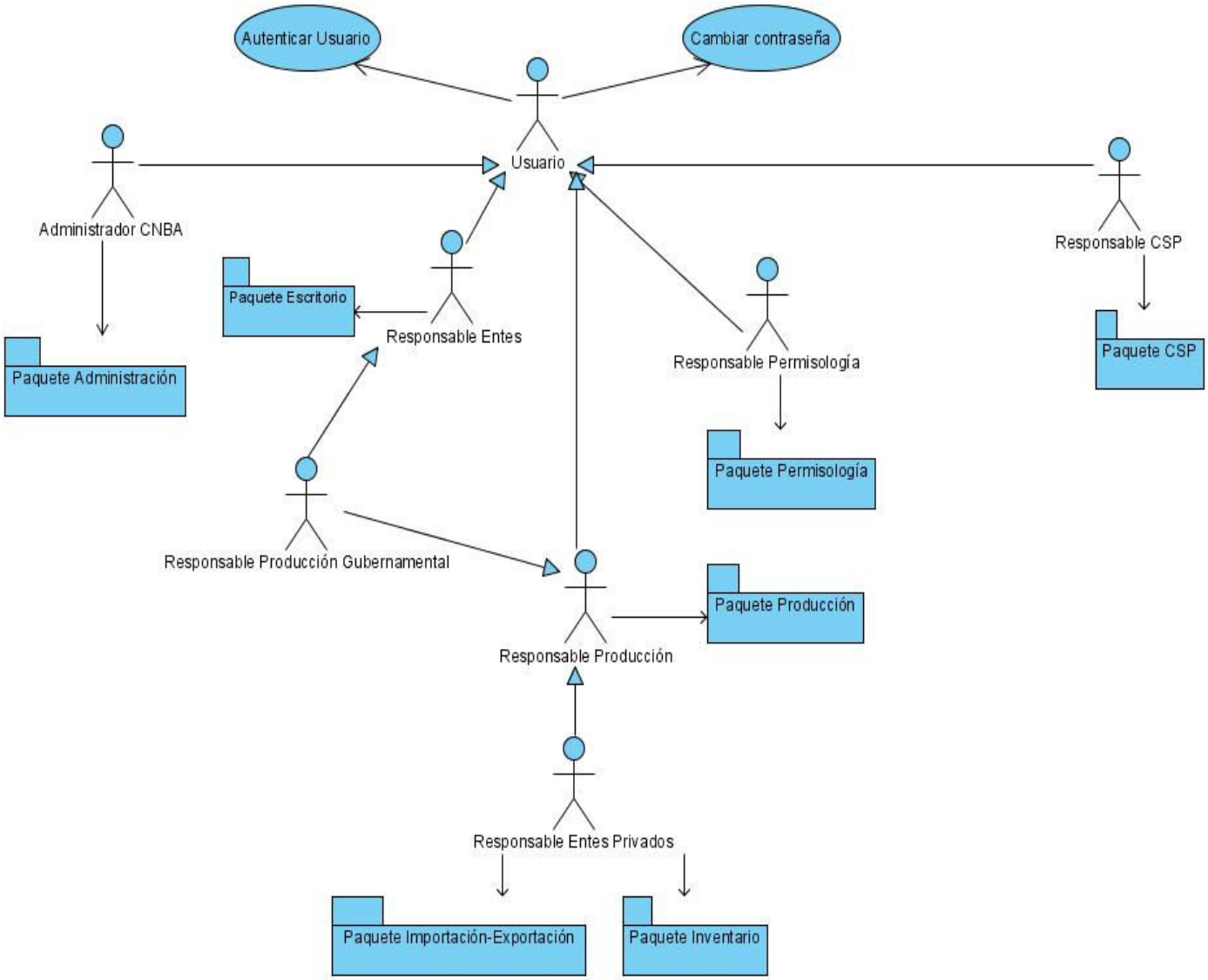


Fig. 6 Diagrama de casos de uso del sistema

Se expone en detalle el Diagrama de Casos de Uso del Paquete Escritorio en el que se identifica el CU Enviar Información que describe el proceso de envío de la información, razón fundamental del desarrollo del componente de software.

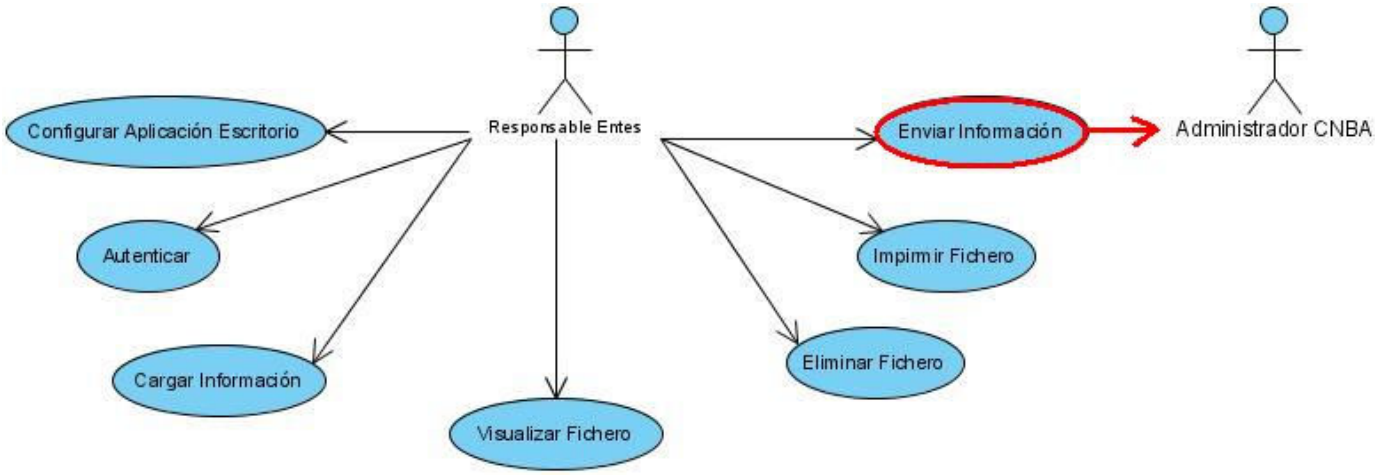


Fig. 7 DCUS del paquete Escritorio

La aplicación de escritorio estará presente en los entes que tienen la información digitalizada en los módulos:

- Permisología
- Producción
- Inventario
- Importación-Exportación
- Acceso Económico

Los módulos de Administración y Consumo Social Priorizado (CSP) utilizan la aplicación web.

2.1.4 Especificación de los Casos de Uso del Sistema.

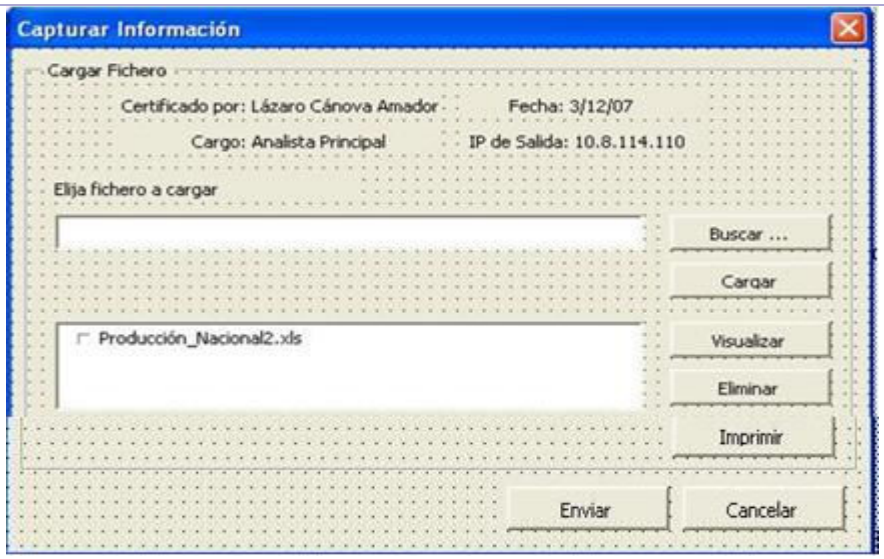
Se describe a continuación el CU Enviar Información correspondiente al Paquete Escritorio. Se detalla en el flujo normal de eventos los pasos a seguir para lograr el envío de la información al CNBA.

2.1.4.1 Caso de Uso "Enviar Información"

Caso de Uso:	Enviar Información.
Actores:	Responsable Entes. (Inicia), Administrador

Resumen:	El caso de uso se inicia cuando el Responsable Entes, indica enviar los ficheros cargados. Si se confirma el envío, el sistema verifica la validez del IP de salida, registra los datos de la certificación y se envían los ficheros, finalizando el caso de uso.
Precondiciones:	Debe haberse cargado al menos un fichero.
Referencia:	R.5.3, R.5.4, R.7, R.12.1,R.12.2
Prioridad:	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso se inicia cuando El Responsable Entes indica enviar la información.	2. El sistema muestra el mensaje de confirmación.
3. El Responsable Entes confirma el envío.	4. El sistema muestra una barra de proceso mostrando el avance del envío.
	5. Si se confirma el envío, el sistema verifica la completitud de la información.
	6. El Sistema registra los datos necesarios para la certificación.
	7. El sistema verifica que la información se esté enviando desde un IP válido.
	8. Si el IP es válido, el sistema envía todos los ficheros cargados a la aplicación de integración del CNBA para ser almacenados.
	9. El sistema envía un mensaje al administrador notificándole el estado de completitud de la información.
	10. El sistema muestra una barra de proceso completada en un 100%, indicando que ha sido enviada la información. Finaliza el caso de uso.
Prototipo de Interfaz	



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	5. Si no se confirma el envío, el sistema finaliza el caso de uso.
	8. Si el IP no es válido, el sistema envía todos los ficheros cargados marcándolos como "información en cuarentena" Ir a la acción 9.
Poscondiciones	Queda almacenado el fichero en el sistema.

2.2 DISEÑO

2.2.1 Aplicación de los patrones de diseño.

En el diseño del componente se utilizó el patrón **Factory** para la creación de la clase que brindará el servicio (según el mecanismo seleccionado para el envío), por dos vías: según los parámetros obtenidos del fichero de configuración utilizando inyección de dependencia internamente al invocar getCliente() y a través la construcción de un objeto de tipo *ConfiguracionEnvio* (Fig. 8) que se le pasa como parámetro al método getCliente(configuracion: ConfiguracionEnvio). Una invocación al método getCliente() de la clase ClienteFactory en cualquiera de los dos casos**,** devuelve la instancia de *ICliente* para la cual estará configurado el servicio de enviar información; de igual manera en el servidor se utiliza el patrón a través de la clase *ServidorFactory,* la cual devuelve una instancia de *IServidor* que permite la publicación del servicio del lado del servidor.

Como se muestra a continuación la clase *ClienteListener* interactúa con *ClienteFactory* y esta a su vez con *ICliente* para la construcción de objetos de su tipo. La interfaz *IClienteFactory* representa el comportamiento de la clase.

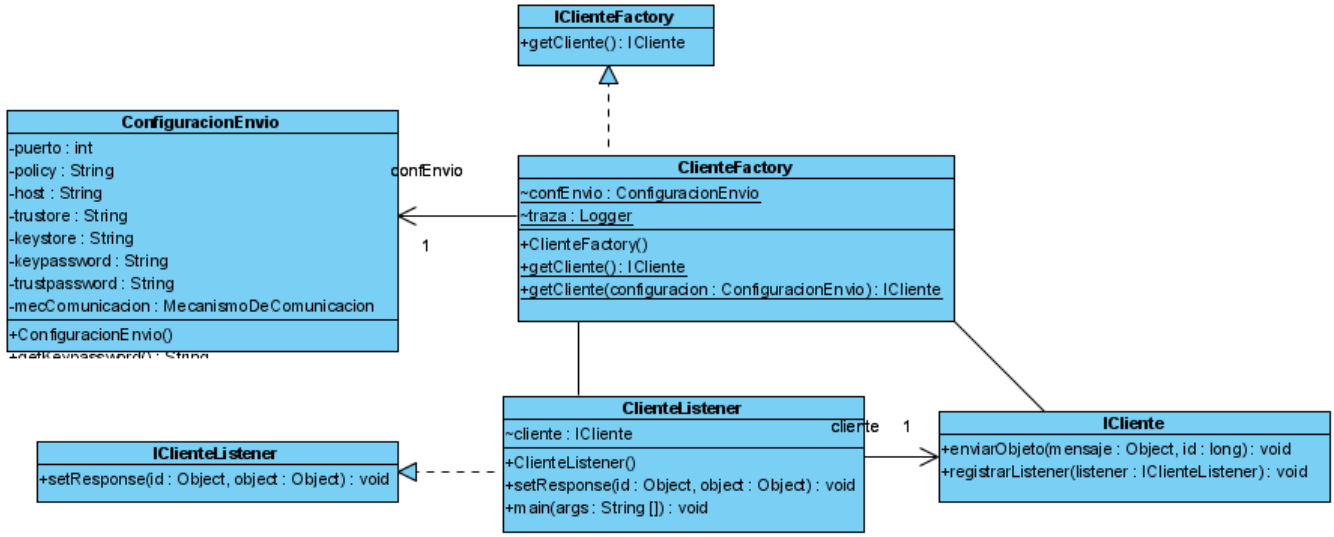


Fig. 8 Uso del patrón Factory

Como se mencionaba anteriormente el **patrón de inyección de dependencia** se utiliza para la configuración y acceso al servicio, removiendo de esta manera del código los elementos de configuración como: el puerto, la dirección del servidor, los certificados a utilizar, entre otros elementos (ver clase ConfiguracionEnvio en Fig.8). El proceso se realiza de manera transparente, pues se ejecuta

enteramente a través de Spring.[10]

Otro ejemplo de su utilización en la investigación se muestra como se inyecta el servicio establecido por la interfaz *IServidorService* a través de la clase *HttpInvokerServiceExporter* de Spring a la clase *ServidorServiceImpl*. De esta manera la publicación del servicio en el lado servidor, responde a través de la implementación de la clase *ServidorServiceImpl*. (ver Fig. 9)

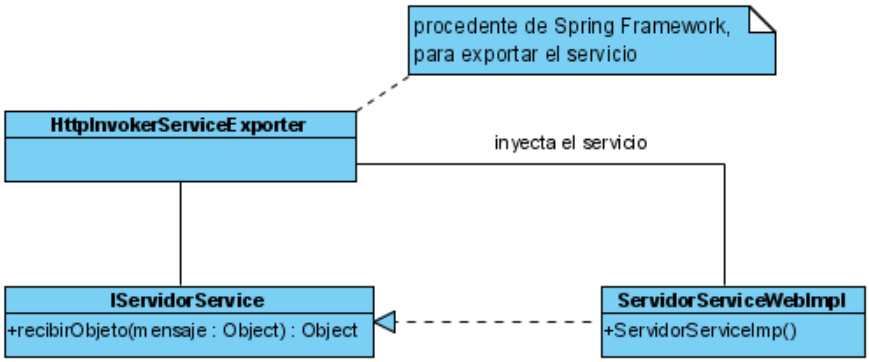


Fig. 9 Inyección de dependencia

El **patrón Experto** garantiza la lógica de las responsabilidades de las clases. Cada una realiza las acciones de las cuales es la experta en información, así por ejemplo la clase ServidorRMI (fig 10) tiene la responsabilidad de iniciar el servidor (iniciar()) y de recibir el objeto (recibirObjeto(Object mensaje)), pues es la que conoce los aspectos necesarios para realizar estas acciones. La **alta cohesión** está relacionada con lo planteado anteriormente y se utiliza por ejemplo en la creación de la clase Emisor (fig 11), la cual hereda de Thread y controla los hilos de envío de las clases Cliente, para abstraer a las mismas de ese comportamiento concurrente.

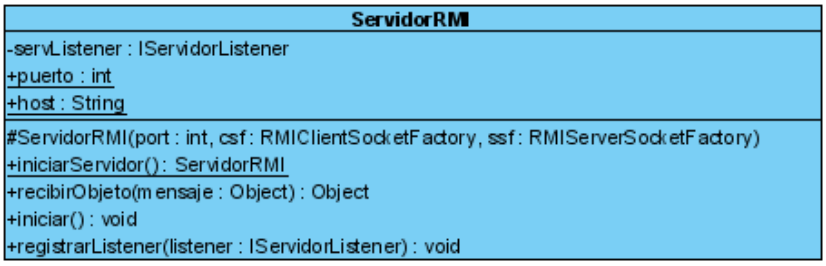


Fig. 10 Clase ServidorRMI

[10] BeanFactory es la clase que se encarga de construir los beans del contexto definido.

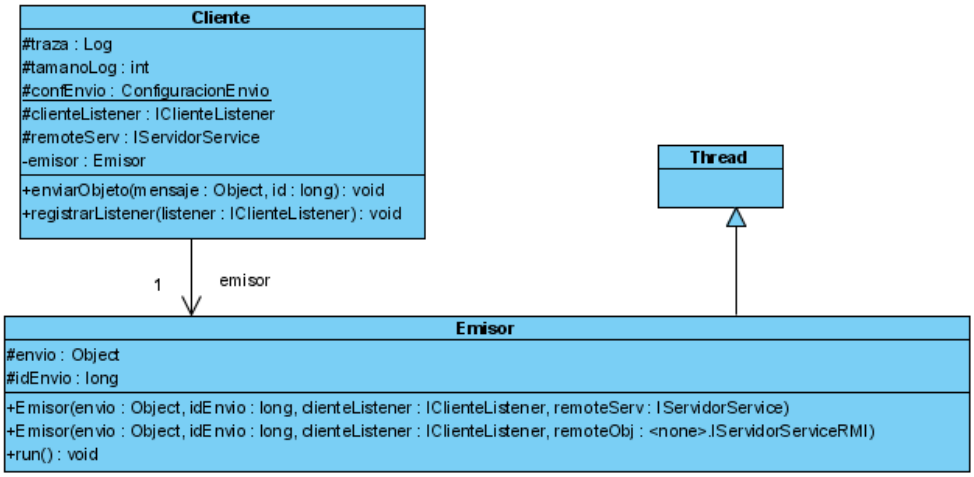


Fig. 11 Clase Emisor

Como elemento notable en el diseño es necesario destacar el **manejo de eventos en Java** (funcionamiento similar al patrón Observer, que basa su funcionamiento en el registro de objetos en escucha de peticiones), lo cual fue necesario al tratarse de un componente que requiere mecanismos para la integración con el resto del sistema. Las interfaces IClienteListener e IServidorListener definen los métodos que funcionan como enlace entre las acciones que se realizan internamente en el componente y el exterior, la primera acción que se realiza al efectuar un envío es registrar un "listener" (mediante la llamada al método registrarListener(IClienteListener listener) de la clase ICliente), que estará a la espera de la respuesta del servidor. Para ello es necesaria la implementación de las interfaces IClienteListener e IServidorListener en la parte del sistema cliente y servidor respectivamente.

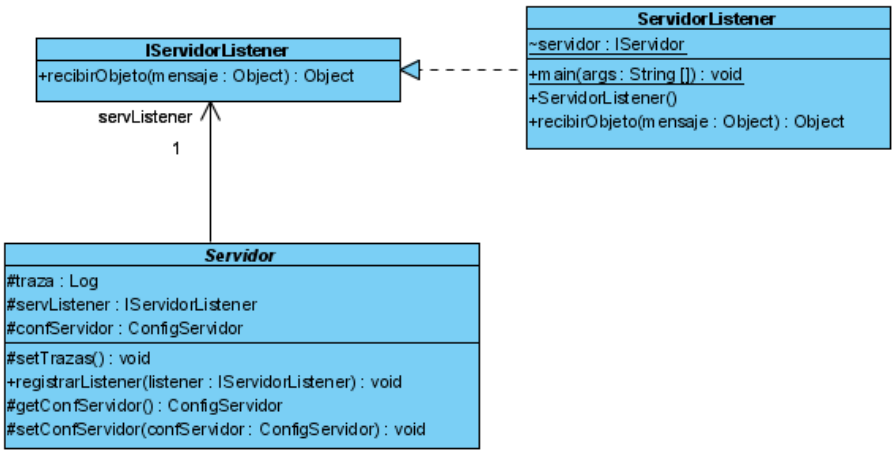


Fig. 12 Clases para el manejo de los eventos

2.2.2 Diagramas de clases del diseño

El diseño del componente para el envío de la información está básicamente dividido en tres paquetes:

Cliente: contiene todas las clases de la lógica del cliente para acceder al servidor. El subpaquete impl contiene la implementación de las clases Cliente que son del tipo ICliente.

Servidor: contiene las clases de la lógica del servidor para la publicación del servicio a brindar. El subpaquete impl contiene la implementación de las clases del tipo IServidor.

Comun: contiene las clases de ambas partes.

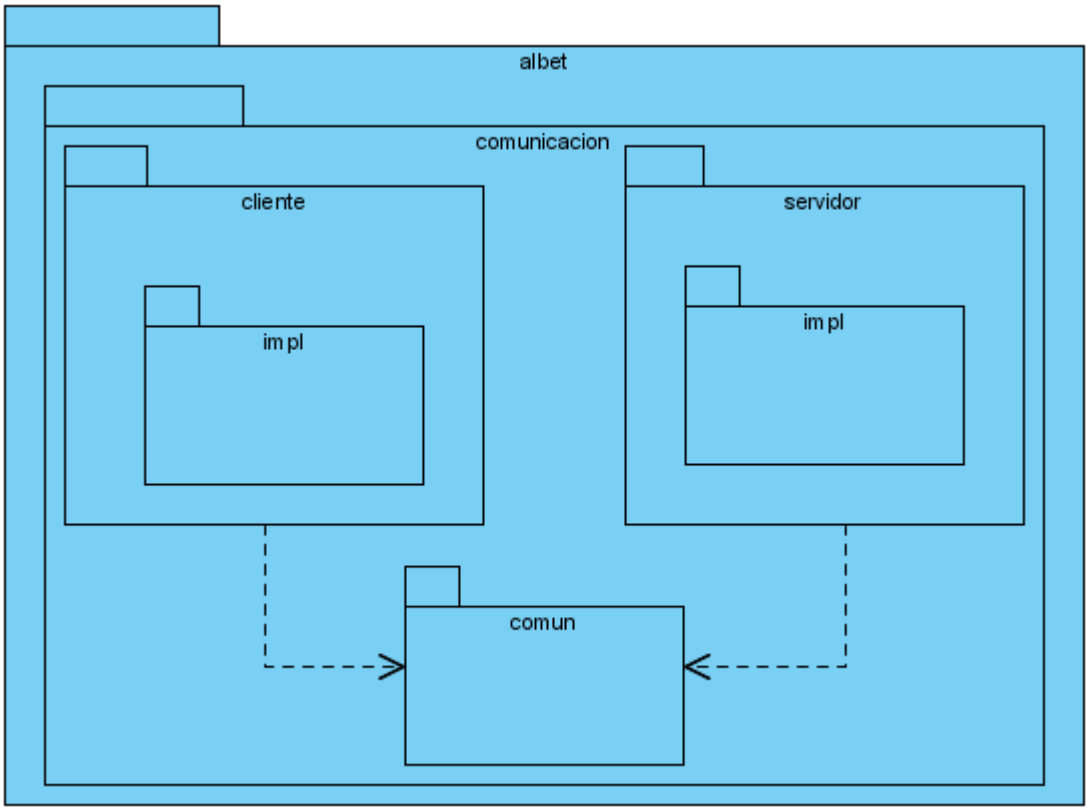


Fig. 13 Estructura de los paquetes del componente

A continuación se exponen los diagramas de clases correspondientes a los paquetes cliente y servidor en ese mismo orden, las clases comunes están representadas en el paquete comun, con las que se relacionan ambos paquetes.

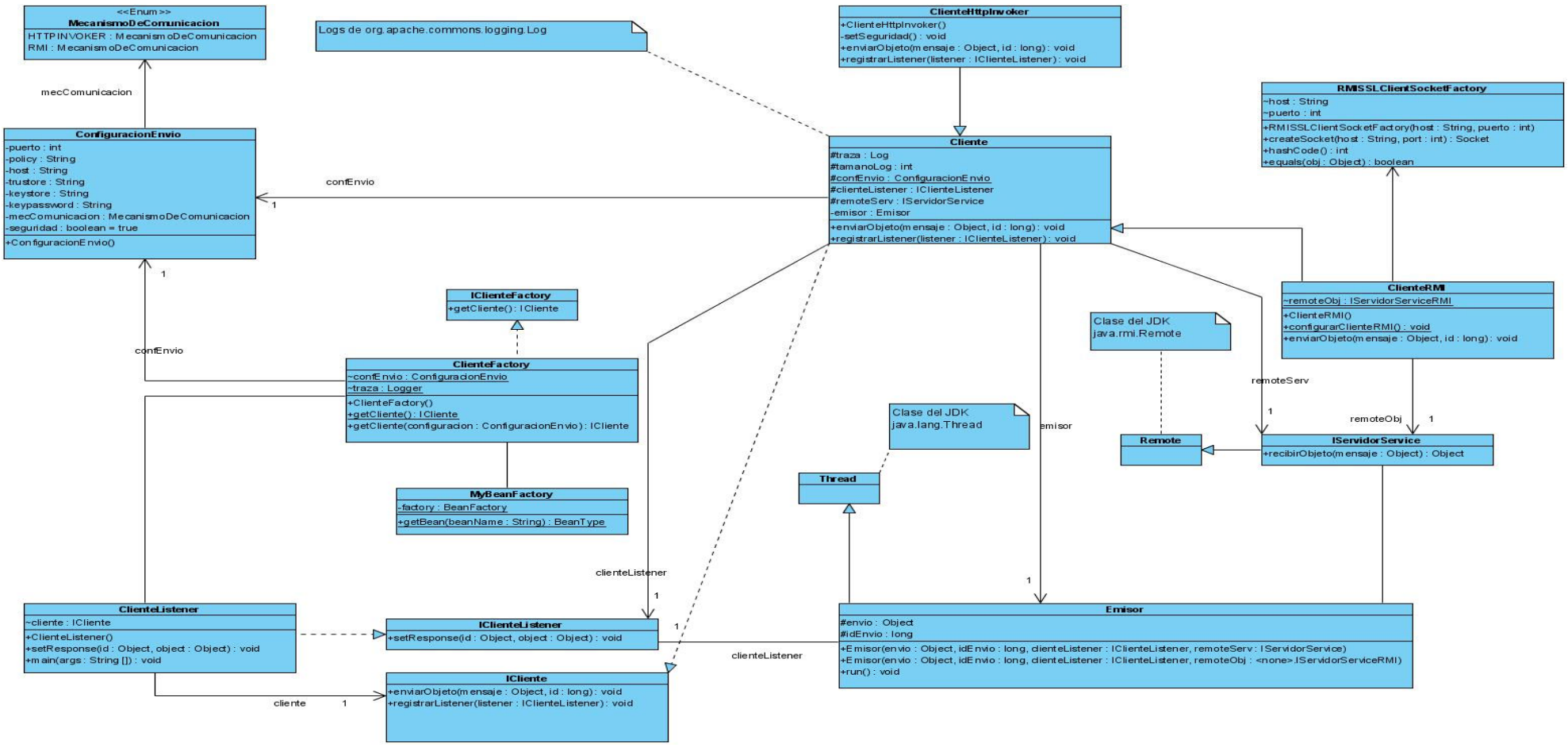


Fig. 14 Diagrama de clases. Paquete cliente

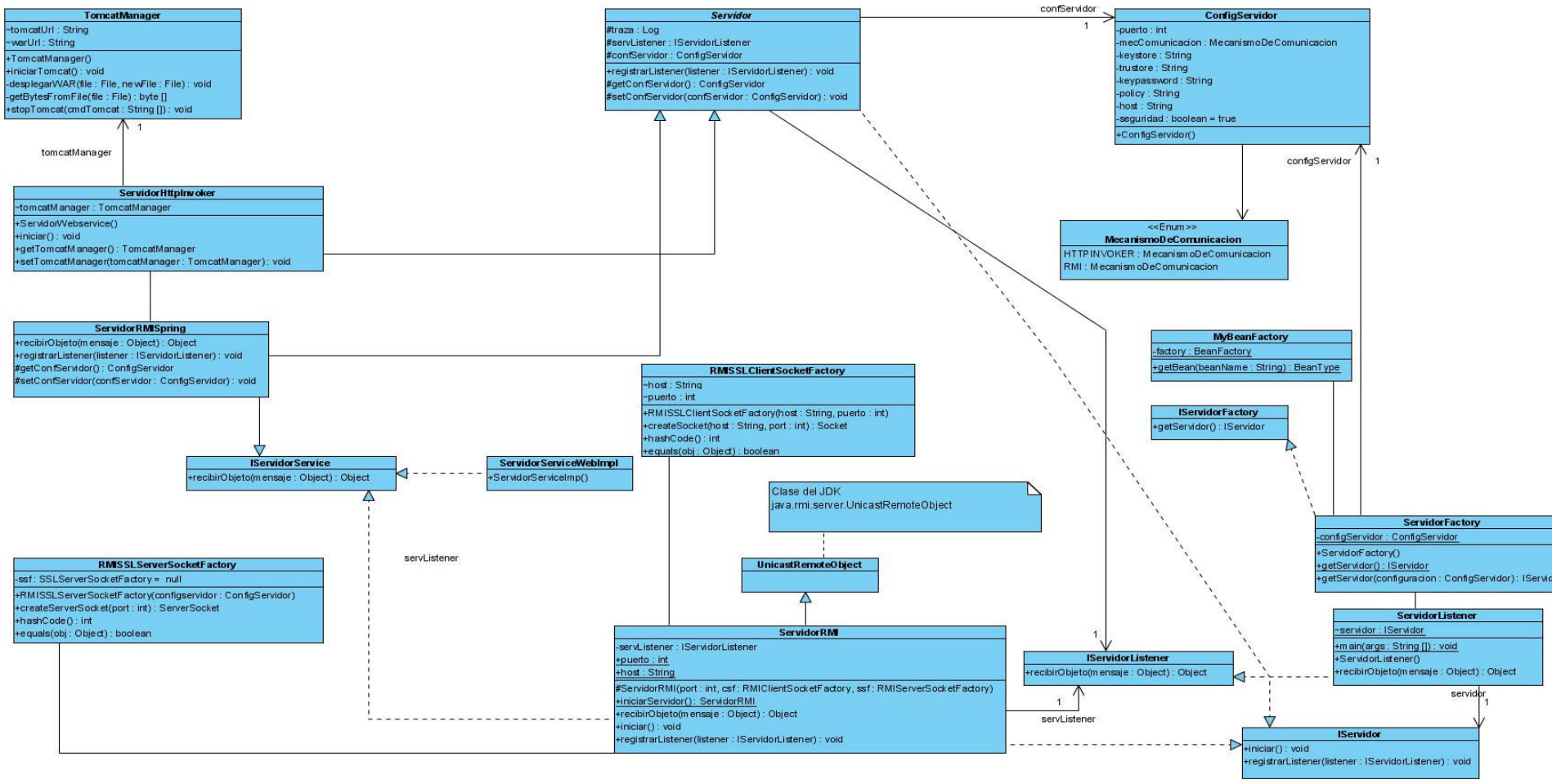


Fig. 15 Diagrama de clases. Paquete servidor

La mayor parte de las clases han sido tratadas en la sección anterior, mediante la descripción de la aplicación de los patrones. Para aclarar un poco más se describe a continuación el papel que tiene algunas de las clases que aparecen representadas en los diagramas:

- MyBeanFactory: Clase que construye los beans especificándole el contexto donde se encuentran los mismos.
- MecComunicacion: Enumerador que establece los tipos posibles de tecnologías a utilizar para la transferencia de datos.
- RMISSLClientSockectFactory y RMISSLServerSockectFactory: construyen los sockects para la comunicación por RMI de manera segura.
- ServidorRMISpring: ofrece el servicio RMI a través de Spring sin contemplar la seguridad.

2.2.3 Diagramas de Secuencia

Los diagramas de secuencia muestran en detalle cómo interactúan las clases y guían al programador durante la fase de implementación. A continuación se exponen los mismos:

La Fig. 16 expone la interacción entre las clases que componen al paquete servidor. La interacción entre las mismas concluye al iniciar el servicio y deja listo el servidor para peticiones de los clientes a través de la interfaz de servicio IServidorService.

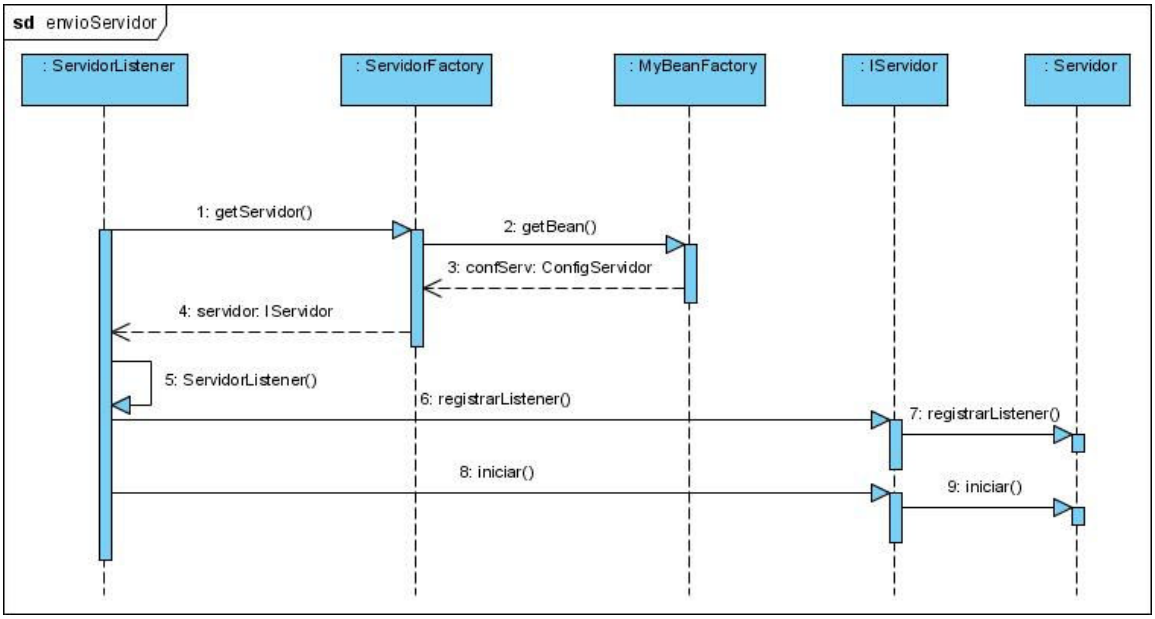


Fig. 16 Diagrama de interacción (iniciar servidor)

Se representa la clase Servidor como representante de la jerarquía de clases servidores posibles, la clase que accederá concretamente al servicio está en dependencia del mecanismo seleccionado para la transferencia.

La Fig.17 muestra el diagrama de secuencia para el caso en que el servicio a activar sea HTTP Invoker y por tanto requiera del uso de un servidor web. Aparece la clase *TomcatManager* que maneja el war[11] del servicio web y se encarga de su despliegue en el servidor de aplicaciones web Apache Tomcat y la clase *ServidorRMISpring* que se encarga de activar un servicio RMI utilizando Spring para la comunicación local entre el servidor web y la aplicación de integración.

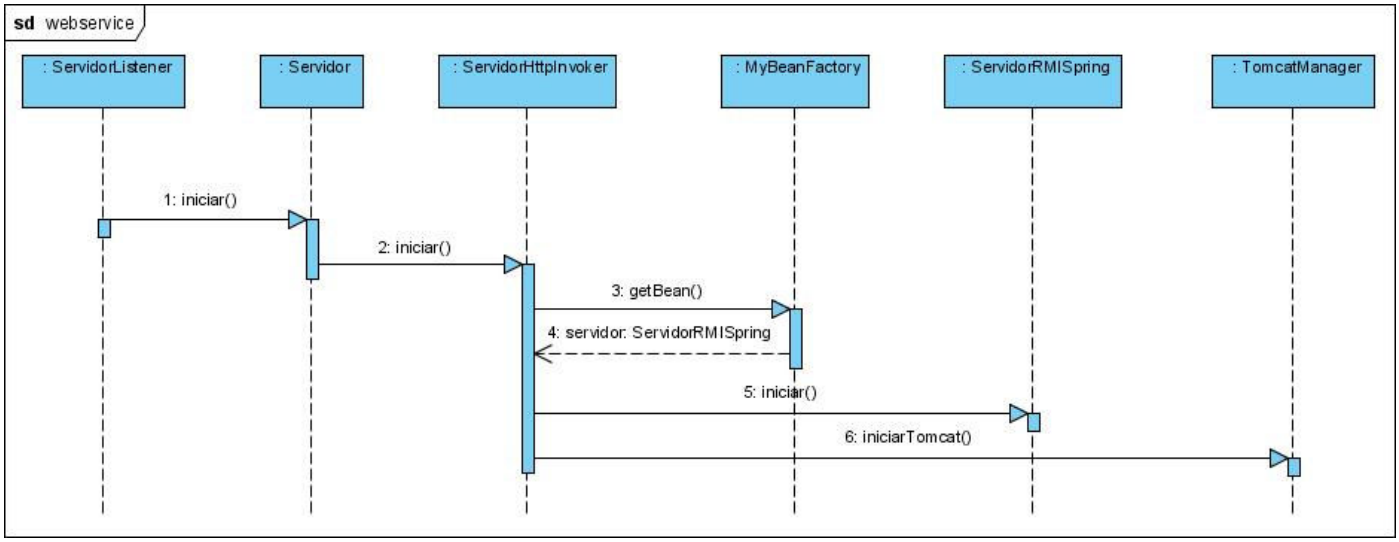


Fig. 17 Diagrama de interacción (desplegar servicio HTTP Invoker)

El próximo diagrama (Fig. 18) muestra la interacción entre las clases que componen el paquete cliente, que hace posible el acceso al servicio de transferencia que brinda el servidor a través de la interfaz IServidorService. Se representa la clase Cliente como superclase de ClienteRMI y ClienteHttpInvoker, porque ambas proceden de la misma manera, y realmente el diseño esta realizado de forma tal que no sea necesario conocer que clase es la que maneja el servicio realmente.
Luego del acceso a IServidorService se muestra los pasos que se siguen en el servidor para dar respuesta al cliente.

[11] Archivo generado para el despliegue de una aplicación web implementada en java.

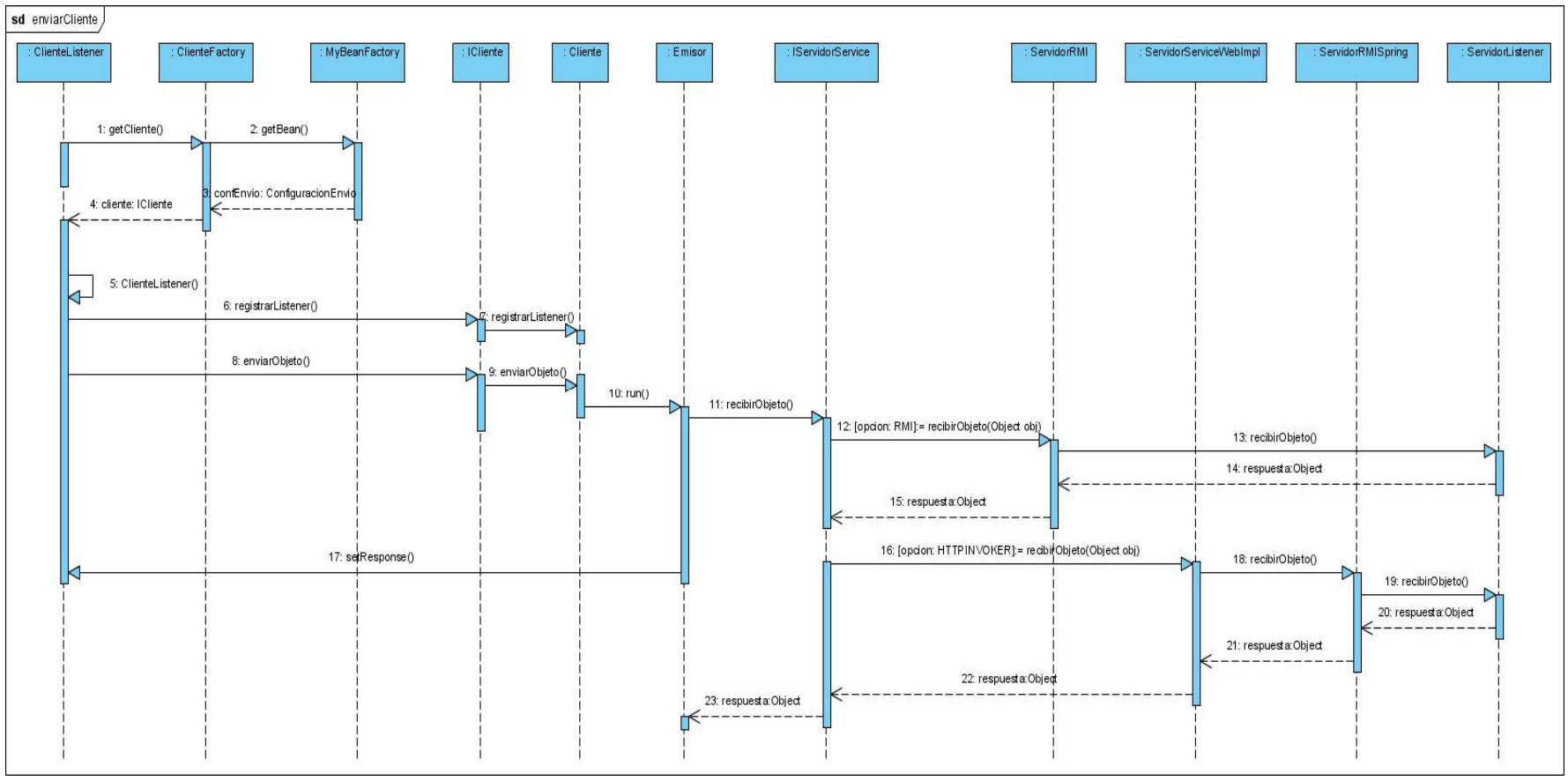


Fig. 18 Diagramas de interacción (acceso al servicio)

2.2.4 Diagrama de despliegue

El diagrama de despliegue del sistema muestra los nodos “Servidor CNBA” y “Estación de trabajo con la aplicación cliente” en los cuales se despliega el componente de software para establecer la transferencia entre los mismos, siguiendo el precepto de la distribución lógica de las aplicaciones en tres capas como se ha definido en la Solución del CNBA.

Entre los nodos en los que se despliega el componente se representa en enlace que puede ser por RMI o por HTTPS.

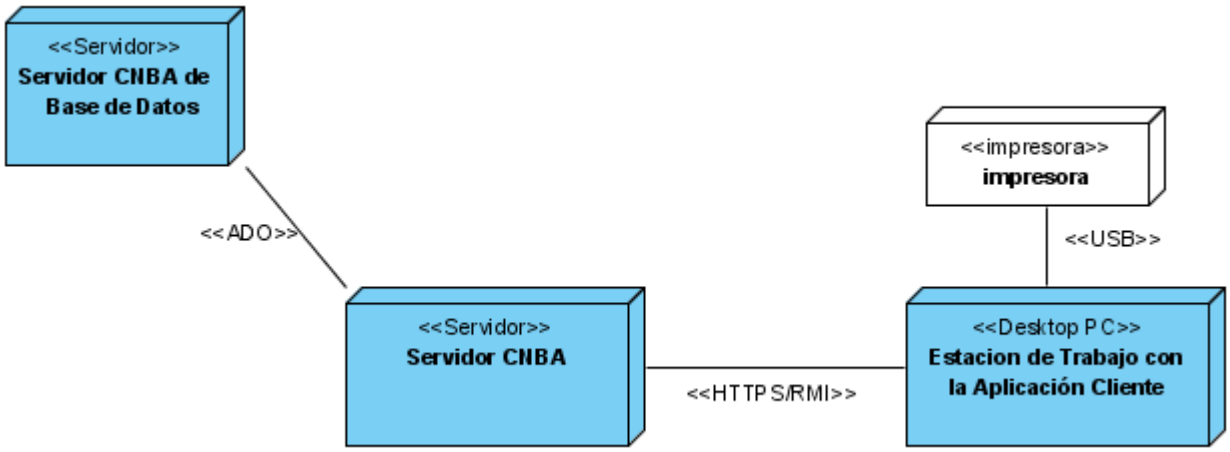


Fig. 19 Diagrama de despliegue del sistema

CONCLUSIONES

En el presente capítulo fueron elaborados y descritos brevemente algunos de los artefactos propuestos por RUP para el desarrollo de software, tales como el modelo de dominio, definición de los requisitos funcionales y no funcionales, diagrama de casos de uso del sistema y la descripción textual de los casos de uso del sistema. Se aborda además, el diseño del componente, los diagramas de clases, de interacción y despliegue. Se destaca en cada caso como se inserta el componente de software a desarrollar en la Solución Informática para el Centro Nacional de Balance Alimentario, específicamente en la aplicación de escritorio e integración.

CAPÍTULO 3
IMPLEMENTACIÓN DEL COMPONENTE

En este capítulo se describe cómo fue implementado el componente. Se presenta el diagrama de componentes, que muestra la distribución física de los componentes para la implementación. Se describen los fragmentos relevantes del código y los elementos esenciales que deben conocer los desarrolladores para utilizar con el componente. Además se describen las pruebas exploratorias, funcionales y de sistema realizadas para la validación del resultado obtenido en la investigación.

3.1 DIAGRAMA DE COMPONENTES

UML define el diagrama de componentes para la representación de un sistema de software en componentes físicos (por ejemplo archivos, cabeceras, módulos, paquetes) y establecer las dependencias entre estos componentes.

En la Fig. 20 se expone el diagrama de componentes que interactúan en el desarrollo del componente de software, en los componentes “comunicación-servidor.jar”, “comunicación-cliente.jar” y “comunicación-servweb.war” se ubican las clases (.java) de los paquetes ubicados en el servidor y cliente, como sus nombres lo indican. El resto de los componentes representados constituyen los archivos de configuración (clienteContext.xml, servidorContext.xml y comunicacionRefFactory.xml), las librerías utilizadas (agrupadas en el paquete lib), los archivos necesarios para establecer la seguridad con SSL (representados en los paquetes segcliente y segservidor) y en el paquete logs, los archivos necesarios para garantizar la trazabilidad en el envío (Trazas.log para almacenar las trazas y log.properties para establecer el nivel de trazabilidad). Se muestran las interfaces con las que interactúa el sistema que requiera la utilización del componente, `IServidorService` es la interfaz de servicio para la comunicación entre el componente cliente y servidor.

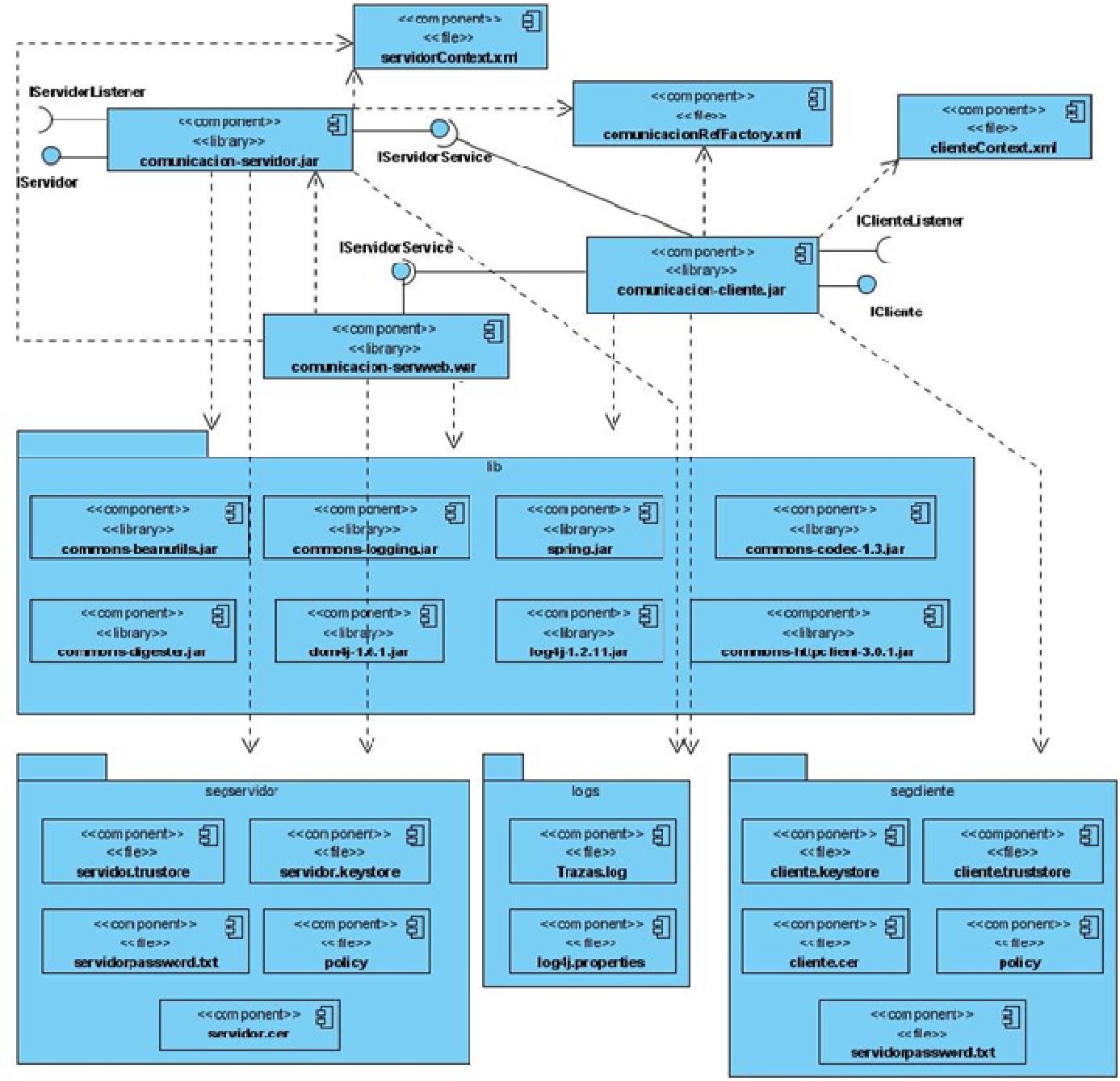


Fig. 20 Diagrama de componentes

Para brindar una idea clara del uso del componente en la solución informática del CNBA, se muestra a continuación los componentes más significativos de la Vista de Implementación del sistema del CNBA, en el cual la aplicación de escritorio y la aplicación de integración (componente validador e intérprete

XML) implementan la interfaz *IClienteListener* e *IServidorListener* y utilizan la interfaz *ICliente* e *IServidor* respectivamente.

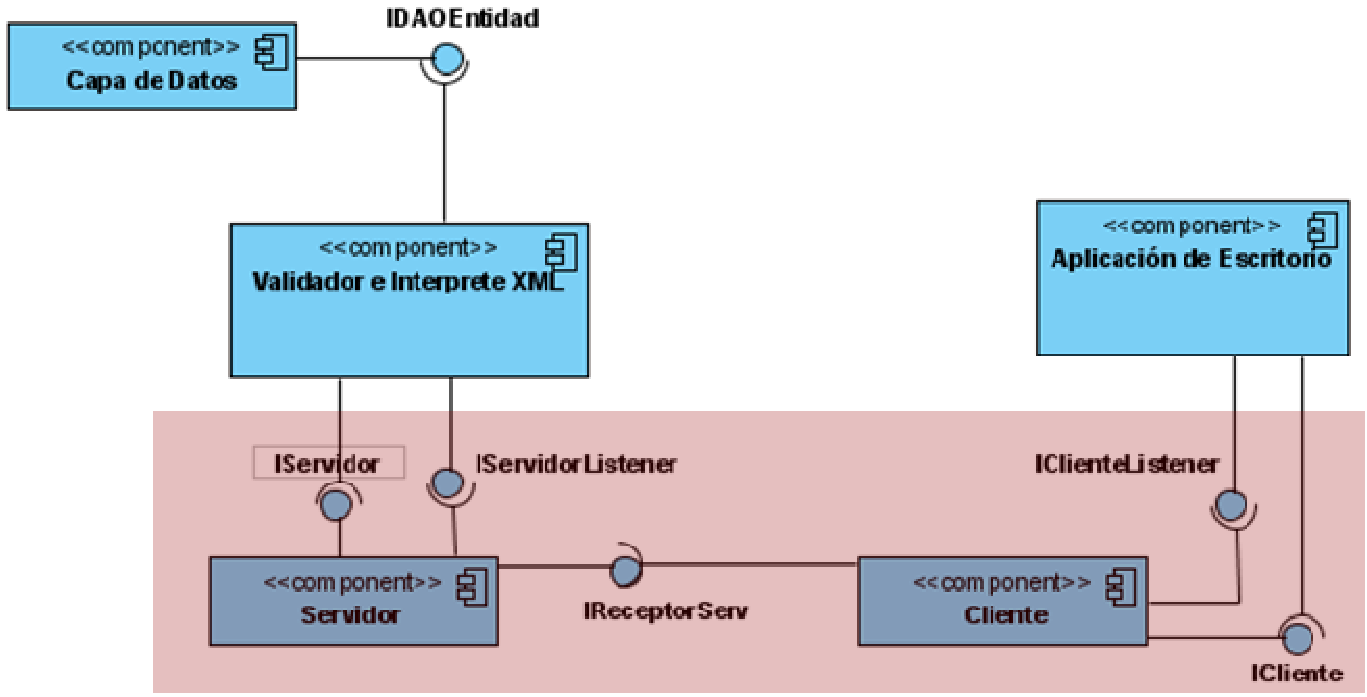


Fig. 21 Relación de los componentes en el sistema

3.2 FRAGMENTOS DE CÓDIGO

En esta sección se exponen los fragmentos de código críticos en el desarrollo del componente, los archivos de configuración que se utilizan para el envío o recibo de la información en el cliente y servidor respectivamente. Haciendo énfasis en los elementos a utilizar para la integración del componente a sistemas externos.

3.2.1 Cliente

3.2.1.1 Configuración del cliente

La configuración del cliente para el envío de la información se realiza a través del archivo XML que se muestra a continuación; el mismo contiene los atributos necesarios para la conexión al servidor (Ver clase *ConfiguracionEnvio*), la dirección de los certificados digitales, la dirección del archivo que establece los permisos en java (policy) y el mecanismo de comunicación que se utilizara para el envío. Este archivo es para el uso de los desarrolladores que deseen utilizar el componente, de esta manera

todos los detalles de configuración son removidos del código, lo que hace más accesible su utilización. Para un desarrollador cambiar la tecnología de comunicación para el envío, bastaría con cambiar la propiedad "mecComunicacion" y colocar HTTPINVOKER en lugar de RMI en este caso y se abstrae del conocimiento empleado en la implementación de las tecnologías.

```
<bean id="configuracion"
class="albet.comunicacion.cliente.ConfiguracionEnvio">
            <property name="puerto">
               <value>5801</value>
            </property>
           <property name="host">
               <value>10.7.7.12</value>
            </property>
           <property name="policy">
               <value>policy</value>
            </property>
           <property name="trustore">
               <value>segcliente/client.truststore</value>
            </property>
            <property name="keystore">
               <value>segcliente/client.keystore</value>
            </property>
            <property name="keypassword">
               <value>segcliente/password.txt</value>
            </property>
            <property name="mecComunicacion">
               <value>RMI</value>
            </property>
        </bean>
```

3.2.1.2 Uso del componente en el cliente

Para acceder al envío de la información es suficiente con cuatro pasos:

1. Crear una instancia del tipo *ICliente* a través de la clase *ClienteFactory* invocando su método estático `getCliente()`, que internamente adquiere los parámetros para la creación del objeto del XML de configuración.
2. Crear una instancia de la clase *IClienteListener* y pasarla como parámetro al método `registrarListener(IClienteListener listener)`, dicha instancia estará en espera de los eventos provocados por la respuesta del servidor. Con el objetivo de permitir el envío simultáneo de varios objetos se crea una instancia del listener para cada petición al servidor.
3. Invocar el método `enviarObjeto(Object obj; Object id)` para el envío de la información, es válido aclarar que es posible el envío de cualquier objeto que tenga como clase base `java.lang.Object`.
4. Implementar la interfaz *IClienteListener* (Fig.22). El método `setResponse(Object id, Object object)` es invocado cuando llega la respuesta de un envío y el programador de la aplicación cliente debe asumir su implementación.

Fig. 22 Interfaz IClienteListener

Ej:

```
public void setResponse(Object id, Object object) {
      System.out.println("El envio con id " + id + "devolvió" + object);
}
```

A través de este método se imprime en consola la respuesta del servidor. Es válido aclarar que es necesario que ambas partes involucradas en la transferencia conozcan el tipo de objeto que envían en las peticiones y respuestas.

A continuación se muestra un fragmento de código como ejemplo a las acciones explicadas anteriormente (los círculos amarillos se corresponden con los pasos expuestos).

```
static ICliente cliente;
static Log traza;

public ClienteListener()
```

```
    {
    }

    public static void main(String[] args)
    {
        // TODO Auto-generated method stub
        traza = LogFactory.getLog(ClienteListener.class);

        try
        {
            cliente  = ClienteFactory.getCliente();
            traza.debug("Creado objeto cliente para envio" +
cliente);
            cliente.registrarListener(new ClienteListener());
            cliente.enviarObjeto("datos", 50);
        }
        catch (Exception e)
        {
            traza.error(e.getStackTrace());
            traza.fatal("No ha sido posible la configuración del
cliente para realizar el envío");

        }
    }
```

1

2

3

Finalmente es necesario el tratamiento de las excepciones que puedan ser lanzadas en el proceso de envío, ya que el componente relanza al exterior todas las excepciones que no puedan ser controladas internamente y que no sean fatales en el funcionamiento del mismo, por ello es necesario que este tratamiento sea de acuerdo al sistema que utilice el componente.

3.2.1.3 Realizando el envío internamente

La respuesta del servidor puede ser más o menos lenta según las condiciones de la red sobre la que se encuentre el servicio desplegado, por ello previendo la demora en dichas respuestas se ha

desarrollado la clase *Emisor* (que extiende de *Thread*) para manejar los envíos efectuados como hilos independientes y de esta manera permitir al usuario que pueda realizar otras acciones mientras culmina el proceso de envío de la información.

En el cuerpo del método `enviarObjeto(Object obj, Object id)` lo que se realiza fundamentalmente es el inicio del hilo que maneja el envío:

```
emisor = new Emisor(mensaje, id, clienteListener, remoteServ); //
inicialización del emisor
emisor.start(); // inicio del hilo de ejecución que internamente
invoca     //al método run()
```

El método `run()` de la clase Emisor invoca al método de servicio publicado en el servidor.

```
public void run()
{

     Object result = null;

          try
          {
               result = remoteServ.recibirObjeto(envio);
               traza.debug(result);
          }
          catch (Exception e)
          {
               traza.error(e.getMessage());
               clienteListener.setResponse(idEnvio, e);
          }

     if (result == null)
     {
          traza.info("La respuesta del servidor fue null");
     }
     else
     {
          traza.info("La respuesta del servidor se ha procesado
          correctamente");
     }
     clienteListener.setResponse(idEnvio, result);
}
```

3.2.2 Servidor

3.3.2.1 Configuración del servidor

La configuración del servidor es similar a lo explicado anteriormente en el epígrafe 3.4.1.1, se realiza a través de un XML similar, que contiene de igual manera la configuración necesaria para la publicación del servicio. De igual manera el desarrollador gana con la fácil exportación de un servicio ya sea por RMI o HTTPINVOKER.

3.2.2.2 Publicando el servicio

Para la publicación del servicio y la puesta en marcha del servidor no es necesario tener conocimientos acerca de la tecnología que va a utilizar el componente para publicar el servicio, solo es necesario los siguientes pasos para la integración del componente al sistema:

1. Crear una instancia del tipo *IServidor* a través de la clase *ServidorFactory* invocando su método estático `getServidor()`. La creación del objeto se realiza a través de los elementos de configuración establecidos en el XML de configuración. (Ver anexo fichero servidorContext.xml)
2. Crear una instancia de la clase *IClienteListener* y pasarla como parámetro al método `registrarListener(IServidorListener listener)`, dicha instancia estará en espera de los eventos provocados por las peticiones de los clientes.
3. Invocar el método `iniciar()`, que arranca el servicio para el servidor específico creado como instancia de la clase base *IServidor*.
4. Implementar la interfaz *IServidorListener* (Fig. 23), el método `recibirObjeto(Object mensaje)` es invocado cuando llega la petición al servidor, el programador de la aplicación servidor debe asumir su implementación de acuerdo a lo requiera hacer con el objeto recibido y a su vez devuelve la respuesta del servidor una vez procesada la petición.

Fig. 23 Interfaz IServidorListener

Ej:

```
public  Object recibirObjeto(Object mensaje)
{
     return "La respuesta del Servidor después de procesar el
            objeto es correcta";;
}
```

A continuación se muestra un fragmento de código como ejemplo a las acciones explicadas anteriormente (los círculos amarillos se corresponden con los pasos expuestos).

```
public static void main(String[] args)
{
      try
      {
            servidor = ServidorFactory.getServidor();   1
            servidor.registrarListener(new ServidorListener());   2
            servidor.iniciar();   3
      }
      catch (Exception e)
      {
            System.out.println("Servidor mal construido");
            e.printStackTrace();
      }
}
```

Es fundamental el tratamiento de las excepciones que se pueden originar en la invocación de los métodos del componente, para el desarrollador que lo utilice constituye una caja negra por lo que es necesario el tratamiento de las excepciones que puedan ser lanzadas al exterior durante la ejecución.

3.3 SERVICIO A TRAVÉS DE REMOTE METHOD INVOCATION

RMI es un protocolo fuerte y reconocido porque soporta una completa serialización de objetos, que es importante cuando se usa modelos de datos complejos que necesitan ser serializados sobre red. Es más recomendado en caso de que donde se vaya a desplegar no haya problemas con el paso de los firewall.

Aunque Spring brinda soporte para la publicación de servicios RMI tradicionales, no presenta una madurez aceptable en cuanto a la seguridad, por ello en la investigación se ha implementado RMI de la manera tradicional de Java.

Para el uso de RMI es necesario destacar dos cuestiones esenciales:

1. La interfaz de servicio debe extender de la interfaz *java.rmi.Remote.*
2. La clase servidora debe extender de la clase *UnicastRemoteObject* e implementar la interfaz de servicio, en este caso *IServidorService*.

El resto no es complejo, pues solo es necesaria la creación del registro y vincular el objeto que ejecutará el servicio remoto en el servidor.

```
Registry registro = LocateRegistry.createRegistry(puerto);
obj = new ServidorRMI();
registro.bind("Objeto", obj);
```

Para acceder en el cliente es necesario conocer el IP del servidor, y estar autorizado al mismo, con sus certificados digitales.

En la clase cliente es necesario localizar el registro y crear un objeto del tipo *IServidorService* para acceder al método del servidor como si fuera local.

```
remoteServ = (IServidorService) registry.lookup("Objeto");
remoteServ.recibirObjeto(objeto);
```

3.4 SERVICIO A TRAVÉS DE HTTP INVOKER

La estrategia remota de Spring, HTTP invoker es una buena opción si se necesita el transporte de los datos sobre HTTP o HTTPS, además acude a la serialización de los objetos de java. Comparte la infraestructura básica de RMI, pero utilizando HTTP como transporte, lo que elimina el problema de RMI con los firewall.

El servicio HTTP Invoker requiere ser desplegado en un servidor de aplicaciones web para java, en este caso en el Tomcat, el componente “comunicacion-servweb.war” representa el compilado de la aplicación web que debe desplegarse. El despliegue del mismo se puede realizar de forma manual o mediante el archivo de configuración (en este caso indicar “true” en la propiedad despliegue del archivo). En el mismo se debe escribir la dirección del TOMCAT_HOME y la ruta de la ubicación de la aplicación servicio a desplegar.

```
          <bean id="tomcat"
class="albet.comunicacion.servidor.TomcatManager">

             <property name="despliegue">
               <value>false</value>
             </property>

                <property name="tomcatUrl">
               <value>/opt/apache-tomcat-6.0.14/</value>
             </property>

              <property name="warUrl">
```

```
        <value>/home/administrador/Escritorio/comunicacion-
        servweb.war</value>
          </property>

      </bean>
```

La clase `HttpInvokerServiceExporter` de Spring es la encargada de exportar el servicio del lado del servidor, es necesario indicarle la propiedad `service` que hace referencia al bean `remoteservice`, para asignarle la clase que sostendrá el servicio, en este caso *ServidorServiceImpl* es la encargada de la soportar la lógica del servicio a través de la implementación de la interfaz *IServidorService* la propiedad `serviceInterface` (para indicarle la interfaz del servicio que brinda). Veamos:

```
<bean name="/xmlservice"
class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExpo
rter">
        <property name="service" ref="remoteService" />
        <property name="serviceInterface"
    value="albet.comunicacion.comun.IServidorService" />
</bean>
<bean id="remoteService"
class="albet.comunicacion.servidor.ServidorServiceImpl" />
```

Del lado cliente Spring brinda la clase `HttpInvokerProxyFactoryBean` para el acceso al servicio, de similar manera es necesario indicarle las propiedades `serviceUrl` (URL del servicio al que se desea acceder), `serviceInterface` (la interfaz del servicio a la que accede) y `httpInvokerRequestExecutor` (para lograr el acceso a servicio a través de HTTPS, la cual es solo necesaria en el caso de que se requiera el acceso seguro al servicio, como en este caso).

```
<bean id="httpWebInvoker"
class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactor
yBean">
        <property name="serviceUrl">
```

```
            <value>

      https://10.7.7.12:8443/Comunicacion_Servidor/remoting/xmlservice
            </value>
            </property>
            <property name="serviceInterface">
            <value>albet.comunicacion.comun.IServidorService</value>
            </property>
            <property name="httpInvokerRequestExecutor">
            <bean
class="org.springframework.remoting.httpinvoker.CommonsHttpInvokerRequ
estExecutor"/>
            </property>
</bean>
```

El bean `remoteHttpInvoker` representa a la clase *ClienteHttpInvoker* que es la clase mediante la cual se accede al servicio, para la que se ha definido una propiedad `remoteServ` (del tipo de la interfaz *IServidorService*, que en este caso toma el valor del bean `httpWebInvoker` de esta forma se "apropia" de la información necesaria para acceder a los objetos remotos publicados en el servidor, explicado anteriormente) y otra propiedad `confEnvio` a través de la cual se hace referencia al bean `configuracion`, que contiene los elementos necesarios para acceder al servidor.

```
<bean id="remoteHttpInvoker"
class="albet.comunicacion.cliente.impl.ClienteWebservice">
            <property name="remoteServ">
            <ref bean="httpWebInvoker" />
            </property>
            <property name="confEnvio">
            <ref bean="configuracion" />
            </property>
</bean>
```

El servicio establecido permite la transferencia de todo tipo de objetos, pero vale destacar que en el caso del servicio a través de HTTP Invoker es necesario agregar las clases de las cuales sea necesario enviar objetos, para que el mecanismo de serialización - deserialización de RMI funcione correctamente, por ejemplo: la aplicación de integración devuelve ante alguna petición una instancia de la clase *Respuesta* que al no ser una clase típica del JDK no es soportada de manera natural. La solución en estos casos consiste en la inclusión de estas clases en el CLASSPATH de la aplicación.

3.5 IMPLEMENTACIÓN DE LA SEGURIDAD

La seguridad se ha garantizado a través del uso de SSL, bajo los siguientes principios:

- El servidor tiene una llave privada (almacenada en el keystore), una pública que es conocida por los clientes que requieren intercambiar información con el mismo y un almacén de llaves públicas de todos los clientes autorizados (truststore).
- Los clientes por su parte tienen igualmente una llave privada y una llave pública, deben incluir la llave pública del servidor entre sus servidores de confianza.
- Las llaves públicas se emiten utilizando certificados digitales de ambas partes.

Técnicamente es necesario, para el funcionamiento de SSL, incluir al CLASSPATH de java los elementos de seguridad para la ejecución tanto del cliente como del servidor (ver código citado). Además de transformar las policy establecidas en JAVA_HOME para que existan los permisos necesarios para el acceso y la exportación del servicio por el puerto escogido para ello. La ruta de los archivos necesarios para el establecimiento de la seguridad es especificada en el XML de configuración expuesto en el diagrama de componentes (servidorContext.xml y clienteContext.xml).

```
System.setProperty("javax.net.ssl.keyStore", confEnvio.getKeystore());
System.setProperty("javax.net.ssl.keyStorePassword",
confEnvio.getKeypassword());
System.setProperty("javax.net.ssl.trustStore",
confEnvio.getTrustore());
```

3.6 VALIDACIÓN DEL RESULTADO

La validación del resultado se enfoca esencialmente a la comprobación de las características establecidas inicialmente en la investigación para lograr una transferencia de datos: seguridad, control e integración. Esta última condición es la unión del buen funcionamiento de las anteriores al plantear que existe integración cuando se logra el envío seguro de información entre dos entidades diferentes.

3.6.1 Trazas

El control de la información se garantiza a través del uso de trazas. Las trazas generadas durante la compilación y ejecución del componente muestran las acciones que se desarrollan en este proceso y son útiles en el diagnóstico de errores en el sistema y en el control de la información. Para ello se utilizó como sistema de gestión de logs/trazas Jakarta-Log4j, que es una de las más antiguas, potentes y conocidas []. Entre sus características principales se encuentran:

- Diferentes niveles de traza. (Error, información, depurar.)
- Filtros según categoría
- Redirección de las trazas a diferentes destinos. (A un archivo, a consola, a BBDD.)
- Diferentes formatos de visualización. (Visualizar fecha, línea, nombre de la clase)
- Configuración por ficheros

Durante la implementación del componente se utilizaron los niveles de INFO, DEBUG, ERROR y FATAL. INFO para el seguimiento del flujo normal en la compilación, DEBUG para mostrar los valores cargados de configuración y los datos que se envían. ERROR cuando se trata de errores que se producen que no limitan totalmente el funcionamiento del componente y FATAL en el caso de que el error que se produzca afecte a un proceso vital para el funcionamiento del componente.
En el diagrama de componentes se localiza el archivo Trazas.log que almacena la información generada por el sistema Log4j.

3.6.2 Pruebas de sistema.

Estas pruebas se limitan a configuraciones específicas que se hicieron dentro de la solución. En el caso de las *pruebas de seguridad*, se diseñan teniendo en cuenta que se utilizó SSL, que hace uso del sistema de codificación con dos claves: una pública y una privada. Se muestran a continuación las pruebas diseñadas en función de comprobar la seguridad de la transferencia.

Caso de Prueba 1	
Entrada:	Cambio de la llave privada del cliente.
Resultado esperado:	El sistema verifica la llave privada del cliente y no permite la conexión.
Resultado de la prueba:	No se trasmiten los datos, ya que el sistema no permite la conexión.

Condiciones:	Tener los permisos necesarios para cambiar el certificado digital.

Caso de Prueba 2	
Entrada:	Cambio de la llave pública del cliente en el servidor.
Resultado esperado:	El sistema verifica la llave pública del cliente en el servidor y no permite la conexión.
Resultado de la prueba:	No se trasmiten los datos, ya que el sistema no permite la conexión.
Condiciones:	Tener los permisos necesarios para cambiar el certificado digital.

De esta manera quedó probada la parte de la confidencialidad de los datos, ya que individuos no autorizados no tuvieron acceso a los mismos.

La integridad de los datos se verificó mediante el uso del programa sniffer Ethereal[12]. Fue utilizado para "captar" las tramas de datos que son transmitidos entre la aplicación de escritorio y la aplicación de integración. (Ver anexo 4)

Caso de Prueba 3	
Entrada:	Uso de Ethereal
Resultado esperado:	La trama que se observa debe estar encriptada.
Resultado de la prueba:	Mediante el uso del mismo no se pudo captar ni interceptar los datos enviados. La trama enviada está encriptada.
Condiciones:	Tener permisos de administración en la PC.

De esta manera, quedó probada la protección contra la alteración o corrupción de los datos.

La disponibilidad quedó probada, ya que tienen permiso para acceder al servidor todos aquellos clientes que necesitan mandar una determinada información.

Resultado general de las pruebas de seguridad: Para cada tipo de usuario conocido, las funciones y datos apropiados y todas las transacciones funcionan como se esperaba.

[12] Programa para monitorizar y analizar el tráfico en una red de computadoras

Para la *prueba de volumen* se realizó una sobrecarga en el envío de la información, con el objetivo de sobrepasar el flujo normal de información que se envía a los analistas del CNBA.

Caso de Prueba 4	
Entrada:	Envío de 60 MB
Resultado esperado:	El componente permite el envío.
Resultado de la prueba:	Se efectuó el envió total del flujo de datos sin presentar problemas y en un tiempo rápido.
Condiciones:	El usuario debe estar autorizado.

A pesar de que los ficheros que se necesitan enviar desde los entes a los analistas del CNBA son de escaso tamaño, se realizó esta prueba con el objetivo de soportar la posible reutilización del componente en problemas que requieran mayor volumen en la transferencia.
Resultado: El componente garantizó el envío sin problemas y quedó probada la capacidad de enviar los volúmenes de información requeridos.

Para la realización de las *pruebas de configuración*, se realizaron diferentes casos de prueba a partir del archivo XML, que contiene los atributos para la configuración ya sea del servidor o del cliente.

Caso de Prueba 5	
Entrada:	Cambio del puerto (5801).
Resultado esperado:	No se realiza la configuración.
Resultado de la prueba:	[FATAL] 16:44:36 ExecuteClient - No ha sido posible la configuración del cliente para realizar el envío
Condiciones:	Tener los permisos necesarios para cambiar el fichero de configuración.

Caso de Prueba 6	
Entrada:	Cambio de la dirección del host (10.7.7.12).
Resultado	No se realiza la configuración.

esperado:	
Resultado de la prueba:	[ERROR] 16:45:27 ExecuteClient - Exception creating connection to: 10.7.7.11; nested exception is: java.net.NoRouteToHostException: No route to host [FATAL] 16:45:27 ExecuteClient - No ha sido posible la configuración del cliente para realizar el envío.
Condiciones:	Tener los permisos necesarios para cambiar el fichero de configuración.

Caso de Prueba 7	
Entrada:	Cambio de los permisos (policy).
Resultado esperado:	No se realiza la configuración.
Resultado de la prueba:	No es posible acceder al servicio, provoca una excepción controlada del tipo “No Security Manager defined”.
Condiciones:	Tener los permisos necesarios para cambiar el fichero de configuración.

Caso de Prueba 8	
Entrada:	Cambio de la ruta del trustore.
Resultado esperado:	No se realiza la configuración.
Resultado de la prueba:	Ocurre una excepción del tipo IOException y no es posible la configuración del cliente.
Condiciones:	Tener los permisos necesarios para cambiar el fichero de configuración.

Caso de Prueba 9	
Entrada:	Cambio de la ruta del keystore.
Resultado esperado:	No se realiza la configuración.
Resultado de la prueba:	Ocurre una excepción del tipo IOException y no es posible la configuración del cliente.
Condiciones:	Tener los permisos necesarios para cambiar el fichero de configuración.

Caso de Prueba 10	
Entrada:	Password incorrecto
Resultado esperado:	No es posible desencriptar el keystore y el cliente no se conecta.
Resultado de la prueba:	Ocurre una excepción relacionada con la seguridad y el cliente no puede realizar el envío.
Condiciones:	Tener los permisos necesarios para cambiar el fichero de configuración.

Caso de Prueba 11	
Entrada:	Mecanismo de comunicación no definido para el componente
Resultado esperado:	No se realiza la configuración.
Resultado de la prueba:	Exception in thread "main" java.lang.ExceptionInInitializerError at albet.comunicacion.cliente.ClienteFactory.getCliente(ClienteFactory.java:36).
Condiciones:	Tener los permisos necesarios para cambiar el fichero de configuración.

El caso de prueba 11, arroja una excepción no controlada del tipo java.lang.ExceptionInInitializerError que fue controlada satisfactoriamente.

CONCLUSIONES

En este capítulo fueron descritos algunos fragmentos de la implementación del código del componente de software de esta investigación, así como su diagrama de componentes y las pruebas realizadas al mismo. Se destaca además la posibilidad de reutilización de este componente. La aplicación de las pruebas de sistema, que son importantes en la validación del resultado de una forma práctica.

CONCLUSIONES GENERALES

- La correcta especificación de los requisitos del componente de software permitió realizar el diseño.
- La utilización de los diferentes patrones de diseño permitió diseñar una solución considerando las buenas prácticas de desarrollo de software.
- Con el desarrollo se logró garantizar la transferencia de datos de manera segura entre la aplicación de escritorio de los entes y la aplicación de integración de la Solución Informática para el CNBA.
- El manual de desarrollo del componente de software implementado facilitará la comprensión de los interesados en hacer uso del componente.

RECOMENDACIONES

1. Implementar otras versiones del componente en otros lenguajes de programación que utilicen a CORBA y a los Web Services como mecanismo de comunicación.
2. Reutilizar el componente en sistemas similares desarrollados en la plataforma java.

REFERENCIAS BIBLIOGRÁFICAS

1. ¿Qué es la Seguridad Alimentaria? [En línea] Marzo 2004 [Citado el: 15 de noviembre de 2007] Disponible en: http://www.tecnociencia.es/especiales/seguridad_alimentaria/1.htm#
2. Figueroa Pedraza, Dixis. Seguridad Alimentaria Familiar. *Revista Salud Pública y Nutrición*. [En línea] Vol. 4, No.2 (2003). [Consulta el: 15 de noviembre de 2007]. Disponible en: http://www.respyn.uanl.mx/iv/2/ensayos/seguridad_alimentaria.htm
3. Extracto del Consejo de Ministros del 9/01/2008. Reflexiones del Presidente
4. Clemens, Szyperski. Component Software Component Software Beyond Object – Oriented Programming". P 20 -22. 1998.
5. Mike Bray, Lockheed-Martin [En línea] Enero 2007 [Citado el: 5 de Febrero de 2008.] Disponible en: http://www.sei.cmu.edu/str/descriptions/middleware.html
6. Protocolos de comunicación en Internet [En línea] mayo 2008 [Citado el: 13/02/08] Disponible en : http://es.wikipedia.org/wiki/Familia_de_protocolos_de_Internet
7. Introducción al lenguaje Java [En línea] 2007 [Citado el: 24 de Enero de 2008.] Disponible en: http://www.wikilearning.com/curso_gratis/introduccion_al_lenguaje_java-introduccion_a_java/5054-2
8. JAVA: El impacto del lenguaje en la telefonía celular. [En línea] 8 de Abril de 2008[Citado el: 15 de Mayo de 2008] Disponible en: http://www.taringa.net/posts/info/1146153/JAVA:-El-impacto-del-lenguaje-en-la-telefonia-celular.html
9. Sánchez, Jordi. ¿Qué es un framework? [En línea] Septiembre de 2006. [Citado el: 9 de Diciembre de 2007.] Disponible en: http://jordisan.net/modules/wordpress/2006/que-es-un-framework/
10. Spring Framework [En línea] Noviembre de 2007 [Citado el: 9 de Diciembre de 2007.] Disponible en: http://www.linuxparatodos.net/portal/staticpages/index.php?page=spring-framework
11. Servicios Hessian y Burlap [En línea] [Citado el: 10 de Enero de 2008.] Disponible en: http://www.caucho.com/resin-3.0/protocols/index.xtp

12. Servicios HTTP Invoker [En línea] [Citado el 10 de Enero de 2008.] Disponible en: http://di002.edv.uniovi.es/~cueva/asignaturas/doctorado/2006/trabajos/SW_ligeros.ppt

13. Hurtado Jara, Omar. Sistemas Distribuidos [En línea] [Citado el: 15 de Noviembre de 2007.] Disponible en: http://www.monografias.com/trabajos16/sistemas-distribuidos/sistemas-distribuidos.shtml

14. Catalog of OMG CORBA/IIOP Specifications. [En línea] Enero 2008 [Citado el: 5 de Febrero de 2008.] Disponible en: http://www.omg.org/technology/documents/corba_spec_catalog.htm

15. Interface Definition Language (IDL) [En línea] [Citado el: 5 de Febrero de 2008.] Disponible en : http://es.wikipedia.org/wiki/IDL

16. Object Request Broker (ORB) [En línea] [Citado el: 5 de Febrero de 2008.] Disponible en : http://sisbib.unmsm.edu.pe/bibvirtualdata/publicaciones/risi/N1_2004/a06.pdf

17. Remote Method Invocation (RMI) [En línea] [Citado el: 5 de Febrero de 2008.] Disponible en: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

18. Web Services Activity Statement [En línea] [Citado el: 10 de Enero de 2008.] Disponible en: http://www.w3.org/2002/ws/Activity

19. JENDROCK, Eric, *et al.* The Java EE 5 Tutorial for Sun Java System Application Server 9.1 [En línea] Septiembre de 2007 [Citado el: 1 de marzo de 2008]. Disponible en: http://java.sun.com/javaee/5/docs/tutorial/doc/bnayl.html

20. Mecanismos de Seguridad [En línea] [Citado el: 2 de Marzo de 2008.] Disponible en: http://es.tldp.org/Tutoriales/doc-tutorial-recomendaciones-seguridad/doc-tutorial-recomendaciones-seguridad.sgml.gz

21. VPN Consortium. VPN Technologies: Definitions and Requirements [En línea] March de 2006. [Citado el: 12 de Diciembre de 2007.] Disponible en: http://www.vpnc.org/vpn-technologies.html

22. OpenVPN. [En línea] [Citado el: 12 de Diciembre de 2007.] Disponible en: http://openvpn.net/

23. Onyszko, Tomasz. Authentication, Access Control & Encryption [En línea] Julio de 2004. [Citado el: 15 de Diciembre de 2007.] Disponible en: http://www.windowsecurity.com/articles/Secure_Socket_Layer.html

24. Metodologías de Desarrollo Software [En línea] [Citado el: 20 de Febrero de 2008] Disponible en: http://www.scribd.com/doc/2050925/metodologias-de-desarrollo-software

25. Programación Extrema [En línea] Junio del 2004.[Citado el: 15 de diciembre de 2007.] Disponible en: www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html

26. Visual Paradigm.[En línea][Citado el: 10 de Marzo de 2008] Disponible en: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%5Bcuenta_de_Plataforma_de_Java_14715_p/

27. Bienvenido a NetBeans y a www.nwetbeans.org [En línea] [Citado el: 24 de Enero de 2008.] Disponible en: http://www.netbeans.org/index_es.html

28. Introducción a la plataforma Eclipse [En línea] [Citado el: 24 de Enero de 2008.] Disponible en: http://www.fing.edu.uy/inco/grupos/coal/investigacion/proyectos/lead/docs/IntroduccionEclipse.pdf

29. The Apache Software Fundation [En línea] [Citado el: 10 de Abril de 2008.] Disponible en: http://tomcat.apache.org/

30. Subversion [En línea] [Citado el: 10 de Abril de 2008.] Disponible en: http://subversion.tigris.org/

31. Lago Torres, Manuel. Introducción al diseño con patrones [En línea] Diciembre de 2002. [Citado el: 15 de Diciembre de 2007.] Disponible en: http://www.elrincondelprogramador.com/default.asp?id=29&pag=articulos/leer.asp

32. Pruebas [En línea] [Citado el 28 de Abril de 2008.] Disponible en: http://rogeliodavila.com/tcs/TCS%20Notes%20JAVega/Parte_15_TestBlack.ppt

BIBLIOGRAFÍA

1. Información general acerca de servicios Web[En línea] [Citado el: 1 de Noviembre de 2007] Disponible en: http://msdn.microsoft.com/library/spa/default.asp?url=/library/SPA/cpguide/html/cpconwebservicesoverview.asp
2. Java Remote Method Invocation - Distributed Computing for Java. [En línea] [Citado el: 6 de Noviembre de 2007] Disponible en: http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp
3. Ramiro Lago. Introducción a Remote Method Invocation. [En línea] Febrero de 2006 [Citado el: 6 de Noviembre de 2007] Disponible en: http://www.proactiva-calidad.com/java/rmi/introduccion.html
4. Smitha Krishna Nagesh , Marina Sum. Designing Patterns With UML. [En línea] 12 de Abril de 2006 [Citado el: 8 de Noviembre de 2007] Disponible en: http://developers.sun.com/jsenterprise/nb_enterprise_pack/reference/techart/uml_patterns.html
5. Ayub Khan, Marina Sum. Introducing Design Patterns in XML Schemas. [En línea] 9 de Noviembre de 2006 [Citado el: 8 de Noviembre de 2007] Disponible en: http://developers.sun.com/jsenterprise/nb_enterprise_pack/reference/techart/design_patterns.html
6. Catálogo de Patrones de Diseño J2EE. I.- Capa de Presentación. [En línea] [Citado el: 8 de Noviembre de 2007] Disponible en: http://www.programacion.net/java/tutorial/patrones/1/
7. Sun Java Center J2EE Patterns. [En línea] [Citado el: 12 de Noviembre de 2007] Disponible en: http://java.sun.com/j2ee/patterns/DataAccessObject.html
8. Core J2EE Patterns - Business Delegate. [En línea] [Citado el: 12 de Noviembre de 2007] Disponible en: http://java.sun.com/blueprints/corej2eepatterns/Patterns/BusinessDelegate.html
9. Hernández Carabaño, Héctor. Seguridad Alimentaria y Calidad de Vida en Venezuela. *Gaceta Médica de Caracas*. [En línea] vol.112, no.3 (2004), [Citado el: 15 de noviembre de 2007]. Disponible en: http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S0367-47622004000300014&lng=es&nrm=iso
10. Utilice las redes privadas virtuales para la transferencia segura de datos en Internet[En línea] 16 de Septiembre de 2003 [Citado el: 18 de Noviembre de 2007] Disponible en:

http://www.microsoft.com/spain/windowsxp/using/mobility/expert/vpns.mspx

11. Ariza Rojas, Maribel y Molina García, Juan Carlos. Introducción y principios básicos del desarrollo de software basado en componentes. Septiembre de 2004. [Citado el: 6 de diciembre de 2007].
12. Comunicación entre aplicaciones distribuidas [En línea] [Citado el: 8 de Enero de 2008] Disponible en: http://msdn.microsoft.com/library/spa/default.asp?url=/library/SPA/vsent7/html/vxoriChoosingBusinessLogicTechnologies.asp
13. Pressman, Roger S. Ingeniería del Software: un enfoque práctico. Parte I y II / Madrid, McGraw-Hill, 2002, ed. 5ta. (MON-002581) 601p. [Citado el: 10 de Enero de 2008.] Disponible en: http://bibliodoc.uci.cu/pdf/reg02689.pdf
14. Cristo Rodríguez,Iván F. Cuéllar. CORBA: Common Object Request Broker Architectura. [En línea] [Citado el: 10 de Enero de 2008] Disponible en: http://agamenon.uniandes.edu.co/~revista/articulos/corba/corba.htm
15. Jacobson, Ivar, Booch, Grady y Rumbaugh, James. El Proceso Unificado de Desarrollo de Software. s.l.: PEARSON EDUCACIÓN S.A, 2000.[Citado el: 12 de Enero de 2008]
16. Zukowski, John. Programación en Java J2SE 1.4. Noviembre de 2007. Ciudad Habana, Editorial.: Felix Varela, 2007. [Citado el: 15 de Febrero de 2008.]
17. Eclise documentation - Archived Release [En línea][Citado el: 2 de Marzo de 2008] Disponible en: http://help.eclipse.org/help31/index.jsp?topic=/org.eclipse.jst.ws.doc.user/concepts/cjaxrpc.html
18. Aplicaciones distribuidas en Internet/Intranets: de los sockets a los Objetos Distribuidos. [En línea] 25 de Septiembre de 2006 [Citado el: 12 de Marzo de 2008] Disponible en: http://ditec.um.es/~dsevilla/
19. Larman, C. UML y Patrones: Introducción al análisis y programación orientada a objetos. México, Prentice Hall, 1999, 536 p.; (MON-001311) [Citado el: 15 de Abril de 2008.] Disponible en: http://bibliodoc.uci.cu/pdf/reg00062.pdf
20. Rodríguez Lasterra, Enrique. Sistemas de Trazas [En línea] Enero de 2003. [Citado el: 9 de mayo de 2008] Disponible en: http://www.javahispano.org/contenidos/es/sistemas_de_trazas/
21. Servicio Web. [En línea] 22 de Octubre de 2007 [Citado el: 14 de Mayo de 2008] Disponible en: http://es.wikipedia.org/wiki/Servicio_Web

22. Introducción a los Servicios Web en Java. [En línea] [Citado el: 14 de Mayo de 2008] Disponible en: http://www.programacion.com/tutorial/servic_web/

23. Invocación Remota de Métodos (RMI). [En línea] [Citado el: 14 de Mayo de 2008] Disponible en: http://www.programacion.com/tutorial/rmi/2/

24. Ayuda extendida del Rational Rose Enterprise Edition 2003. [Citado el: 18 de Mayo de 2008]

25. Sun Developer Network(SDN) GlassFish Community [En línea] [Citado el: 18 de Mayo de 2008] Disponible en: http://java.sun.com/javaee/community/glassfish/

ANEXOS

Anexo 1: Control de versiones

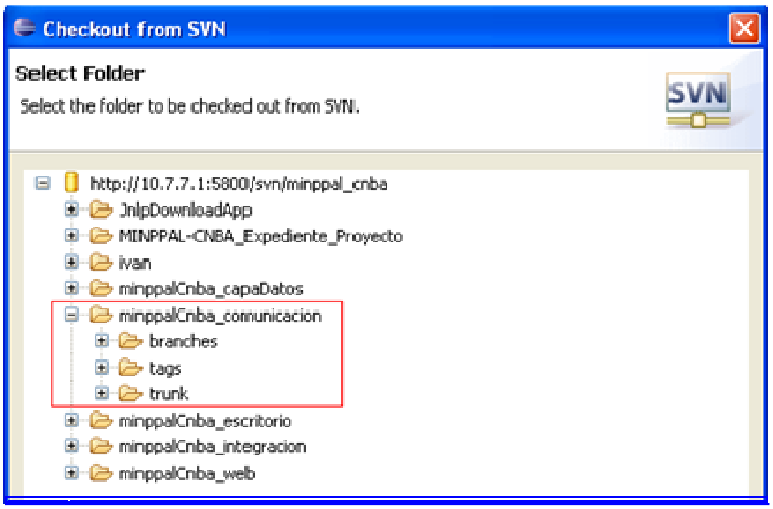
El control de versiones se realizó a través del uso del SubEclipse (cliente de Subversion que se integra como un plugin al Eclipse), propiciando una adecuada gestión de la implementación entre los miembros del equipo. Se utilizó la estructura de carpetas ofrecida por esta herramienta para una mayor organización.

Trunk: para el desarrollo del tronco principal de la implementación.

Tags: para la ubicación de las versiones estables

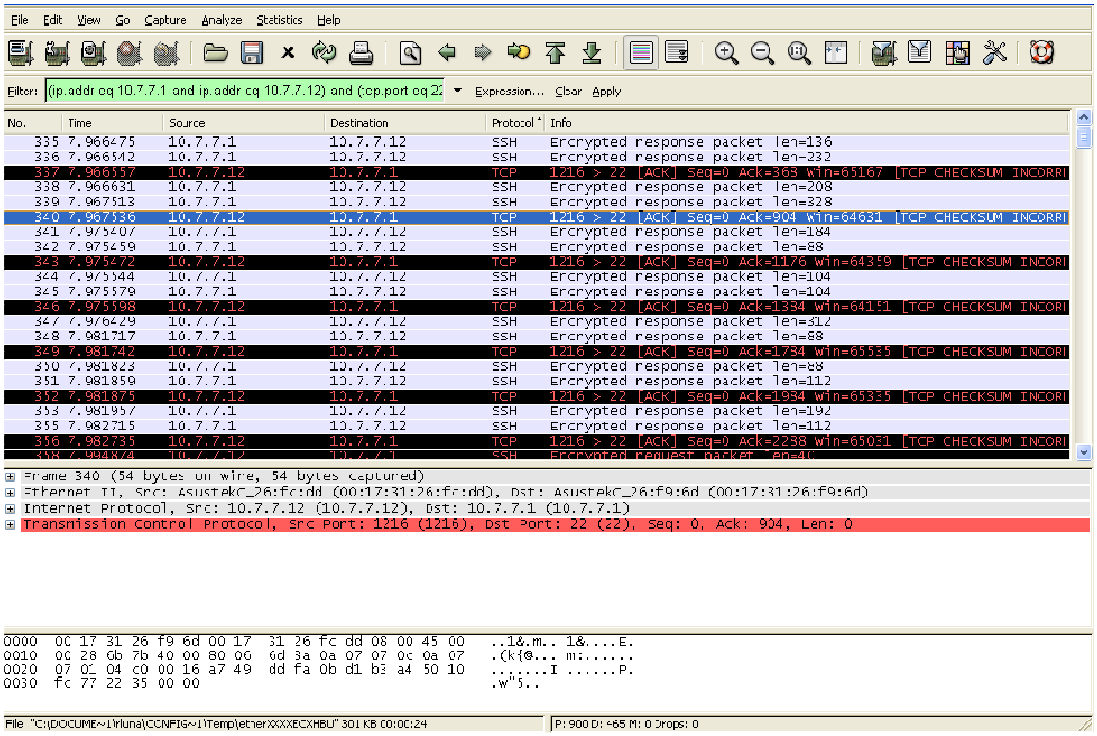
Branches: para la corrección de los problemas detectados durante las pruebas a las versiones estables anteriormente ubicadas en el Tags.

La estructura de las carpetas se señala en rojo en la imagen que se muestra a continuación.



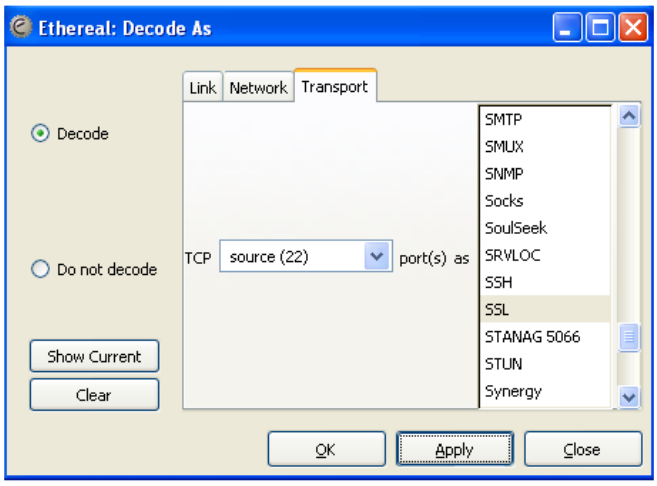
Estructura de las carpetas en el Subversion

Anexo 2: Imágenes capturadas de las pruebas utilizando Ethereal

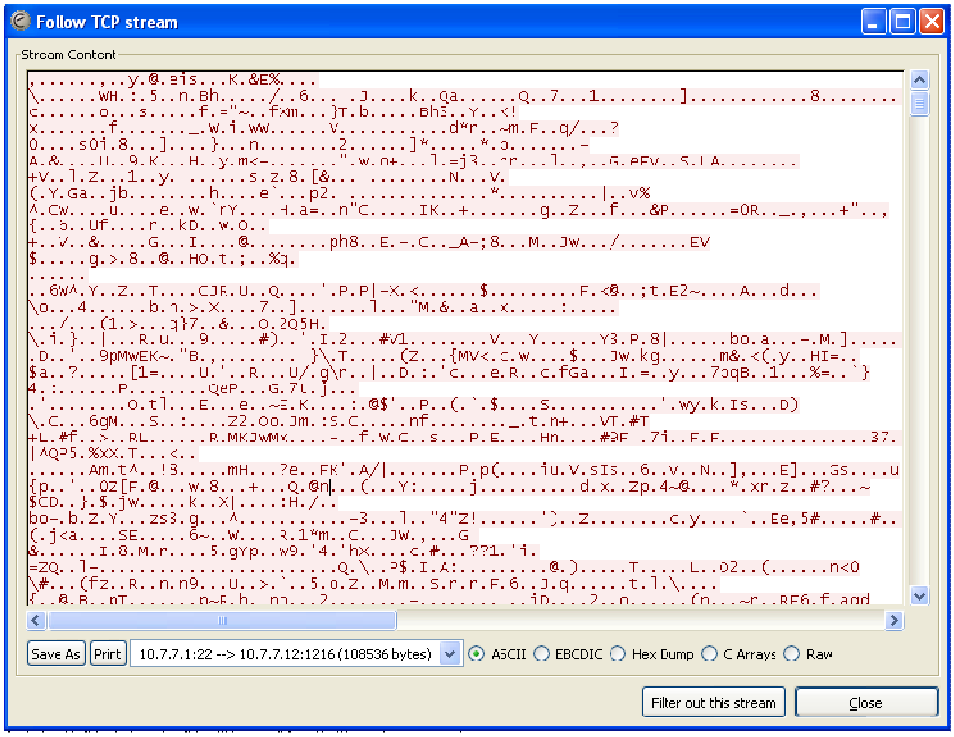


La imagen anterior muestra la pantalla principal de Ethereal en la se pueden observar las tramas de red entrantes y salientes de la PC.

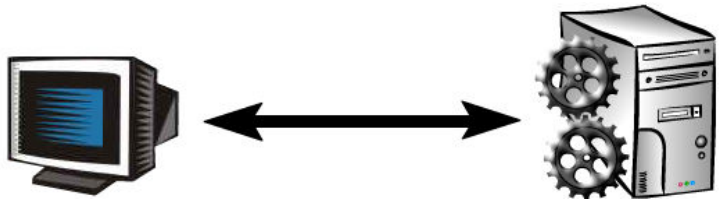
Se trata de decodificar la trama que viaja por SSL.



La trama obtenida está encriptada y no es posible decodificarla, como se muestra a continuación.


Follow TCP stream
Stream Content
Save As
Print
10.7.7.1:22 --> 10.7.7.12:1216 (108536 bytes)
ASCII
EBCDIC
Hex Dump
C Arrays
Raw
Filter out this stream
Close

Anexo 3: Manual del desarrollador



Componente de Software para la transferencia de datos en arquitecturas Cliente/Servidor
Manual del desarrollador
Versión 1.0

Control de versiones

Fecha	Versión	Descripción	Autor
17/05/08	1.0	Redacción del documento	Reynaldo Alvarez Luna

Tabla de contenido

Introducción

Un componente de software es una unidad de composición con interfaces contractualmente especificadas y explícitas sólo con dependencias dentro de un contexto. Un componente de software puede ser desplegado independientemente y es sujeto a la composición de terceros.
En el manual se expondrán los elementos necesarios para los desarrolladores interesados en utilizar el componente en problemas que requieran transferencias de datos en arquitecturas Cliente/Servidor en sistemas más complejos. La utilización del mismo permite de manera simple dicha transferencia sin tener que conocer las particularidades de las tecnologías de comunicación utilizadas en el envío.

Tecnologías utilizadas en el desarrollo

Plataforma Java

Como elemento fundamental vale destacar que el componente fue desarrollado en la plataforma Java y exige el uso de Java tanto en el cliente como el servidor. Se exponen a continuación, de manera resumida algunas características de Java.
La tecnología Java, una tecnología madura, extremadamente eficaz y sorprendentemente versátil, se ha convertido en un recurso inestimable ya que permite a los desarrolladores:

- Desarrollar software en un Sistema Operativo y ejecutarlo en prácticamente cualquier otro Sistema Operativo.

- Crear programas para que funcionen en un navegador web y en servicios web.

- Desarrollar aplicaciones para servidores como foros en línea, tiendas, encuestas, procesamiento de formularios HTML, etc.

- Combinar aplicaciones o servicios basados en la tecnología Java para crear servicios o aplicaciones totalmente personalizados.

- Desarrollar potentes y eficientes aplicaciones para teléfonos móviles, procesadores remotos, productos de consumo de bajo coste y prácticamente cualquier dispositivo digital.

- Crear aplicaciones en entornos distribuidos.

Tecnologías remotas

Sistema de Invocación Remota de Métodos (RMI)

El sistema de Invocación Remota de Métodos (RMI) de Java permite, a un objeto que se está ejecutando en una Máquina Virtual Java (VM), llamar a métodos de otro objeto que está en otra VM diferente. Esta tecnología está asociada al lenguaje de programación Java, es decir, que permite la comunicación entre objetos creados en este lenguaje.

Las aplicaciones RMI normalmente comprenden dos programas separados: un servidor y un cliente. Una aplicación servidor típica crea cierta cantidad de objetos remotos, hace accesibles unas referencias a dichos objetos remotos, y espera a que los clientes llamen a estos objetos remotos. Una aplicación cliente típica obtiene una referencia remota de uno o más objetos remotos en el servidor y llama a sus métodos. RMI proporciona el mecanismo por el que se comunican y se pasan información del cliente al servidor y viceversa.

Servicios HTTP Invoker

HTTP Invoker es uno de los servicios remotos que son soportados por Spring Framework. El mismo emplea el modelo remoto para hacer llamadas remotas sobre HTTP y al mismo tiempo el paso de objetos de Java usando técnicas de serialización del lenguaje. Esto hace que la implementación de un servicio remoto desde una clase de Java es más fácil y permite al desarrollador concentrarse en la interfaz de negocio del servicio remoto por encima de los detalles de la infraestructura del mismo. Se basa en la infraestructura de invocación de RMI, pero utiliza HTTP como protocolo de transporte. Superando a este en cuanto a la capacidad de atravesar firewalls.

Requerimientos

Software

- Se deberá disponer para el uso del componente, del Sistema Operativo Windows 98 o superior, o cualquier distribución de Linux.
- Para el servidor de la aplicación el sistema operativo recomendado es Windows Server 2003 o superior o Linux.
- Se debe instalar TOMCAT como servidor web.
- Debe estar instalado el Java Runtime Environment (JRE) versión 1.6 o superior.

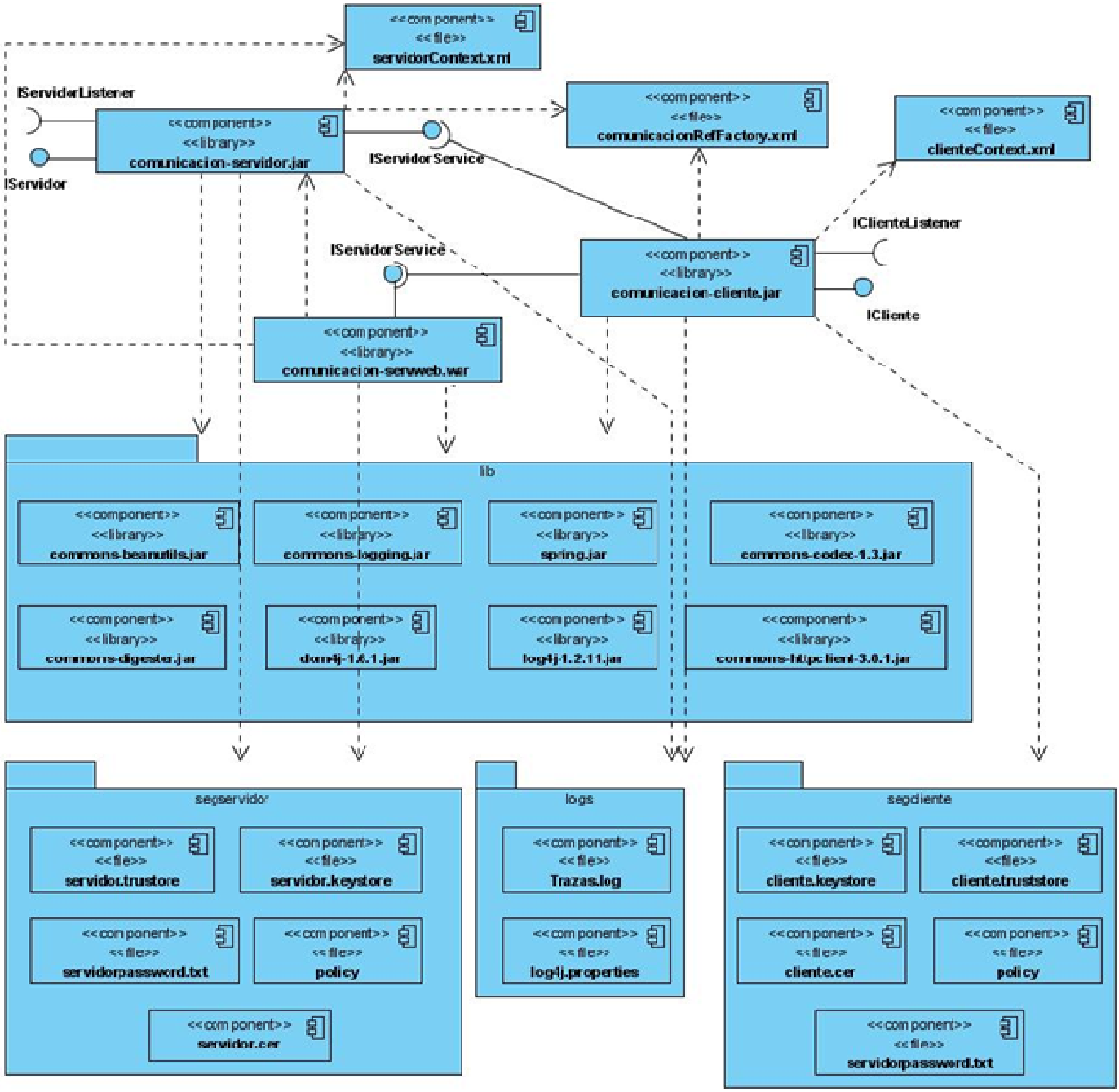
Hardware

Para el funcionamiento del componente se requiere de máquinas con los siguientes requisitos:

- Para las PC donde se ejecutará la aplicación cliente: Procesador Pentium 3 o superior, 256 Mb de RAM, 1 GB de capacidad del disco duro.
- El servidor debe tener las siguientes características: capacidad de disco duro superior a 80.0 GB, microprocesador Pentium IV superior a 2.0 GHz y como mínimo 1.0 GB de RAM.

Estructura del componente

En los componentes “comunicación-servidor.jar”, “comunicación-cliente.jar” y “comunicación-servweb.war” se ubican las clases (.java) de los paquetes ubicados en el servidor y cliente, como sus nombres lo indican. El resto de los componentes representados constituyen los archivos de configuración (clienteContext.xml, servidorContext.xml y comunicacionRefFactory.xml), las librerías utilizadas (agrupadas en el paquete lib), los archivos necesarios para establecer la seguridad con SSL (representados en los paquetes segcliente y segservidor) y en el paquete logs, los archivos necesarios para garantizar la trazabilidad en el envío (Trazas.log para almacenar las trazas y log.properties para establecer el nivel de trazabilidad). Se muestran las interfaces con las que interactúa el sistema que requiera la utilización del componente, `IServidorService` es la interfaz de servicio para la comunicación entre el componente cliente y servidor.



Se muestra a continuación el diagrama de componentes que muestra las dependencias internas del componente.

Integración a sistemas

Cliente

Configuración del cliente

La configuración del cliente para el envío de la información se realiza a través del archivo XML que se muestra a continuación; el mismo contiene los atributos necesarios para la conexión al servidor (Ver clase *ConfiguracionEnvio*), la dirección de los certificados digitales, la dirección del archivo que establece los permisos en java (policy) y el mecanismo de comunicación que se utilizara para el envío. Este archivo es para el uso de los desarrolladores que deseen utilizar el componente, de esta manera todos los detalles de configuración son removidos del código, lo que hace más accesible su utilización. Para un desarrollador cambiar la tecnología de comunicación para el envío, bastaría con cambiar la propiedad "mecComunicacion" y colocar HTTPINVOKER en lugar de RMI en este caso y se abstrae del conocimiento empleado en la implementación de las tecnologías.

```
<bean id="configuracion"
class="albet.comunicacion.cliente.ConfiguracionEnvio">
            <property name="puerto">
               <value>5801</value>
            </property>
           <property name="host">
               <value>10.7.7.12</value>
            </property>
           <property name="policy">
               <value>policy</value>
            </property>
           <property name="trustore">
               <value>segcliente/client.truststore</value>
            </property>
            <property name="keystore">
               <value> segcliente /client.keystore</value>
            </property>
            <property name="keypassword">
               <value> segcliente /password.txt</value>
```

```
        </property>
        <property name="mecComunicacion">
            <value>RMI</value>
        </property>
    </bean>
```

Integración del cliente al sistema

Para acceder al envío de la información es suficiente con cuatro pasos:

1. Crear una instancia del tipo *ICliente* a través de la clase *ClienteFactory* invocando su método estático getCliente(), que internamente adquiere los parámetros para la creación del objeto del XML de configuración.
2. Crear una instancia de la clase *IClienteListener* y pasarla como parámetro al método registrarListener(IClienteListener listener), dicha instancia estará en espera de los eventos provocados por la respuesta del servidor. Con el objetivo de permitir el envío simultáneo de varios objetos se crea una instancia del listener para cada petición al servidor.
3. Invocar el método enviarObjeto(Object o; Obejct id) para el envío de la información, es válido aclarar que es posible el envío de cualquier objeto que tenga como clase base java.lang.Object.
4. Implementar la interfaz *IClienteListener* (Fig.22). El método `setResponse(Object id, Object object)` es invocado cuando llega la respuesta de un envío y el programador de la aplicación cliente debe asumir su implementación.

Interfaz IClienteListener

Ej:

```
public void setResponse(Object id, Object object) {
     System.out.println("El envio con id " + id + "devolvió" + object);
}
```

A través de este método se imprime en consola la respuesta del servidor. Es válido aclarar que es necesario que ambas partes involucradas en la transferencia conozcan el tipo de objeto que envían en las peticiones y respuestas.

A continuación se muestra un fragmento de código como ejemplo a las acciones explicadas

anteriormente (los círculos amarillos se corresponden con los pasos expuestos).

```
static ICliente cliente;
static Log traza;
public ClienteListener()
{
}

public static void main(String[] args)
{
	traza = LogFactory.getLog(ExecuteClient.class);
	try
	{
		cliente  = ClienteFactory.getCliente();
		traza.debug("Creado objeto cliente para envio" +
cliente);
		cliente.registrarListener(new ExecuteClient());
		cliente.enviarObjeto("datos", 50);
	}
	catch (Exception e)
	{
		traza.error(e.getStackTrace());
		traza.fatal("No ha sido posible la configuración del
cliente para realizar el envío");
	}
}
```

1 2 3

Finalmente es necesario el tratamiento de las excepciones que puedan ser lanzadas en el proceso de envío, ya que el componente relanza al exterior todas las excepciones que no puedan ser controladas internamente y que no sean fatales en el funcionamiento del mismo, por ello es necesario que este tratamiento sea de acuerdo al sistema que utilice el componente.

Realizando el envío internamente

La respuesta del servidor puede ser más o menos lenta según las condiciones de la red sobre la que se encuentre el servicio desplegado, por ello previendo la demora en dichas respuestas se ha desarrollado la clase *Emisor* (que extiende de *Thread*) para manejar los envíos efectuados como hilos independientes y de esta manera permitir al usuario que pueda realizar otras acciones mientras culmina el proceso de envío de la información.

En el cuerpo del método enviarObjeto(Object obj, Object id) lo que se realiza fundamentalmente es el inicio del hilo que maneja el envío:

```
emisor = new Emisor(mensaje, id, clienteListener, remoteServ); //
inicialización del emisor
emisor.start(); // inicio del hilo de ejecución que internamente
invoca     //al método run()
```

El método run() de la clase Emisor invoca al método de servicio publicado en el servidor.

```
public void run()
{
     Object result = null;
     try
     {
          result = remoteServ.recibirObjeto(envio);
          traza.debug(result);
     }
     catch (Exception e)
     {
          traza.error(e.getMessage());
          clienteListener.setResponse(idEnvio, e);
     }
     if (result == null)
     {
          traza.info("La respuesta del servidor fue null");
     }
     else
     {
          traza.info("La respuesta del servidor se ha procesado
correctamente");
     }
     clienteListener.setResponse(idEnvio, result);
}
```

Servidor

Configuración del servidor

La configuración del servidor es similar a lo explicado anteriormente, se realiza a través de un XML similar, que contiene de igual manera la configuración necesaria para la publicación del servicio. De igual manera el desarrollador gana con la fácil exportación de un servicio ya sea por RMI o HTTPINVOKER.

Publicando el servicio

Para la publicación del servicio y la puesta en marcha del servidor es sencillo y no es necesario tener conocimientos acerca de la tecnología que va a utilizar el componente para publicar el servicio, solo es necesario los siguientes pasos para la integración del componente al sistema:

1. Crear una instancia del tipo *IServidor* a través de la clase *ServidorFactory* invocando su método estático getServidor(). La creación del objeto se realiza a través de los elementos de configuración establecidos en el XML de configuración (servidorContext.xml).
2. Crear una instancia de la clase *IClienteListener* y pasarla como parámetro al método registrarListener(IServidorListener listener), dicha instancia estará en espera de los eventos provocados por las peticiones de los clientes.
3. Invocar el método iniciar(), que arranca el servicio para el servidor específico creado como instancia de la clase base IServidor.
4. Implementar la interfaz *IServidorListener* (Fig. 23), el método `recibirObjeto(Object mensaje)` es invocado cuando llega la petición al servidor, el programador de la aplicación servidor debe asumir su implementación de acuerdo a lo requiera hacer con el objeto recibido y a su vez devuelve la respuesta del servidor una vez procesada la petición.

Interfaz IServidorListener

Ej:

```
public  Object recibirObjeto(Object mensaje)
{
     return "La respuesta del Servidor después de procesar el
            objeto es correcta";;
}
```

A continuación se muestra un fragmento de código como ejemplo a las acciones explicadas

anteriormente (los círculos amarillos se corresponden con los pasos expuestos).

```
public static void main(String[] args)
{
	try
	{
		servidor = ServidorFactory.getServidor();
		servidor.registrarListener(new ExecuteServidor());
		servidor.iniciar();
	}
	catch (Exception e)
	{
		System.out.println("Servidor mal construido");
		e.printStackTrace();
	}
}
```

Es fundamental el tratamiento de las excepciones que se pueden originar en la invocación de los métodos del componente, para el desarrollador que lo utilice constituye una caja negra por lo que es necesario el tratamiento de las excepciones que puedan ser lanzadas al exterior durante la ejecución.

Publicación y acceso al servicio de las tecnologías remotas disponibles en esta versión.

Servicio a través de RMI

El otro procedimiento para el acceso al servicio se ha implementado a través de RMI, que es un protocolo fuerte y reconocido porque soporta una completa serialización de objetos, que es importante cuando usamos modelos de datos complejos que necesitan ser serializados sobre red. Es más recomendado en caso de que donde se vaya a desplegar no haya problemas con el paso de los firewall.

Aunque Spring brinda soporte para la publicación de servicios RMI tradicionales, no presenta una madurez aceptable en cuanto a la seguridad, por ello en la investigación se ha implementado RMI de la manera tradicional de Java.

Para el uso de RMI es necesario destacar dos cuestiones esenciales:

1. La interfaz de servicio debe extender de la interfaz *java.rmi.Remote.*

2. La clase servidora debe extender de la clase *UnicastRemoteObject* e implementar la interfaz de servicio, en este caso *IServidorService*.

El resto no es complejo, pues solo es necesaria la creación del registro y vincular el objeto que ejecutará el servicio remoto en el servidor.

```
Registry registro = LocateRegistry.createRegistry(puerto);
obj = new ServidorRMI();
registro.bind("Objeto", obj);
```

Para acceder en el cliente es necesario conocer el IP del servidor, y estar autorizado al mismo, con sus certificados digitales.

En la clase cliente es necesario localizar el registro y crear un objeto del tipo *IServidorService* para acceder al método del servidor como si fuera local.

```
remoteServ = (IServidorService) registry.lookup("Objeto");
remoteServ.recibirObjeto(objeto);
```

Servicio a través de HTTP Invoker

La estrategia remota de Spring, HTTP invoker es una buena opción si se necesita el transporte de los datos sobre HTTP o HTTPS, además acude a la serialización de los objetos de java. Comparte la infraestructura básica de RMI, pero utilizando HTTP como transporte, lo que elimina el problema de RMI con los firewall.

El servicio HTTP Invoker requiere ser desplegado en un servidor de aplicaciones web para java, en este caso en el Tomcat, el componente “comunicacion-servweb.war” representa el compilado de la aplicación web que debe desplegarse. El despliegue del mismo se puede realizar de forma manual o mediante el archivo de configuración (en este caso indicar “true” en la propiedad despliegue del archivo). En el mismo se debe escribir la dirección del TOMCAT_HOME y la ruta de la ubicación de la aplicación servicio a desplegar.

```
          <bean id="tomcat"
class="albet.comunicacion.servidor.TomcatManager">

             <property name="despliegue">
               <value>false</value>
             </property>

                <property name="tomcatUrl">
               <value>/opt/apache-tomcat-6.0.14/</value>
             </property>
```

```
            <property name="warUrl">
          <value>/home/administrador/Escritorio/comunicacion-
          servweb.war</value>
            </property>
        </bean>
```

La clase HttpInvokerServiceExporter de Spring es la encargada de exportar el servicio del lado del servidor, es necesario indicarle la propiedad service que hace referencia al bean remoteservice, para asignarle la clase que sostendrá el servicio, en este caso *ServidorServiceImpl* es la encargada de la soportar la lógica del servicio a través de la implementación de la interfaz *IServidorService* la propiedad serviceInterface (para indicarle la interfaz del servicio que brinda). Veamos:

```
<bean name="/xmlservice"
class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExpo
rter">
          <property name="service" ref="remoteService" />
          <property name="serviceInterface"
     value="albet.comunicacion.comun.IServidorService" />
</bean>
<bean id="remoteService"
class="albet.comunicacion.servidor.ServidorServiceImpl" />
```

Del lado cliente Spring brinda la clase HttpInvokerProxyFactoryBean para el acceso al servicio, de similar manera es necesario indicarle las propiedades serviceUrl (URL del servicio al que se desea acceder), serviceInterface (la interfaz del servicio a la que accede) y httpInvokerRequestExecutor (para lograr el acceso a servicio a través de HTTPS, la cual es solo necesaria en el caso de que se requiera el acceso seguro al servicio, como en este caso).

```
<bean id="httpWebInvoker"
class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactor
yBean">
          <property name="serviceUrl">
          <value>
https://10.7.7.12:8443/Comunicacion_Servidor/remoting/xmlservice
```

```
            </value>
            </property>
            <property name="serviceInterface">
            <value>albet.comunicacion.comun.IServidorService</value>
            </property>
            <property name="httpInvokerRequestExecutor">
            <bean
class="org.springframework.remoting.httpinvoker.CommonsHttpInvokerRequ
estExecutor"/>
            </property>
</bean>
```

El bean remoteHttpInvoker representa a la clase *ClienteHttpInvoker* que es la clase mediante la cual se accede al servicio, para la que se ha definido una propiedad remoteServ (del tipo de la interfaz *IServidorService*, que en este caso toma el valor del bean httpWebInvoker de esta forma se "apropia" de la información necesaria para acceder a los objetos remotos publicados en el servidor, explicado anteriormente) y otra propiedad confEnvio a través de la cual se hace referencia al bean configuracion, que contiene los elementos necesarios para acceder al servidor.

```
<bean id="remoteHttpInvoker"
class="albet.comunicacion.cliente.impl.ClienteWebservice">
            <property name="remoteServ">
            <ref bean="httpWebInvoker" />
            </property>
            <property name="confEnvio">
            <ref bean="configuracion" />
            </property>
</bean>
```

El servicio establecido permite la transferencia de todo tipo de objetos, pero vale destacar que en el caso del servicio a través de HTTP Invoker es necesario agregar las clases de las cuales sea necesario enviar objetos, para que el mecanismo de serialización - deserialización de RMI funcione correctamente, por ejemplo: la aplicación de integración devuelve ante alguna petición un objeto de la Respuesta que al no ser una clase típica del JDK no es soportada de manera natural. La solución

entonces consiste en incluir la clase en el CLASSPATH de la aplicación.

Seguridad

La seguridad se ha garantizado a través del uso de certificados digitales y SSL, además de que en ambos casos los objetos viajan serializados.

Es necesario para el buen funcionamiento, incluir al CLASSPATH de java los elementos de seguridad para la ejecución tanto del cliente como del servidor (ver código citado). Además de transformar las policy establecidas en JAVA_HOME para que existan los permisos necesarios para el acceso y la exportación del servicio por el puerto escogido para ello. La ruta de los archivos necesarios para el establecimiento de la seguridad es especificada en el XML de configuración expuesto en el diagrama de componentes(servidorContext.xml y clienteContext.xml).

```
System.setProperty("javax.net.ssl.keyStore", confEnvio.getKeystore());
System.setProperty("javax.net.ssl.keyStorePassword",
confEnvio.getKeypassword());
System.setProperty("javax.net.ssl.trustStore",
confEnvio.getTrustore());
```

En el caso de RMI se realiza la creación de sockets a través de las clases *RMIClientSocketFactory* y *RMIServerSocketFactory*, que como su nombre lo indica fabrican los sockets en el cliente y servidor, para lograr la seguridad de la transferencia. Para ello es necesario indicarle a dichas clases la dirección de los certificados digitales, el puerto y el host. Y las mismas se garantizan que la información viaje cifrada a través de la red.

El cifrado se realiza a través del algoritmo SunX509, el servidor tiene el conjunto de clientes autorizados en un archivo denominado **trustore,** el conjunto de claves válidas en el **keystore,** y la encriptación se realiza con una contraseña solo conocida por los clientes autorizados.

En el caso de HTTP Invoker interviene el servidor de aplicaciones web, Apache Tomcat, al cual en su archivo de configuración (conf/server.xml) es necesario especificarle la ruta del keystore (keystoreFile) y la contraseña (keystorePass), elementos necesarios para exportar el servicio por HTTPS (SSL sobre HTTP) como se muestra en el siguiente fragmento:

```
<Connector protocol="org.apache.coyote.http11.Http11Protocol"
port="8443" minSpareThreads="5" maxSpareThreads="75"
enableLookups="true" disableUploadTimeout="true" acceptCount="100"
maxThreads="200" scheme="https" secure="true" SSLEnabled="true"
```

```
keystoreFile="/home/server_cer/servidor.keystore"
keystorePass="qazwsx" clientAuth="false" sslProtocol="TLS" />
```

Trazas

Las trazas generadas durante la compilación y ejecución del componente muestran las acciones que se desarrollan en este proceso y son útiles en el diagnóstico de errores en el sistema. Para ello se utilizó como sistema de gestión de logs/trazas Jakarta-Log4j, que es una de las más antiguas, potentes y conocidas. Entre sus características principales se encuentran:

- Diferentes niveles de traza. (Error, información, depurar.)
- Filtros según categoría
- Redirección de las trazas a diferentes destinos. (A un archivo, a consola, a BBDD.)
- Diferentes formatos de visualización. (Visualizar fecha, línea, nombre de la clase)
- Configuración por ficheros

Durante la codificación del componente se utilizaron los niveles de INFO, DEBUG, ERROR y FATAL. INFO para el seguimiento del flujo normal en la compilación, DEBUG para mostrar los valores cargados de configuración y los datos que se envían. ERROR cuando se trata de errores que se producen que no limitan totalmente el funcionamiento del componente y FATAL en el caso de que el error que se produzca es afecta a un proceso vital para el funcionamiento del componente.

En el diagrama de componentes se localiza el archivo Trazas.log que almacena la información generada por el sistema Log4j.

GLOSARIO

API: Application Program Interface

ARP: Address Resolution Protocol

BD: Base de Datos

CNBA: Centro Nacional de Balance Alimentario

CORBA: Arquitectura Común de Intermediarios en Peticiones a Objetos

CU: Caso de Uso

Firewall: elemento utilizado en redes de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas.

FTP: File Transfer Protocol

GPL: General Public License

Hibernate: Framework para persistencia de Java.

HTTP Invoker: servicio remoto soportado por Spring Framework.

HTTP: Hyper Text Transfer Protocol

HTTPS: Web encriptado bajo Secure Socket Layer (SSL)

IDE: Entorno Integrado de Desarrollo

IDL: Interface Definition Language (Lenguaje de Definición de Interfaces)

IP: Protocolo de Internet

J2EE: Java 2 Enterprise Edition. Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java.

JDT: Java Development Tool

LAN: Local Area Network

LDAP: Lightweight Directory Access Protocol

MINPPAL: Ministerio del Poder Popular para la Alimentación

OASIS: Organization for the Advancement of Structured Information Standards

ODBC: Open Database Connectivity (Conectividad Abierta de Base de Datos)

ORBs: Object Request Brokers

OSI: Open System Interconnection

POP: Post Office Protocol

POP3: Post Office Protocol 3

RMI: Invocación Remota de Métodos (Remote Invocation Method)

RPC: Llamada a Procedimiento Remoto (Remote Procedure Call)

RSA: sistema criptográfico con clave pública (RSA por las iniciales de sus creadores Ron Rivest, Adi Shamir y Len Adleman.
RUP: Rational Unified Process
SMTP: Simple Mail Transfer Protocol
SOAP: Simple Object Access Protocol
SSH: Secure Shell
SSL: Secure Socket Layer
TCP: Protocolo de Trasmisión de Información
TELNET: TELecommunication NETwork
UDDI: Universal Description, Discovery and Integration
UML: Lenguaje Unificado de Modelado
VM: Máquina Virtual
VPN: Virtual Private Network
W3C: World Wide Web Consortium
War: Archivo generado para el despliegue de una aplicación web implementada en java.
WSDL: Web Services Description Languages
WS-I: Web Services Interoperability Organization
WSPS: Web Services Protocol Stack
WS-Security: Web Service Security
XML: Extensible Markup Language
XML-RPC: XML Remote Producer Call
XP: Extreme Programming