Título: Transferencia segura de datos a través de internet.

Autores: Ing. Alain Arias Yanez [(1)]
Ing. Diosnel Quiñones Rosales [(2)]

1 Universidad de las Ciencias Informáticas-Facultad Regional de Granma, Cuba, aayanez@grm.uci.cu
2 Universidad de las Ciencias Informáticas-Facultad Regional de Granma, Cuba, dquinones@grm.uci.cu

Resumen

La presente investigación surge por la necesidad de proteger la información que es enviada a través de internet entre aplicaciones, dado que internet se concibe como una red abierta y por tanto insegura. En ésta 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 cliente-servidor del conocimiento de las tecnologías utilizadas en la transferencia de datos, brindándole un procedimiento sencillo para la integración en sus aplicaciones.

1. Introducción

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 y desarrollo de internet incrementó las expectativas y los esfuerzos en el estudio del tema referido. Internet ha sido concebida como una red de redes que para la interconexión descentralizada de computadoras a través de un conjunto de protocolos, permitiendo quemillones de personas compartan información (Dictionary, 2010). Por ello se desarrollan numerosas aplicaciones que aprovechan su capacidad bidireccional de manejar la información. 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 fallas 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.

¿Cómo lograr la transferencia segura de datos entre aplicaciones cliente-servidor?

Para implementar un sistema sobre internet es necesario tener en cuenta los siguientes requerimientos:

- seguridad; que no es más que la protección de las comunes y crecientes amenazas de internet.
- control; que es tener control total de toda la información que se envía (quién, qué, cuándo, dónde).
- integración; donde la transferencia de la información habitualmente se realiza entre diferentes entidades, por ello es importante que sea capaz de permitir a otros conectarse de manera eficiente y segura.

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 realiza 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.

Desarrollo

Herramientas y tecnologías

Para comenzar es necesario definir qué es un componente de software. Un componente de software puede ser desplegado independientemente y es sujeto a la composición de terceros, (Clemens, 1998). En el trabajo se exponen 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 conocer las particularidades de las tecnologías de comunicación utilizadas en el envío. Es importante además especificar las tecnologías utilizadas en su 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 (Oracle, 2010):

- 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.

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

- Crear aplicaciones en entornos 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" (Sevilla Ruiz, 2007). 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.

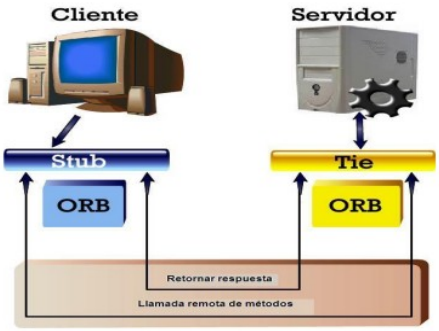


Figura 1. Estructura general de invocación de métodos remotos.

Tecnologías remotas

Sistema de Invocación Remota de Métodos (RMI)

El sistema de Invocación Remota de Métodos 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, (Oracle, 2010).

Servicios Hyper Text Transfer Protocol (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 éste en cuanto a la capacidad de atravesar los cortafuegos. (Interface21)

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 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 de software

Como se muestra en la Figura 1, 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.

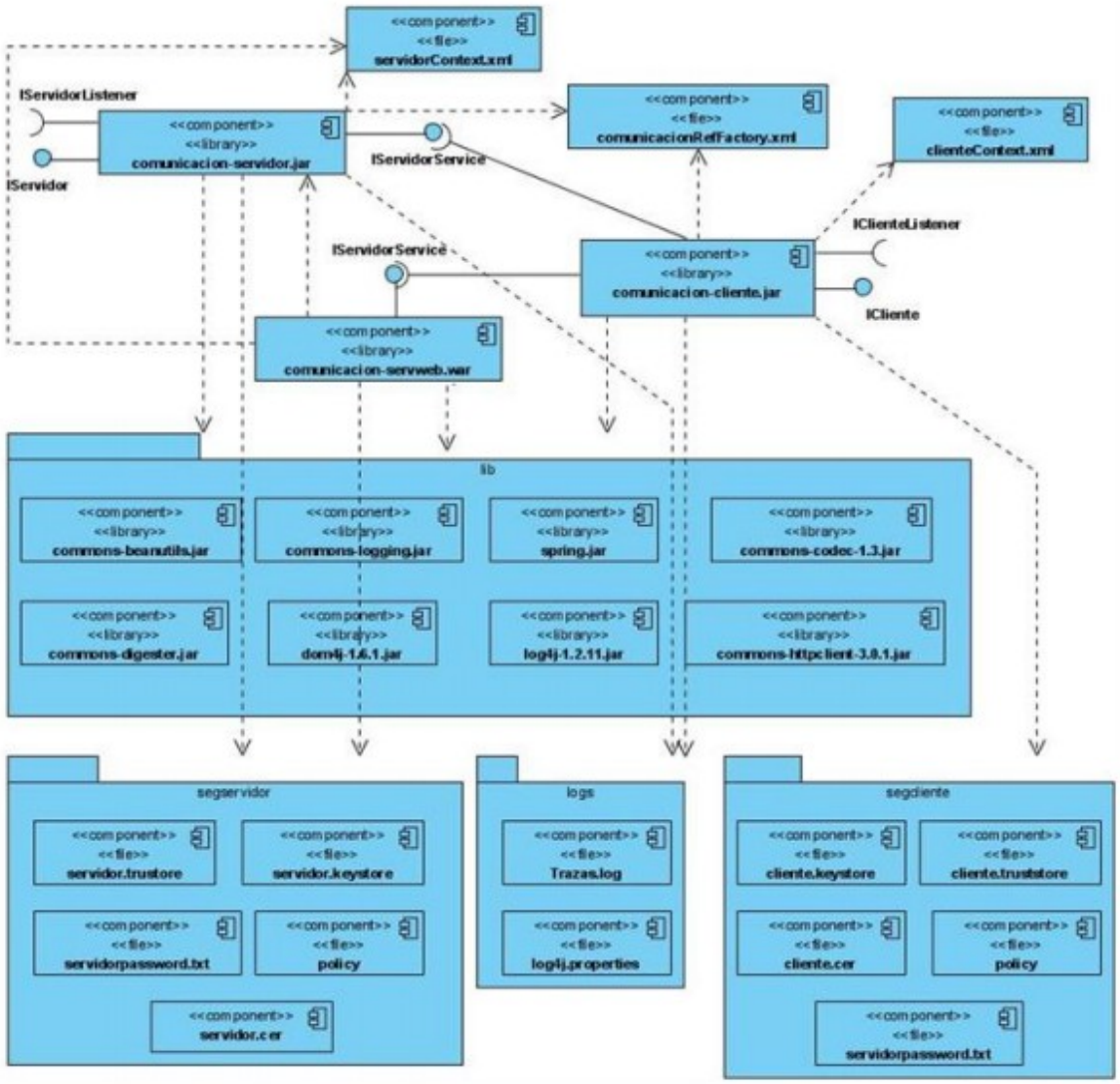


Figura 2. Diagrama de componentes que muestra las dependencias internas del componente.

Integración a sistemas

En el presente epígrafe se describe la configuración del componente para la integración a sistemas. Se define la forma de integración tanto para el cliente como para el servidor, así como las tecnologías remotas para la comunicació implementadas en esta versión del componente.

Publicación y acceso al servicio de las tecnologías remotas

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 se usan 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.

Cliente

Configuración del cliente

La configuración del cliente para el envío de la información se realiza a través de un archivo XML; el mismo contiene los atributos necesarios para la conexión al servidor, 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.

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 listenerpara cada petición al servidor.

3. Invocar el método enviar objeto (Object o; Obejct id) para el envío de la información, es posible el envío de cualquier objeto que tenga como clase base java.lang.Object.

4. Implementar la interfaz IClienteListener (Figura 3). 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.

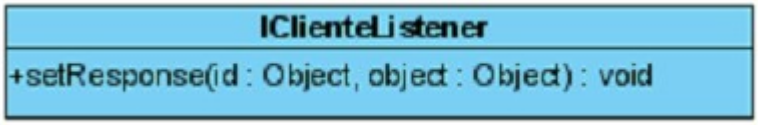


Figura 3. Interfaz IclienteListener.

A continuación se muestra un ejemplo:

```
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.

Realizar 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 enviar objeto (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() encargado de la llamada al procedimiento remoto.

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.

Publicar 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 (Figura 4), 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.

Figura 4.Interfaz IServidorListener.

A continuación se muestra un ejemplo.

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

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 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 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" />
```

Pruebas realizadas al componente de software desarrollado

En una primera etapa el componente fue evaluado internamente, haciendo énfasis fundamental en la seguridad a través de una aplicación capaz de analizar los datos enviados a través de la red.

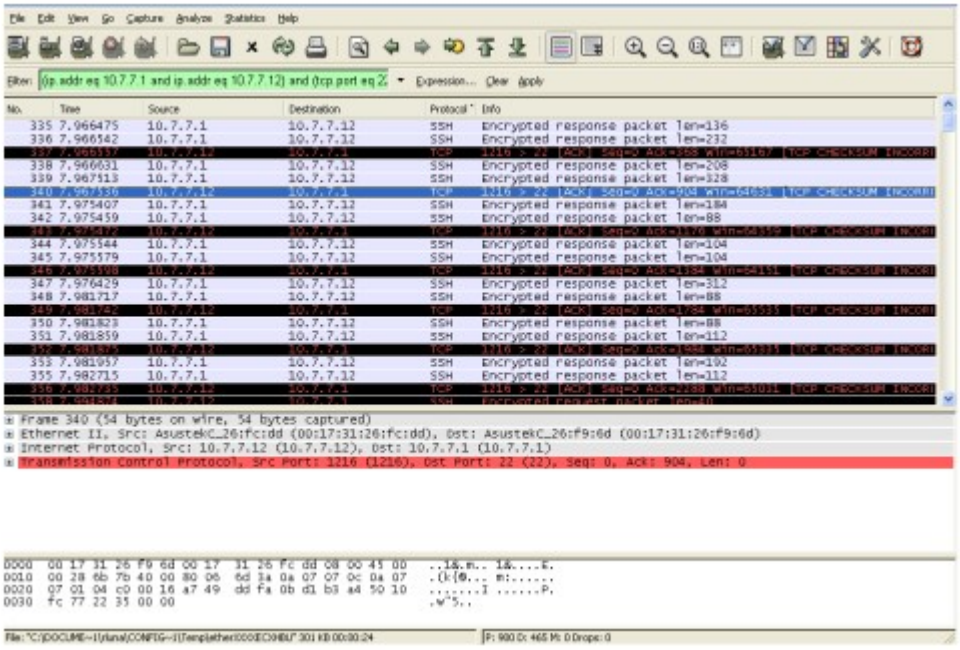


Figura 5. Imágenes capturadas de las pruebas utilizando Ethereal.

La imagen anterior muestra la pantalla principal de Ethereal en la se pueden observar las tramas de red entrantes y salientes de la PC. Luego se trata de decodificar la trama que viaja por SSL. La Figura 6 muestra la trama obtenida encriptada y no es posible decodificarla.

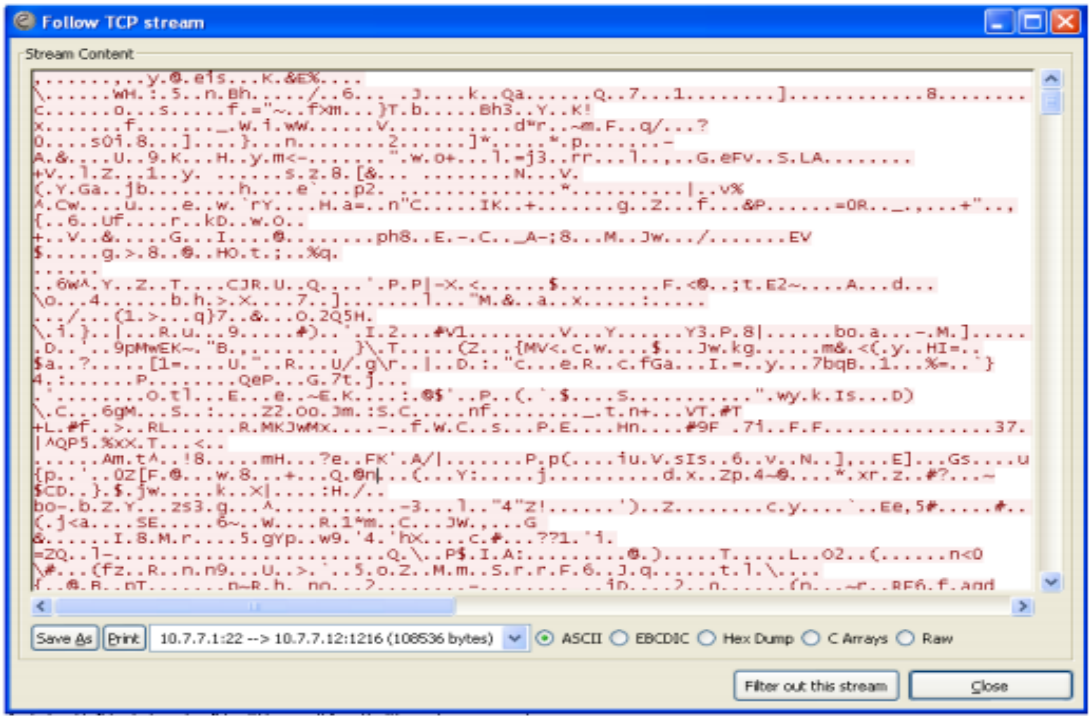


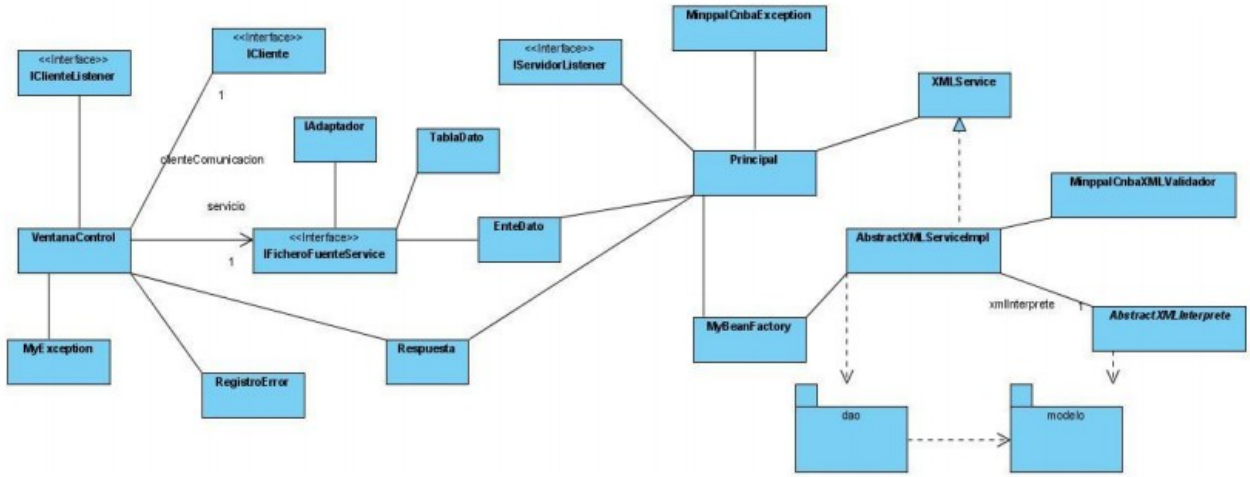
Figura 6. Imágenes capturadas de las pruebas utilizando Ethereal.

Caso de éxito

En otro nivel se realizaron pruebas al componente integrado a una aplicación de escritorio desarrollada para el proyecto “Solución Informática para el Centro Nacional de Balance

Alimentario de Venezuela", proyecto desplegado exitosamente en el Ministerio del Poder Popular para la Alimentación de Venezuela (MINPPAL). La arquitectura de esta solución tenía una Aplicación de Integración que se ejecuta como un proceso continuo encargado de recibir y almacenar los datos y una Aplicación de Escritorio que va a radicar en aquellos entes gubernamentales con presencia de datos digitalizados que debe llevar interfaz gráfica para la interacción con los usuarios.

La Figura 7 muestra el diagrama de clases muestra la integración del componente a la solución descrita.



Conclusiones

Con el resultado de la presente investigación se obtiene un componente de software "Ethereal" para la transferencia de datos a través de Internet entre aplicaciones cliente-servidor. El componente de software implementado es utilizado en una solución desplegada que valida su funcionalidad y garantía de uso en entornos cliente-servidor. Su diseño sobre la base de código libre y abierto permite su adaptación y la implementación de nuevas funcionalidades.

Referencias

- CLEMENS, SZYPERSKI. 1998. Component Software Component Software Beyond Object – Oriented Programming. 1998. p. 20-22.

- Dictionary, Tech Terms. Internet Definition. [en línea] 2010. [Consultado el: 13 de diciembre de 2010]. Disponible en: [http://www.techterms.com/definition/internet].

- FERNÁNDEZ IGLESIAS, NOÉ. Servicios HTTP Invoker. [en línea] 2006. [Consultado el: 5 de septiembre de 2010]. Disponible en: [http://di002.edv.uniovi.es/~cueva/asignaturas/doctorado/2006/trabajos/SW_ligeros.ppt].

- Interface21. Remoting and web services using Spring. [en línea] [Consultado el: 9 de noviembre de 2010]. Disponible en: [http://static.springsource.org/spring/docs/2.0.x/reference/remoting.html].

- Java Remote Method Invocation. Distributed Computing for Java. [en línea] 2010. [Consultado el: 6 de noviembre de 2010]. Disponible en: [http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html].

- ONYSZKO, TOMASZ. Authentication, Access Control & Encryption. [en línea] 2004. [Consultado el: 9 de septiembre de 2010.] Authentication, Access Control & Encryption.

- Oracle. Java. [en línea] 2010. [Consultado el: 11 de noviembre de 2010]. Disponible en: [http://www.java.com/es/about/].

- SEVILLA RUIZ, DIEGO. Aplicaciones distribuidas en Internet/Intranets: de los sockets a los Objetos Distribuidos. [en línea] 2007. [Consultado el: 13 de diciembre de 2010]. Disponible en: [http://ditec.um.es/~dsevilla/].