UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

Diseño e implementación de un sistema para envío de información de una terminal remota a un servidor

REPORTE DE INVESTIGACIÓN

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

Huilver Nolasco Aguilar

TUTOR:

Dr. Benjamín Macías Pimentel

2008

Agradecimientos

A mis padres Silvia Aguilar Cardoso y Rodolfo Nolasco Padilla, que a pesar de los muchos desatinos que he tenido en la vida, me han apoyado en todos los aspectos, gracias padres por ser como son.

A mi esposa Irma Flores Rivera por ser el pilar principal en mi familia y por aceptarme tal cual soy. A mis hijos que son el motor que me impulsa seguir cada día.

A mis amigos Eddy, Carlos, Daniel G., Edoardo, Julio Cesar, Abraham, y Daniel E., los cuales han aportado mucho tanto en mi formación académica como en lo personal.

Al Dr. Benjamín Macías P. por darme la oportunidad de ser parte de éste proyecto y de enseñarme que las cosas no siempre son tan difíciles de hacer.

A todos mis profesores que tuve en la carrera, una mención especial al Dr. José de Jesús Galaviz Casas ya que el fue el primer profesor en iniciarme en el camino de la programación.

Índice de contenido

Introducción general

El uso diario de dispositivos móviles tales como teléfonos celulares y PDAs (agenda electrónica) se ha venido incrementando y con ello viene la exigencia de los usuarios de estar comunicados con el mundo en todo momento y de tener la información necesaria para el desarrollo de su actividad laboral, en otras palabras cada día es mas importante que todos los dispositivos móviles tengan comunicación entre si y comunicación con servidores (computadoras) a través de una gran red conocida hoy día como Internet.

Sin embargo, el problema de mantener en línea (en Internet) a los dispositivos móviles en todo momento resulta complicado, ya que se requiere comunicación constante con el proveedor de telefonía, lo que lo hace caro y no práctico, pues esta comunicación puede romperse en cualquier momento porque depende, entre otras cosas, del terreno en donde nos encontremos. Por tanto, se propone una comunicación entre dispositivos móviles y computadoras de manera asíncrona, es decir, se solicita el servicio en cuanto halla red y en cuanto el servicio esté disponible, éste responde sin tener una comunicación constante.

Para llevar a cabo el tipo de comunicación planteada entre terminales remotas y computadoras se requieren muy diversos componentes, que van desde el hardware de los dispositivos móviles, el software que estos poseen, redes entre computadoras, así como las redes de telefonía local e internacional. Es por tanto lógico pensar que si hablamos de redes, dispositivos móviles y computadoras se deben tener en cuenta protocolos de comunicación, lenguajes de programación y sistemas.

En lo que resta del documento se van a describir una serie de cosas tales como redes de computadoras, protocolos de red, redes de telefonía, lenguajes de programación, etc. lo anterior para poner en contexto y respaldar el análisis, diseño e implementación de un sistema que permita integrar, coordinar y llevar a cabo la comunicación entre terminales remotas y servidores.

El presente documento se complementa con el documento titulado "Análisis, diseño e implementación de un sistema para envío de información de un servidor a una terminal remota"[1], ambos documentos corresponden a la especificación de la primera fase del proyecto titulado "Sistemas Remotos de Interacción con Internet"[2], por lo que se recomienda la lectura de [1] y [2].

Objetivos

Los objetivos de la realización de este proyecto se listan a continuación:

- Análisis y diseño de un sistema informático para transferir información entre un servidor y terminales remotas a nivel básico.

- Implementación y documentación de un sistema para el envío de información desde un dispositivo móvil (terminal remota) hacia un servidor de manera asíncrona.

- Implementación y documentación de una arquitectura que permita proveer servicios.

Introducción a protocolos y lenguajes de programación para comunicaciones remotas

Un teléfono móvil o celular, es un dispositivo de comunicación electrónico portátil e inalámbrico con las mismas capacidades básicas de un teléfono de línea telefónica convencional. Un teléfono celular se comunica con los otros teléfonos (móviles y fijos) a través una red de telefonía móvil o celular, que consiste en una red de estaciones transmisoras-receptoras de radio y una serie de centrales telefónicas de conmutación.

La comunicación entre personas por medio de teléfonos móviles no solo es con voz sino también por medio de mensajes de texto (SMS Short Message Service) los cuales se han convertido en una forma muy frecuente de comunicación por su bajo costo y sencillez.

Las computadoras se comunican entre si por medio de protocolos de red tales como TCP/IP, TELNET, SSH, FTP, POP3 y HTTP entre otros. Los servidores pueden establecer conexión con terminales remotas vía alambrada por medio de un cable con salida a USB o por medio inalámbrico mediante la tecnología *Bluetooth.*

El desarrollo del proyecto "Sistemas Remotos de Interacción con Internet" requiere de diversos protocolos y tecnologías, que permitan la transferencia de datos entre las terminales remotas y un servidor, así como la comunicación entre dicho servidor y otros servidores remotos a través de Internet.

Los protocolos que se describirán en este documento son: FTP, HTTP, POP3, SMTP TELNET y SSH. Estos protocolos están basados en el modelo CLIENTE-SERVIDOR, el cual también es descrito en el presente documento; otras tecnologías que son presentadas en este documento son: SMS, WAP (*Wireless Application Protocol*) y *Bluetooth*.

El lenguaje que se describe para el desarrollo de aplicaciones sobre dispositivos móviles es Java J2ME y el lenguaje para aplicaciones sobre servidores es Java J2SE.

Telefonía Móvil

Básicamente existen dos tipos de redes de telefonía móvil:

TMA (Red de telefonía móvil analógica). Como su propio nombre indica, en esta red la comunicación se realiza mediante señales vocales analógicas tanto en el tramo radioeléctrico como en el terrestre. En su primera versión funcionó en la banda radioeléctrica de los 450 MHz, trabajando posteriormente en la banda de los 900 Mhz.

Red de telefonía móvil digital. En esta red la comunicación se realiza mediante señales digitales, lo que permite optimizar tanto el aprovechamiento de las bandas de radiofrecuencia como la calidad de

transmisión. Su exponente más significativo en el ámbito público es el GSM (*Global System For Mobile Communications*) y su tercera generación UMTS.

GSM (Global System For Mobile communications)

Antes de definir y describir GSM, es necesario entender los siguientes términos:

1. Canal de comunicación - el camino único facilitado mediante un medio de transmisión que puede ser: con separación física, tal como un par de cables o con separación eléctrica, tal como la multiplexación por división de frecuencia (MDF) o por división de tiempo (MDT).
2. Ancho de banda— es el rango del limite del canal; entre más grande el ancho de banda, mayor será la velocidad de transmisión.
3. Bits por segundo (bps) — un solo pulso (*on-off*) de dato; ocho bits conforman un byte.
4. Frecuencia- el número de ciclos por unidad de tiempo; la frecuencia es medida en hertz (Hz).
5. Kilo (k)- es la designación para 1000, la abreviación kbps representa 1000 bits por segundo.
6. Megahertz (MHz)- 1000000 de hertz (ciclos por segundo).
7. Milisegundos (ms)- milisegundos.
8. Watt (W)- una medida de poder de un transmisor.

DESCRIPCION GENERAL

GSM (*Global System For Mobile communications*) es una tecnología digital cuya primera funcionalidad es la transmisión de voz, pero que también permite la transmisión de datos a baja velocidad: 9.6 kbit/s. El GSM utiliza la tecnología TDMA (*Time Division Multiple Access*) para el uso de los canales de comunicación.

TDMA es una tecnología que distribuye las unidades de información en ranuras (*slots*) alternas de tiempo, proveyendo acceso múltiple a un reducido número de frecuencias. TDMA es una tecnología inalámbrica de segunda generación que brinda servicios de alta calidad de voz y datos.

TDMA divide un único canal de frecuencia de radio en varias ranuras de tiempo (ocho en GSM). A cada persona que hace una llamada se le asigna una ranura de tiempo específica para la transmisión, lo que hace posible que varios usuarios utilicen un mismo canal simultáneamente sin interferir entre sí.

ESPECIFICACIONES

Las especificaciones del GSM son las siguientes:

1. Banda de frecuencia- el rango de frecuencia especificado para el GSM es de 1850 a 1900MHz.

2. Distancia duplex - la distancia duplex es de 80 Mhz. La distancia duplex es la distancia entre la frecuencia superior y la frecuencia inferior.
3. Separación de canales- la separación entre frecuencias adyacentes es de 200 kHz.
4. Modulación- la modulación es el proceso de enviar una señal cambiando las características de la frecuencia portadora. Esto se hace en GSM mediante GMSK (*Gaussian Minimum Shift Keying*)
5. Tasa de transmisión- GSM es un sistema digital con una tasa de transferencia en aire de 270kbp.
6. Código de palabra- GSM utiliza LPC (*Linear Productive Coding*).

GPRS(General Packet Radio Service)

Es considerada la generación 2.5, entre la segunda generación (GSM) y la tercera (UMTS).Proporciona altas velocidades de transferencia de datos y se utiliza en las redes GSM.

GPRS es sólo una modificación de la forma de transmitir datos en una red GSM, pasando de la conmutación de circuitos en GSM a la conmutación de paquetes.

GPRS es básicamente una comunicación basada en paquetes de datos. Los intervalos de tiempo (*timeslots*) se asignan en GSM generalmente mediante una conexión conmutada, pero en GPRS los intervalos de tiempo se asignan a la conexión de paquetes, mediante un sistema basado en la necesidad. Esto significa que si no se envía ningún dato por el usuario, las frecuencias quedan libres para ser utilizadas por otros usuarios. Dicho en otras palabras, la conmutación por paquetes hace que el usuario sólo use la red al enviar o recibir unos paquetes de información, lo que trae como consecuencia que el uso de red sea mas barato en GPRS que en GSM.

UMTS (Universal Mobile Telecommunications System)

La tecnología UMTS (*Universal Mobile Telecommunications System*) es el sistema de telecomunicaciones móviles de tercera generación, que evoluciona desde GSM y pasando por GPRS.

UMTS utiliza la tecnología WCDMA (*Wide Code Division Multiple Access*) heredada de la tecnología militar. WCDMA consiste en que la señal se expande en frecuencia gracias a un código de ensanchado que sólo conocen el emisor y el receptor. Esta original forma de modulación tiene numerosas ventajas:

- Altas velocidades de transmisión de hasta 2 Mbps, al usar todo el espectro.
- Alta seguridad y confidencialidad debido a la utilización de técnicas que permiten acercarse a la capacidad máxima del canal.
- Acceso múltiple de eficacia máxima mientras no coincidan las secuencias de saltos.

- Alta resistencia a las interferencias.
- Posibilidad de trabajar con dos antenas simultáneamente debido a que siempre se usa todo el espectro y lo importante es la secuencia de salto, lo que facilita el proceso de traspaso de la señal de una antena a otra (*handover*), donde GSM falla mucho.
- UMTS ofrece otra serie de ventajas como *roaming* y cobertura a nivel mundial ya sea vía enlace radio terrestre o vía satélite, y está altamente estandarizado con una interfaz única para cualquier red.

Modelo CLIENTE - SERVIDOR

Existen varias definiciones del modelo CLIENTE-SERVIDOR, algunas de las cuales son listadas a continuación.

- "Cualquier combinación de sistemas que pueden colaborar entre si para dar a los usuarios toda la información que ellos necesiten sin que tengan que saber dónde esta ubicada"[3].
- "Arquitectura de procesamientos cooperativo donde uno de los componentes solicita servicios a otro"[3].

Los elementos principales del modelo CLIENTE-SERVIDOR son justamente el elemento llamado cliente y el otro elemento llamado servidor. Por ejemplo, dentro de un ambiente multimedia, el elemento cliente sería el dispositivo que puede observar el video, cuadros y texto, o reproduce el audio distribuido por el elemento servidor.

Por otro lado, el cliente también puede ser una computadora personal o una televisión inteligente que posea la capacidad de entender datos digitales. Dentro de este caso, el elemento servidor es el depositario del video digital, audio, fotografías digitales y texto y los distribuye bajo demanda de ser una máquina que cuenta con la capacidad de almacenar los datos y ejecutar todo el software que brindan éstos al cliente.

En resumen, el modelo CLIENTE-SERVIDOR es una forma de dividir y especializar programas y equipos de cómputo, en la cual la entidad llamada servidor es un proveedor de servicios, los que son consumidos por la entidad llamada cliente. La interacción entre cliente y servidor es por medio de un mecanismo de mensajes que consiste en la petición del servicio y la respuesta del servicio pedido.

El modelo CLIENTE-SERVIDOR posee las siguientes características:

- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de cliente y servidor pueden estar en plataformas separadas, o en la misma plataforma.
- Un servidor da servicio a múltiples clientes en forma concurrente.
- Cada plataforma puede ser escalable independientemente. Los cambios

realizados en las plataformas de los clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

- Un sistema de servidores realiza múltiples funciones, al mismo tiempo que presenta una imagen de un solo sistema a las estaciones Clientes. Esto se logra combinando los recursos de cómputo que se encuentran físicamente separados en un solo sistema lógico, proporcionando de esta manera el servicio más efectivo para el usuario final.
- También es importante hacer notar que las funciones CLIENTE-SERVIDOR pueden ser dinámicas. Ejemplo, un servidor puede convertirse en cliente cuando realiza la solicitud de servicios a otras plataformas dentro de la red.
- Su capacidad para permitir integrar los equipos ya existentes en una organización, dentro de una arquitectura informática descentralizada y heterogénea.

Protocolos

Los protocolos son reglas y procedimientos para la comunicación entre entidades. El término "protocolo" se utiliza en distintos contextos. Por ejemplo, los diplomáticos de un país se ajustan a las reglas del protocolo creadas para ayudarles a interactuar de forma correcta con los diplomáticos de otros países. De la misma forma se aplican las reglas del protocolo al entorno informático. Cuando dos equipos están conectados en red, las reglas y procedimientos técnicos que dictan su comunicación e interacción se denominan protocolos.

Así que se puede decir que un protocolo de red o protocolo de comunicación es el conjunto de reglas que controlan la secuencia de mensajes que ocurren durante una comunicación entre entidades que forman una red. En este contexto, las entidades de las cuales se habla son programas de computadora o dispositivos electrónicos capaces de interactuar en una red.

Los elementos que definen un protocolo son:

- Sintaxis: formato, codificación y niveles de señal de datos.
- Semántica: información de control y gestión de errores.
- Temporización: coordinación entre la velocidad y orden secuencial de las señales.

Los protocolos de red pueden dividirse en varias categorías, una de las cuales es la categoría de TCP/IP, llamada así en referencia a los protocolos de Internet más usados, los cuales son: el Protocolo de Control de Transmisión (TCP) y el Protocolo de Internet (IP).

Es conveniente por tanto dar una breve descripción de cada nivel de la categoría TCP/IP además de mostrar ejemplos de protocolos pertenecientes a cada nivel.

EL NIVEL FÍSICO

Describe las características físicas de la comunicación, como las convenciones sobre la naturaleza del medio usado para la comunicación (como las comunicaciones por cable, fibra óptica o radio), y todo lo relativo a los detalles como los conectores.

Ejemplos: Cable coaxial, Cable de fibra óptica, Cable de par trenzado, Microondas, Radio, Palomas mensajeras, RS-232.

EL NIVEL DE ENLACE DE DATOS

Especifica cómo son transportados los paquetes sobre el nivel físico, incluidos los delimitadores (patrones de bits concretos que marcan el comienzo y el fin de cada trama).

Ejemplos: Ethernet, Fast Ethernet, Gigabit Ethernet, Token Ring, FDDI, ATM, HDLC.

EL NIVEL DE RED

Soluciona el problema de conseguir transportar paquetes de datos a través de una red sencilla.

Ejemplos: ARP, RARP, IP (IPv4, IPv6), ICMP, IGMP, NetBEUI, IPX.

EL NIVEL DE TRANSPORTE

Los protocolos del nivel de transporte pueden solucionar problemas como la fiabilidad ("¿alcanzan los datos su destino?") y la seguridad de que los datos llegan en el orden correcto.

Ejemplos: TCP, UDP, SPX.

EL NIVEL DE APLICACIÓN

Es el nivel que los programas más comunes utilizan para comunicarse a través de una red con otros programas. Los procesos que acontecen en este nivel son aplicaciones específicas que pasan los datos al nivel de aplicación en el formato que internamente use el programa y es codificado de acuerdo con un protocolo estándar.

Ejemplos: SNMP, SMTP, NNTP, FTP, SSH, HTTP, SMB/CIFS, NFS, TELNET, IRC, ICQ, POP3, IMAP.

SMS (Short Message Service)

El SMS es un mecanismo de envío de mensajes de texto cortos y de contenido binario a través de redes móviles.

Los mensajes son enviados vía un mecanismo de "almacenamiento-y-envío" a una central de servicio de mensajes cortos (SMSC), la cual enviará el mensaje cuando el receptor este disponible. El envío de mensajes no está garantizado, por lo que el emisor puede obtener un pequeño mensaje de notificación de entrega del mensaje.

La transición de mensajes cortos entre el SMSC y un teléfono móvil puede ser realizada a través de diferentes protocolos tales como el SS7 o TCP/IP.

Ventajas

- Proporciona un método conveniente para el intercambio de pequeñas cantidades de información entre usuarios de dispositivos móviles.
- Bajo costo de envío.
- En muchas situaciones es más cómodo enviar un mensaje SMS que hablar por teléfono.

Limitaciones

- Solo puede ser enviado un mensaje de texto plano o un mensaje con contenido binario.
- Los mensajes son limitados por su tamaño. Un mensaje SMS no puede exceder los 160 caracteres.
- Muchos protocolos propietarios son usados por los operadores SMS y los desarrolladores necesitan implementar diferentes interfaces para hacer que sus aplicaciones funcionen con diferentes centrales SMS.
- El protocolo de unidades de datos del SMS definido en GSM 03.40 no es eficiente.

WAP (Wireless Application Protocol)

WAP proviene de *Wireless Application Protocol* o Protocolo de Aplicación Inalámbrica. Es un protocolo con el que se ha tratado de dotar a los dispositivos móviles de un pequeño y limitado navegador Web. WAP exige la presencia de una puerta de enlace encargada de actuar como intermediario entre Internet y el terminal. Esta puerta de enlace o *gateway* es la que se encarga de convertir las peticiones WAP a peticiones Web habituales y viceversa.

Las páginas que se transfieren en una petición usando WAP no están escritas en HTML, si no que están escritas en WML, un subconjunto de éste. WAP ha sido un gran avance, pero no ha resultado ser la herramienta que se prometía. La navegación es muy engorrosa (la introducción de URLs largas por teclado es muy pesada, además de que cualquier error en su introducción requiere que se vuelva a escribir la dirección completa por el teclado del móvil). Además su costo es bastante elevado ya que el pago de uso de esta

tecnología se realiza en base al tiempo de conexión a una velocidad, que no es muy buena.

HTTP (Hipertext Transfer Protocol)

El HTTP (*Hypertext Transfer Protocol*) es el protocolo de transferencia de hipertexto. El hipertexto es el contenido de las páginas Web, así que el HTTP es un protocolo de transferencia mediante el cual se envían las peticiones de acceso y respuesta a una página Web. El HTTP es un protocolo estándar que no posee estado y que puede ser usado para otras tareas además de su uso para el hipertexto, tales como el manejo de servidores de nombres y de sistemas distribuidos a través de la extensión de sus métodos de solicitud, códigos de error y encabezados.

Una característica importante del HTTP es la negociación de la representación de datos lo cual permite a los sistemas ser construidos independientemente de los datos que se estén transfiriendo.

El formato tanto del mensaje como de la respuesta es como sigue:

```
<Línea inicial>
Encabezado-1: valor-1
...
Encabezado-n: valor-n

<Cuerpo del mensaje (Opcional)>
```

La "línea inicial'' es diferente en las solicitudes y en las respuestas. Para el caso de una solicitud los campos requeridos son: método, recurso y versión del protocolo; un ejemplo de esto es "GET /ruta/al/archivo/index.html HTTP/1.0".

Para el caso de una respuesta, los campos requeridos son: versión del protocolo, código de respuesta y mensaje; un ejemplo de esto es "HTTP/1.0 200 OK".

Los encabezados están especificados en el protocolo, e incluyen, en el caso de una solicitud, información del navegador y eventualmente del usuario cliente; en el caso de una respuesta, información sobre el servidor y sobre el recurso. El cuerpo del mensaje contiene el recurso a transferir o el texto de un error en el caso de una respuesta.

FTP (File Transfer Protocol)

El FTP (*File Transfer Protocol*) es un protocolo que sirve para el envío de grandes cantidades de datos (archivos) a través de cualquier red con soporte TCP/IP.

Los objetivos del FTP:

- Promover que los usuarios compartan archivos.
- Implícitamente anima el uso de computadoras remotas.
- Transferencia de datos de forma segura y eficiente.

Los inconvenientes del FTP:

- La contraseña y los contenidos son enviados en texto plano por lo que representa un riesgo de seguridad.
- El funcionamiento de este protocolo requiere de muchas conexiones TCP/IP.
- Se puede abusar de las características del protocolo para usar un Proxy.
- Es un protocolo con mucho retraso debido al número de comandos requeridos para iniciar una transferencia.
- No corrobora la integridad de datos en el lado del receptor.

POP3 (Post Office Protocol 3)

El protocolo POP3 permite a cualquier estación de trabajo o computadora tener acceso de forma dinámica a un contenedor de correo en un servidor, es decir que el protocolo POP3 es usado para permitir que una estación de trabajo obtenga el correo alojado en un servidor.

El POP3 nos proporciona una serie de comandos para la manipulación del correo en el servidor de correo; las operaciones más comunes son la descarga de correo y eliminación del correo del servidor.

SMTP (Simple Mail Transfer Protocol)

El SMTP es un protocolo simple y basado en texto cuyo objetivo es la transferencia de correos electrónicos de forma segura y eficiente.

TELNET

El propósito del protocolo TELNET es brindar al usuario la facilidad de establecer una comunicación bi-direccional entre dos servidores mediante Internet. TELNET fue diseñado para emular una Terminal asociada a otra computadora, es decir, una Terminal remota.

En muchos sistemas, los programas clientes de TELNET pueden ser usados para interactuar con sesiones TCP y para establecer comunicación con servicios como POP3 sin un software adicional.

Desventajas:

- Generalmente los servidores TELNET tienen muchas vulnerabilidades que han sido descubiertas a lo largo de los años y quizás otras aun no han sido descubiertas
- Por defecto, TELNET no cifra la información que envía sobre la conexión, incluyendo contraseñas.

SECURE SHELL

SSH (*Secure Shell*) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo el ordenador mediante un intérprete de comandos, y también puede redirigir el tráfico de X para poder ejecutar programas gráficos, si tenemos un Servidor X arrancado.

Además de la conexión a otras máquinas, SSH nos permite copiar datos de forma segura (tanto ficheros sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectar las máquinas y pasar los datos de cualquier otra aplicación por un canal seguro de SSH.

Ventajas de *Secure Shell* sobre TELNET

SSH trabaja de forma similar a como se hace con TELNET. La diferencia principal es que SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión; aunque es posible atacar este tipo de sistemas por medio de ataques de REPLAY y manipular así la información entre destinos.

Otras tecnologías

Además de los protocolos descritos en la sección anterior es necesario describir otro tipo de tecnologías que si bien no son exclusivas de los dispositivos móviles, si son de uso muy común en ellos, ya que les permite extender su gama de comunicación. La tecnología que se describe en esta sección es *Bluetooth*.

Bluetooth

El *Bluetooth* es una especificación industrial para redes inalámbricas personales. El *Bluetooth* proporciona una manera de conectar e intercambiar información entre dispositivos móviles tales como asistentes digitales personales, teléfonos celulares, laptops, PCs, impresoras y otros periféricos.

Características

- Tecnología inalámbrica. Reemplaza la conexión alambrada en distancias que no excedan los 10 metros, alcanzando velocidades del rango de 1Mbps.
- Comunicación automática. La estructura de los protocolos que lo forman favorece la comunicación automática sin necesidad de que el usuario la inicie.
- Bajo consumo de energía. Lo pequeño de los dispositivos y su portabilidad requieren de un uso adecuado de la energía.

- Integración de servicios. Puede soportar transmisiones de voz y datos de manera simultánea.
- Transmisión en diversas direcciones. Debido a que basa su comunicación en radiofrecuencia, no requiere línea de vista y permite configuraciones punto-multipunto.
- Seguridad. Utiliza *Spread Spectrum Frequency Hopping* como técnica de multiplexaje, lo que disminuye el riesgo de que las comunicaciones sean interceptadas o presenten interferencia con otras aplicaciones. Provee también especificaciones para autenticar dispositivos que intenten conectarse a la red *Bluetooth*, así como cifrado en el manejo de llaves para proteger la información.
- Establecimiento de redes. Tiene la característica de formar redes en una topología donde un dispositivo hace las veces de maestro y hasta siete más operan como esclavos. Esta configuración se conoce como *piconet*. Un grupo de *piconets*, no más de diez, es referido como *Scatternet*.

Lenguajes de programación

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora y consiste en un conjunto de reglas sintácticas y semánticas.

Un lenguaje de programación permite a un programador especificar de manera precisa: sobre qué datos debe operar una computadora, cómo deben ser almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico.

Java

Java es un lenguaje de programación orientado a objetos desarrollado por James Gosling y sus colegas en Sun Microsystems a inicios de los años 90's. Java no esta diseñado para generar código nativo sino *bytecode* que es ejecutado por la JVM (*Java Virtual Machine*).

Java tiene una sintaxis muy parecida a C y C++, pero con un modelado de objetos más simple y a diferencia de C, no tiene manejo a bajo nivel como son los apuntadores.

Las características principales de Java son:

- Es un lenguaje orientado a objetos.
- Es independiente de la plataforma.
- Tiene Soporte integrado para usar redes de computadoras.
- Permite la ejecución remota de código de manera segura.
- Tiene un modelado de objetos sencillo.

Existen tres ediciones distintas de Java, cada una enfocada a cubrir ciertas necesidades de los usuarios, estas versiones son: J2SE (*Java Standard Edition*) orientada al desarrollo de aplicaciones independientes de la plataforma, J2EE (*Java Enterprise Edition*) orientada al entorno empresarial y J2ME (*Java Micro Edition*) orientada a dispositivos con capacidades restringidas.

J2SE

Esta versión de Java contiene el conjunto básico de herramientas usadas para desarrollar Java *Applets*, así cómo las APIs (*Application Programming Interface*) orientadas a la programación de aplicaciones de usuario final: Interfaz gráfica de usuario, multimedia, redes de comunicación, etc.

J2EE

Esta versión está orientada al entorno empresarial. El software empresarial tiene unas características propias marcadas: está pensado no para ser ejecutado en un equipo, sino para ejecutarse sobre una red de ordenadores de manera distribuida y remota mediante EJBs (*Enterprise Java Beans*). De hecho, el sistema se monta sobre varias unidades o aplicaciones. En muchos casos, además, el software empresarial requiere que se sea capaz de integrar datos provenientes de entornos heterogéneos. Esta edición está orientada especialmente al desarrollo de servicios Web, servicios de nombres, persistencia de objetos, XML, autenticación, APIs para la gestión de transacciones, etc. El cometido de esta especificación es ampliar la J2SE para dar soporte a los requisitos de las aplicaciones de empresa.

J2ME

Esta versión de Java está enfocada a la aplicación de la tecnología Java en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs o electrodomésticos "inteligentes". Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (*Kilo Virtual Machine*, debido a que requiere sólo unos pocos Kilobytes de memoria para funcionar) en vez del uso de la JVM clásica, y la inclusión de un pequeño y rápido recolector de basura.

Java ME esyá dividido en configuraciones, perfiles y paquetes opcionales.

Configuraciones

Las configuraciones son especificaciones que detallan una maquina virtual y un conjunto de bibliotecas base que proporcionan el API necesario para ser usado con cierto tipo de dispositivo. Estas bibliotecas proveen la funcionalidad base para un rango particular de dispositivos que tienen características en común, tales como conectividad de red y poca cantidad de memoria.

Actualmente hay dos tipos de configuraciones de Java ME las cuales son: CLDC (Connected *Limited Device Configuration*) y CDC (*Connected Device Configuration*).

Perfiles

Un perfil complementa a una configuración agregando un API más específico para hacer un entorno de ejecución completo que permita correr aplicaciones en una categoría específica de dispositivos. Un perfil es un conjunto de APIs de alto nivel que define el modelo de ciclo de vida de la aplicación, la interfaz de usuario, almacenamiento persistente y acceso a propiedades del dispositivo.

Un ejemplo ampliamente usado es la combinación del CLDC con el perfil MIDP (*Mobile Information Devise Profile*) para proporcionar un entorno de aplicación completo de Java para teléfonos celulares y otros dispositivos

con características similares.

Opcionales

Los paquetes opcionales agregan funcionalidad a la plataforma Java ME tales como: conectividad a una base de datos, mensajes inalámbricos, multimedia, gráficos 3D y servicios Web.

Análisis y diseño de un sistema informático para transferir información entre un servidor y terminales remotas a nivel básico

A continuación se describe el sistema entero que realice durante el periodo de trabajo en el proyecto. El énfasis estuvo en ir de la fase de especificación al diseño; a pesar de ello, algunas partes fueron implementadas para demostrar que el análisis era correcto.

Especificación y diseño alternativos del sistema que transfiere información entre el servidor y las terminales remotas a nivel básico

En esta sección se describe el diseño de un sistema que permite la transferencia de información desde una terminal remota hacia un servidor y viceversa, además del diseño de algunas clases necesarias para proveer servicios.

Protocolo de comunicación

Para llevar a cabo la comunicación entre terminales remotas y un servidor, cuyo propósito sea proveer los servicios especificados en el proyecto, se debe definir un protocolo que permita establecer dicha comunicación.

En dicho protocolo se definen los tipos de servicio, así como los parámetros necesarios para ofrecer dichos servicios. Cabe destacar que si se desean agregar nuevos servicios en el futuro, se debe extender este protocolo.

El protocolo mencionado es descrito a continuación además se muestra un ejemplo por cada servicio que debe ser proporcionado por el servidor.

Autenticar al usuario con el servidor

Instrucción	parámetros	descripción
Lng		Indica al servidor que se debe autenticar un usuario
Usr	Nombre de usuario	Indica al servidor que el usuario pasado como parámetro será autenticado
Pass	Contraseña	Usado junto con la instrucción Usr para autenticar al usuario

Ejemplo:

Envío del mensaje.

Lng
User miusuario
Pass micontraseña

Recepción de un mensaje especificando si la sesión para el usuario dado fue iniciada o rechazada.

Envío de correo:

Instrucción	parámetros	descripción
MaF	Remitente (cuenta de correo)	Indica que el remitente enviará un correo
Pass	Contraseña (de la cuenta de correo)	Indica la contraseña (de la cuenta de correo) del remitente
MaT	Destinatario	Indica el destinatario al cual será enviado el correo
Subj	Encabezado	Indica el encabezado del mensaje
Mess	Mensaje	Indica el mensaje que será enviado
EMess		Indica el fin del mensaje. Hasta que este comando sea especificado el correo será enviado

Ejemplo:

Envío del mensaje.

MaF bob@gmail.com
Pass micontraseña
MaT alice@gmail.com
Subj prueba de correo
Mess El mensaje que será enviado por correo

Recepción del mensaje notificando que ingrese la continuación del mensaje.

Envío del mensaje.

Continuación del mensaje de correo
EMess

Recepción de notificación de fin de proceso.

Recepción de correo

Instrucción	parámetros	descripción
GMa		Indica al servidor que se iniciará la obtención de encabezados de correo
Email	Dirección de correo	Indica la dirección de correo de la que se obtendrán los encabezados de la cuenta de correo
Pass	Contraseña	Contraseña de la cuenta de correo
Nmail	Número de correo a revisar	Se enumerarán los encabezados obtenidos para luego ser elegidos por número

Ejemplo:

Envío del mensaje:

GMa
Email bob@gmail.com
Pass micontraseña

Recepción de los encabezados de los correos recibidos con un número asociado a ellos.

Envío del mensaje:

Nmail 2

Recepción del correo número 2 a revisar.

Búsqueda de información en alguna red en particular Internet

Instrucción	parámetros	descripción
Srch	Frase a buscar	Es la frase que se buscará desde una red

NMsr	Número máximo de paginas buscadas	Define un valor máximo para el resultado de búsqueda de paginas que contengan la frase solicitada

Ejemplo:

Envío de mensaje

Srch computación
NMsr 3

Recepción de un sumario de información relacionada con la palabra computación

Petición de noticias de algún tópico

Instrucción	parámetros	descripción
News	Frase a buscar	Es la frase que se buscará desde un servidor de noticias

Ejemplo:

Envío del mensaje

News elecciones

Recepción de las noticias asociadas a la palabra elecciones.

Ejecución de un comando remoto en un servidor asociado al usuario dado

Instrucción	parámetros	descripción
Rcmd	Comando file	Comando a ejecutar en un servidor que está asociado con el usuario dado, en particular puede ser la computadora personal de dicho usuario
Server	dirección web	IP del servidor en el cual se desea ejecutar el comando
Port	Número de puerto	Puerto abierto para ejecutar comandos a través de una terminal remota
User	Usuario	Usuario registrado en el servidor que se

		desea ejecutar el comando
Pass	Contraseña	Contraseña del usuario especificado

Ejemplo:

Envío del mensaje

```
Rcmd playS miCancion.mp3
Server 192.68.0.1
Port 1024
User bob
Pass micontraseña
```

Recepción de notificación de terminación de ejecución.

Diseño de clases

A partir del análisis de los requerimientos funcionales, describiremos un diseño de clases que abstraigan los distintos casos de uso en base al paradigma de orientación a objetos.

Antes de iniciar el diseño de clases es necesario mencionar las entidades involucradas en el intercambio de información, las cuales son:

- La terminal remota del usuario que solicita los servicios a la que nombraré TRC.
- El servidor que provee los servicio a la que nombraré SRV.
- La terminal remota que sirve como puente entre TRC y SRV a la que nombraré TRS.

En principio, se nota la necesidad de un intercambio de información entre la TRS que recibe los SMS de los usuarios y SRV que satisface las peticiones. De esta manera, se establece una clase abstracta llamada Conmutador que definirá la forma de intercambiar información entre ambas cosas.

Clase abstracta Conmutador

Descripción

En esta clase, se debe tener el nombre del servidor y el puerto asignado para escuchar peticiones.

Al ser atributos de la clase, se crearán métodos para asignarles valor y otros para solicitar los mismos; ésta es la base para los procesos

que estarán verificando la solicitud de servicios, por lo tanto, se debe tener un método que "escuche" cuando un servicio es solicitado.

Atributos

nameServer:guarda el nombre del servidor.

port:guarda el puerto de solicitud de un servicio.

Métodos

setNameServer(String):asigna un nombre al servidor de aplicaciones.

getNameServer():devuelve el valor del nombre del servidor.

setNumPort(Integer):asigna un número de puerto para conexión al servidor.

getNumPort():regresa el valor del puerto.

listen(String):verifica cuando ocurre un evento de llegada de información para un servicio solicitado.

Clase ConmutadorTR

Una primera implementación de la interfaz Conmutador es necesaria del lado de TRS. A esta se le llamará ConmutadorTR.

Adicionalmente a la implementación de la interfaz, ConmutadorTR debe tener un conjunto de métodos para poder recibir los mensajes SMS, formatearlos y enviar la información de los mismos, en forma de archivo al servidor.

Atributos

path: guarda el lugar donde se escriben los archivos de respuesta por parte de los servicios.

Métodos

sendFile(String):envía un archivo con un formato del servicio solicitado.

receiveFile(String):recibe un archivo como respuesta a un servicio.

listenAndFilter(String):verifica cuando recibe un SMS, y lo filtra al servicio correspondiente.

setPath(String):asigna la ruta para escribir los archivos de respuesta.

getPath():regresa la ruta para escribir archivos de respuesta.

sendResponse():envía el SMS al cliente con la respuesta del servidor.

Clase SearchInformation

Descripción

La clase SearchInformation es la encargada de realizar la búsqueda de la información solicitada así como de crear un resumen con el resultado de la misma y enviar la respuesta al TR conectado al servidor.

Atributos

word:arreglo de cadenas que son las que contienen los datos de la información a consultar.

pageNum:entero que especifica el número máximo de resultados, para crear la síntesis o resumen de de la búsqueda.

searchRes:cadena con la información de la síntesis creada a partir del resultado de la búsqueda realizada.

searcher:cadena con la información del buscador a utilizar.

log:cadena de que contiene la información de la ejecución de procesos de esta clase.

Métodos

sendSearch(String[], int):es el encargado de ejecutar la búsqueda de la información.

createSintesis(String[]):crea una síntesis a partir del resultado de la búsqueda efectuada.

getSintesis():regresa el contenido del atributo searchRes.

saveLogFile():guarda a la bitácora de nuestro sistema la información de estado de terminación del proceso, esto lo hace delegando esta responsabilidad a la clase Logger.

buildresponse():construye el mensaje de respuesta y se lo pasa a un objeto ResponseSender, para que éste lo procese.

sendResponse():envía el mensaje de respuesta a partir de ResponseSender.

Clase abstracta Mailer

Descripción

Clase abstracta que se encarga de definir la funcionalidad de un cliente de correo, es decir, en esta clase se definen los métodos para enviar y recibir correos electrónicos.

Atributos

server:cadena que contiene la dirección del servidor de correo electrónico.

port:entero que contiene el puerto por el cual el servidor provee el servicio de correo.

user:cadena que contiene el nombre del usuario de correo electrónico

password:cadena que contiene la contraseña del usuario de correo electrónico.

response:objeto que maneja el envío de respuesta hacia la TR conectada al servidor.

Métodos

sendMail(String,String,String,String[]):envía un mensaje de correo electrónico a las direcciones de correo especificadas.

ArrayList receiveRecentMails():obtiene los correos electrónicos recientes del usuario especificado en user.

Clase MailSender

Descripción

Es una clase que implementa a la clase Mailer y añade funcionalidades como la de guardar estado de ejecución a bitácora, construir y enviar mensaje de respuesta.

Métodos

buildResponse():construye el mensaje de respuesta hacia la TR conectada al servidor.

sendResponse():envía el mensaje de respuesta a través de ResponseSender.

saveLogFile():guarda en la bitácora de nuestro sistema la información de estado de terminación de proceso, esto lo hace delegando esta responsabilidad a la clase Logger.

Interfaz Command

Descripción

Esta interfaz define los métodos mínimos que cualquier comando debe tener.

Métodos

execute():envía la orden de ejecución de comando.

login():establece conexión con el servidor en cual se ejecutará el comando.

Clase TelnetExecuter

Descripción

Clase que implementa a la interfaz Command, esta clase ejecuta comandos vía telnet.

Atributos

user:cadena que contiene el nombre del usuario telnet.

password:cadena que contiene la contraseña de usuario telnet.

command:arreglo de cadenas que contienen el comando telnet a ejecutar así como sus parámetros.

Métodos

closeConnection():cierra la conexión con el servidor telnet.

buildResponse():construye el mensaje de respuesta.

getCommand():regresa el arreglo de cadenas que contienen el comando telnet a ejecutar.

setCommand(String[]):actualiza el comando telnet a ejecutar.

saveLogFile():guarda en la bitácora de nuestro sistema la información de estado de terminación de proceso, esto lo hace delegando esta responsabilidad a la clase Logger.

Clase SSHExecuter

Descripción

Clase que implementa a la interfaz Command, esta clase ejecuta comandos vía SSH.

Atributos

user:cadena que contiene el nombre del usuario SSH.

password:cadena que contiene la contraseña de usuario SSH.

command:arreglo de cadenas que contienen el comando SSH a ejecutar así como sus parámetros.

Métodos

closeConnection():cierra la conexión con el servidor SSH.

buildResponse():construye el mensaje de respuesta.

getCommand():regresa el arreglo de cadenas que contienen el comando SSH a ejecutar.

setCommand(String[]):actualiza el comando SSH a ejecutar.

saveLogFile():guarda a la bitácora de nuestro sistema la información de estado de terminación de proceso, esto lo hace delegando esta responsabilidad a la clase Logger.

Clase JavaExecuter

Descripción

Clase que implementa a la interfaz Command, esta clase ejecuta comandos vía java RMI (Remote Method Invocation).

Atributos

user:contiene el nombre del usuario RMI.

password:contiene la contraseña de usuario RMI.

command:arreglo de cadenas que contienen el comando RMI a ejecutar así como sus parámetros.

Métodos

closeConnection():cierra la conexión con el servidor RMI.

buildResponse():construye el mensaje de respuesta.

getCommand():regresa el arreglo de cadenas que contienen el comando RMI a ejecutar.

setCommand(String[]):actualiza el comando RMI a ejecutar.

saveLogFile():guarda en la bitácora de nuestro sistema la información de estado de terminación del proceso, esto lo hace delegando esta responsabilidad a la clase Logger.

Diagramas UML

UML (*Unified Modeling Language*) es una especificación de OMG (*Object Management Group*), que sirve para especificar, visualizar y documentar modelos de sistemas de software, incluyendo su estructura y diseño; aunque pueden modelarse también otro tipo de sistemas.

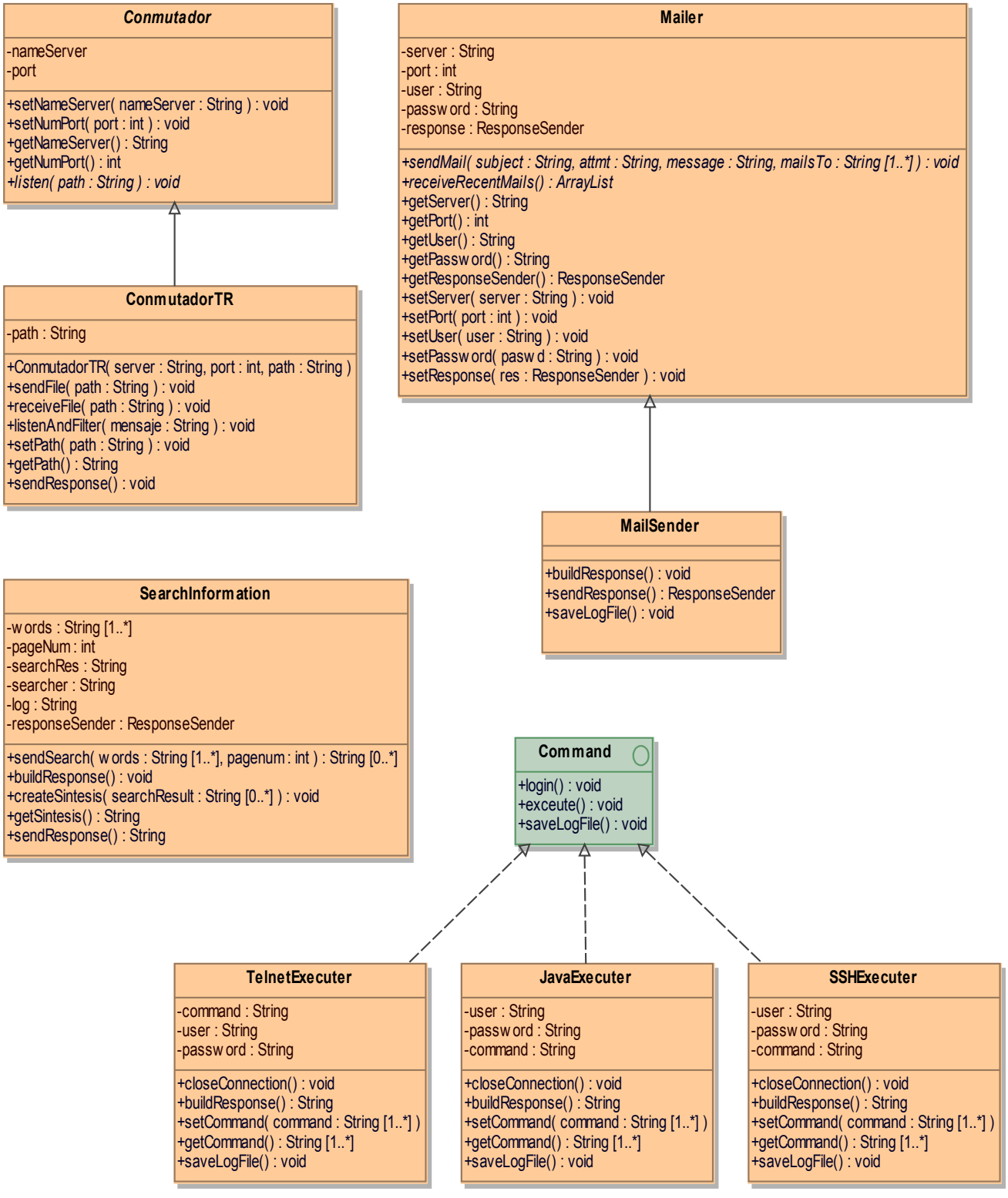
Los diagramas que define el UML se pueden dividir en tres categorías:

- Diagramas de estructura. Se incluyen diagramas de clase, diagramas de objetos, diagramas de componentes, diagramas de composición de estructura, diagramas de paquetes y diagramas de desplegado.
- Diagramas de comportamiento. Se incluyen diagramas de casos de uso, diagramas de actividad y diagramas de estados.
- Diagramas de interacción. Derivados de diagramas de comportamiento, aquí se encuentran los diagramas de secuencia.

Para el caso del proyecto se utilizaron los diagramas de clase y los diagramas de secuencia.

Diagramas de clase

En esta sección se muestran los diagramas de clase, descritas en la sección anterior.


Conmutador
-nameServer
-port
+setNameServer(nameServer : String) : void
+setNumPort(port : int) : void
+getNameServer() : String
+getNumPort() : int
+listen(path : String) : void
ConmutadorTR
-path : String
+ConmutadorTR(server : String, port : int, path : String)
+sendFile(path : String) : void
+receiveFile(path : String) : void
+listenAndFilter(mensaje : String) : void
+setPath(path : String) : void
+getPath() : String
+sendResponse() : void
Mailer
-server : String
-port : int
-user : String
-password : String
-response : ResponseSender
+sendMail(subject : String, attmt : String, message : String, mailsTo : String [1..*]) : void
+receiveRecentMails() : ArrayList
+getServer() : String
+getPort() : int
+getUser() : String
+getPassword() : String
+getResponseSender() : ResponseSender
+setServer(server : String) : void
+setPort(port : int) : void
+setUser(user : String) : void
+setPassword(passwd : String) : void
+setResponse(res : ResponseSender) : void
MailSender
+buildResponse() : void
+sendResponse() : ResponseSender
+saveLogFile() : void
SearchInformation
-words : String [1..*]
-pageNum : int
-searchRes : String
-searcher : String
-log : String
-responseSender : ResponseSender
+sendSearch(words : String [1..*], pagenum : int) : String [0..*]
+buildResponse() : void
+createSintesis(searchResult : String [0..*]) : void
+getSintesis() : String
+sendResponse() : String
Command
+login() : void
+exceute() : void
+saveLogFile() : void
TelnetExecuter
-command : String
-user : String
-password : String
+closeConnection() : void
+buildResponse() : String
+setCommand(command : String [1..*])
+getCommand() : String [1..*]
+saveLogFile() : void
JavaExecuter
-user : String
-password : String
-command : String
+closeConnection() : void
+buildResponse() : String
+setCommand(command : String [1..*])
+getCommand() : String [1..*]
+saveLogFile() : void
SSHExecuter
-user : String
-password : String
-command : String
+closeConnection() : void
+buildResponse() : String
+setCommand(command : String [1..*])
+getCommand() : String [1..*]
+saveLogFile() : void

Diagramas de secuencia

En esta sección se muestra el flujo del sistema de transporte de información asociado con el proyecto. Para ello se describen diferentes escenarios asociados cada uno de ellos con un servicio que el sistema debería proporcionar.

La descripción del proceso que involucra cada escenario se hace mediante diagramas de secuencia.

La petición de servicios al servidor se hace mediante el protocolo de comunicación descrito en la sección 4.1.1, por lo que es requisito leer dicho protocolo antes de proseguir con esta lectura.

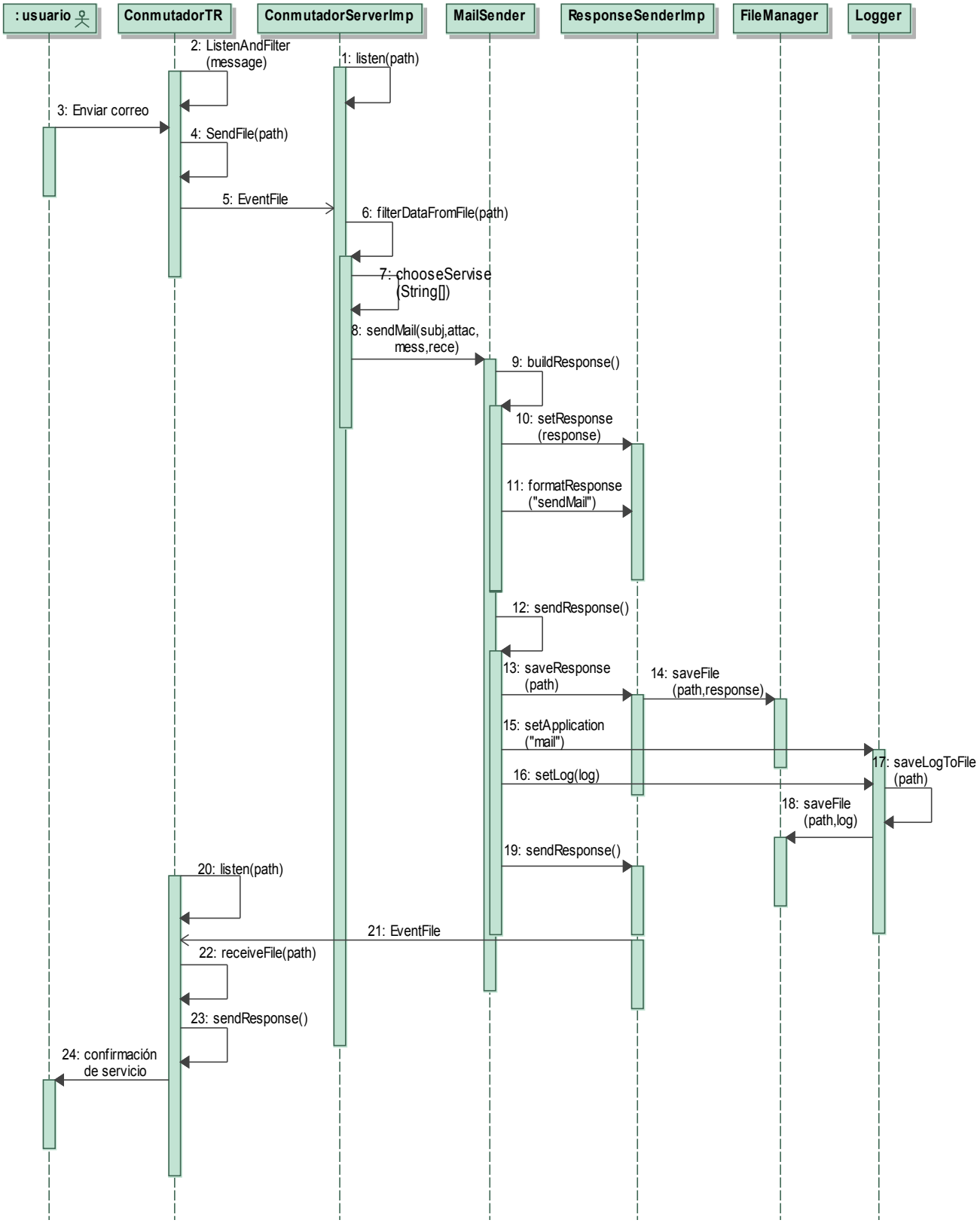
Escenario: Envío de correo

A partir de la instrucción de envío de correo electrónico por parte del usuario, el sistema debe realizar llamadas a métodos de distintos objetos los cuales tienen una función específica qué realizar, el flujo se describe a continuación.

1. El usuario envía la instrucción de enviar correo. El envío sería de la siguiente forma:
    a. MaF huilver@gmail.com
    b. Pass micontraseña
    c. MaT avallelcc@gmail.com
    d. Subj prueba de correo
    e. Mess El mensaje que será enviado por correo
    f. EMess
2. La clase ConmutadorTR, que reside en la TR conectada al servidor por USB o tecnología *Bluetooth*, está en espera (en ciclo infinito) de mensajes SMS con el formato congruente con el protocolo de comunicación establecido. ConmutadorTR realiza lo anterior mediante su método ConmutadorTR.ListenAndFilter(message).
3. Se ejecuta ConmutadorTR.saveFile(message, path) el cual delega la responsabilidad de guardar el mensaje recibido a un archivo a FileManager.saveFile(path, message). Por comodidad llamaremos a este archivo arch.in.
4. Se ejecuta el método ConmutadorTR.sendFile(path), el cual permite enviar a arch.in.
5. La clase ConmutadorServerImp, que reside en nuestro servidor, está en espera (en ciclo infinito) de eventos generados por ConmutadorTR, el evento que espera es la recepción del archivo arch.in. Lo anterior se realiza mediante la ejecución del método ConmutadorServerImp.Listen(path).
6. Se ejecuta ConmutadorServerImp.FilterDataFromFile(path), el cual se encarga de extraer y filtrar los datos del archivo arch.in.
7. Se ejecuta ConmutadorServerImp.chooseServise(String[]), el cual se encarga de elegir el servicio solicitado.

8. Se ejecuta MailSender.sendMail(subj,attch,mess,mailsTo), el cual tiene como propósito enviar el correo especificado.
9. Se ejecuta MailSender.buildResponse(), el cual tiene como propósito construir el archivo de respuesta del servicio. En este caso solo contendrá la notificación de ejecución del proceso, MailSender.buildResponse() delega esta responsabilidad a otros métodos de otras clases.
10. Cuando MailSender.buildResponse() es ejecutado, éste ejecuta a ResponseSenderImp.setResponse(response) el cual coloca la respuesta para su posterior formateo, mediante ResponseSenderImp.formatResponse(tipoServicio).
11. Se ejecuta sendResponse() cuyo objetivo es enviar la respuesta a la TR conectada al servidor. Para lograr el envío de esta respuesta es necesario primero guardar la respuesta en un archivo mediante ResponseSenderImp.saveResponse(path), a este archivo le llamaremos arch.out. Antes de que sea efectuado el envío del archivo arch.out a la TR conectada al servidor por ResponseSenderImp.sendResponse(), se debe registrar en la bitácora el evento mediante setAplication("mail") y setLog(log).
12. El usuario recibe una notificación del proceso.

Lo explicado arriba se ilustra en el siguiente diagrama de secuencia.


: usuario
ConmutadorTR
ConmutadorServerImp
MailSender
ResponseSenderImp
FileManager
Logger
2: ListenAndFilter (message)
1: listen(path)
3: Enviar correo
4: SendFile(path)
5: EventFile
6: filterDataFromFile(path)
7: chooseServise (String[])
8: sendMail(subj,attac, mess,rece)
9: buildResponse()
10: setResponse (response)
11: formatResponse ("sendMail")
12: sendResponse()
13: saveResponse (path)
14: saveFile (path,response)
15: setApplication ("mail")
16: setLog(log)
17: saveLogToFile (path)
18: saveFile (path,log)
19: sendResponse()
20: listen(path)
21: EventFile
22: receiveFile(path)
23: sendResponse()
24: confirmación de servicio

Escenario: Ejecución de comando remoto.

En este escenario, el sistema debe permitir la ejecución de comandos remotos en un servidor indicado por el usuario.

La sintaxis que el protocolo de comunicación establece para este servicio sería de la siguiente forma:

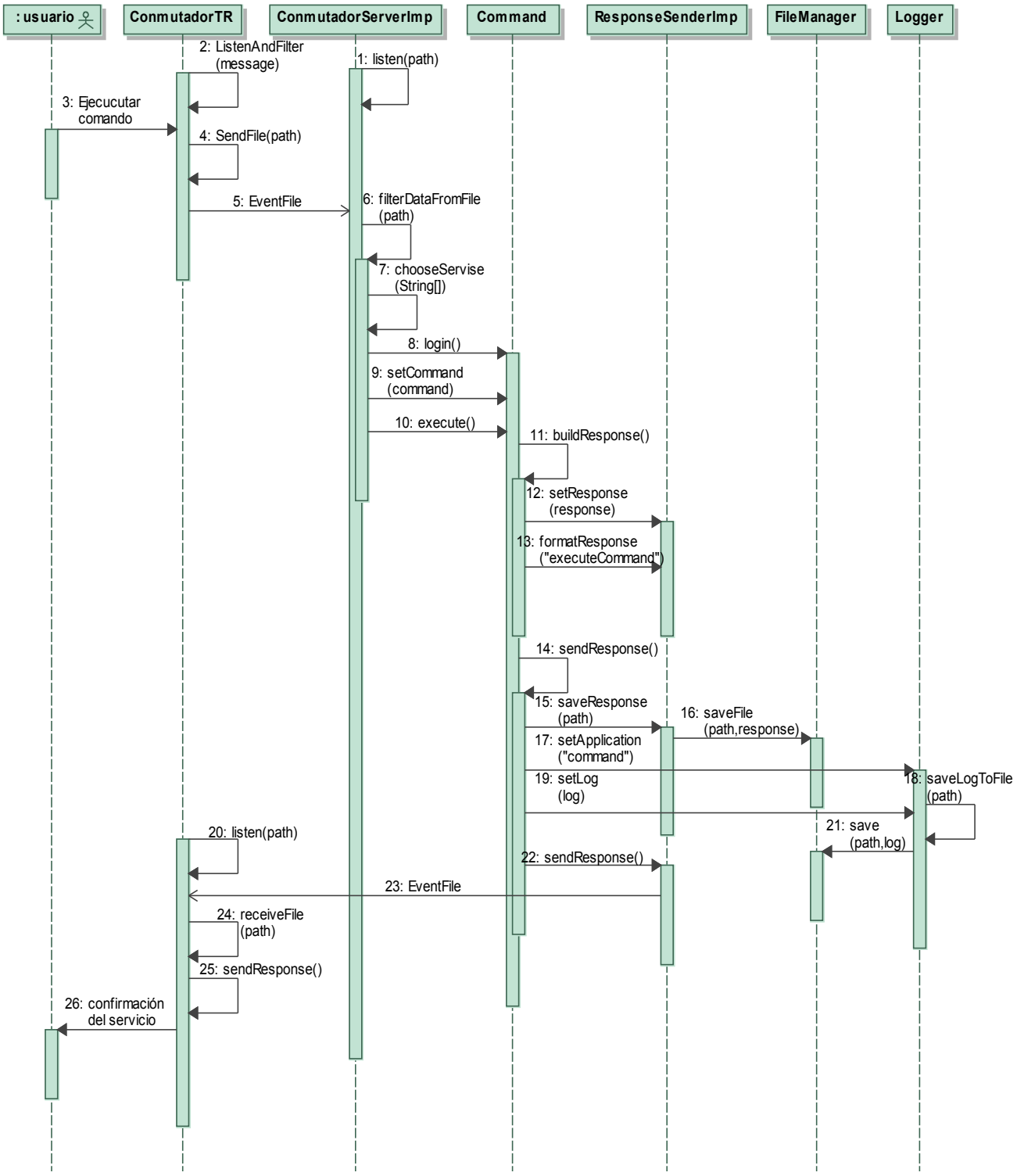
1. El usuario envía la instrucción de ejecutar comando. El envío sería de la siguiente forma:
   a. Rcmd playS miCancion.mp3
   b. Server 192.68.0.1
   c. Port 1024
   d. User huilver
   e. Pass micontraseña

Los pasos 2, 3, 4, 5, 6 y 7 son los mismos pasos que en el escenario, de envío de correo, los pasos 9,10 y 11 son análogos al escenario de envió de email salvo que el objeto que ejecuta las llamadas correspondientes a MailSender los ejecuta Command; así que solo faltan por describir los pasos 8 y 12.

8. Se ejecuta Command.login(), el cual permite al usuario autenticarse con el servidor en el que se ejecutará el comando; después se ejecuta Command.setCommand(commad) el cual fija el comando a ejecutar; y por último se ejecuta Command.execute() cuya función es ejecutar el comando remoto especificado.

12. El usuario recibe notificación de la ejecución del comando remoto ejecutado.

El diagrama de secuencia de este escenario se muestra a continuación.


: usuario
ConmutadorTR
ConmutadorServerImp
Command
ResponseSenderImp
FileManager
Logger
1: listen(path)
2: ListenAndFilter (message)
3: Ejecucutar comando
4: SendFile(path)
5: EventFile
6: filterDataFromFile (path)
7: chooseServise (String[])
8: login()
9: setCommand (command)
10: execute()
11: buildResponse()
12: setResponse (response)
13: formatResponse ("executeCommand")
14: sendResponse()
15: saveResponse (path)
16: saveFile (path,response)
17: setApplication ("command")
18: saveLogToFile (path)
19: setLog (log)
20: listen(path)
21: save (path,log)
22: sendResponse()
23: EventFile
24: receiveFile (path)
25: sendResponse()
26: confirmación del servicio

Escenario: Búsqueda de información en la red.

En este escenario, el sistema debe permitir la búsqueda de información en los servidores conectados en la red a la que nuestro servidor tenga acceso.

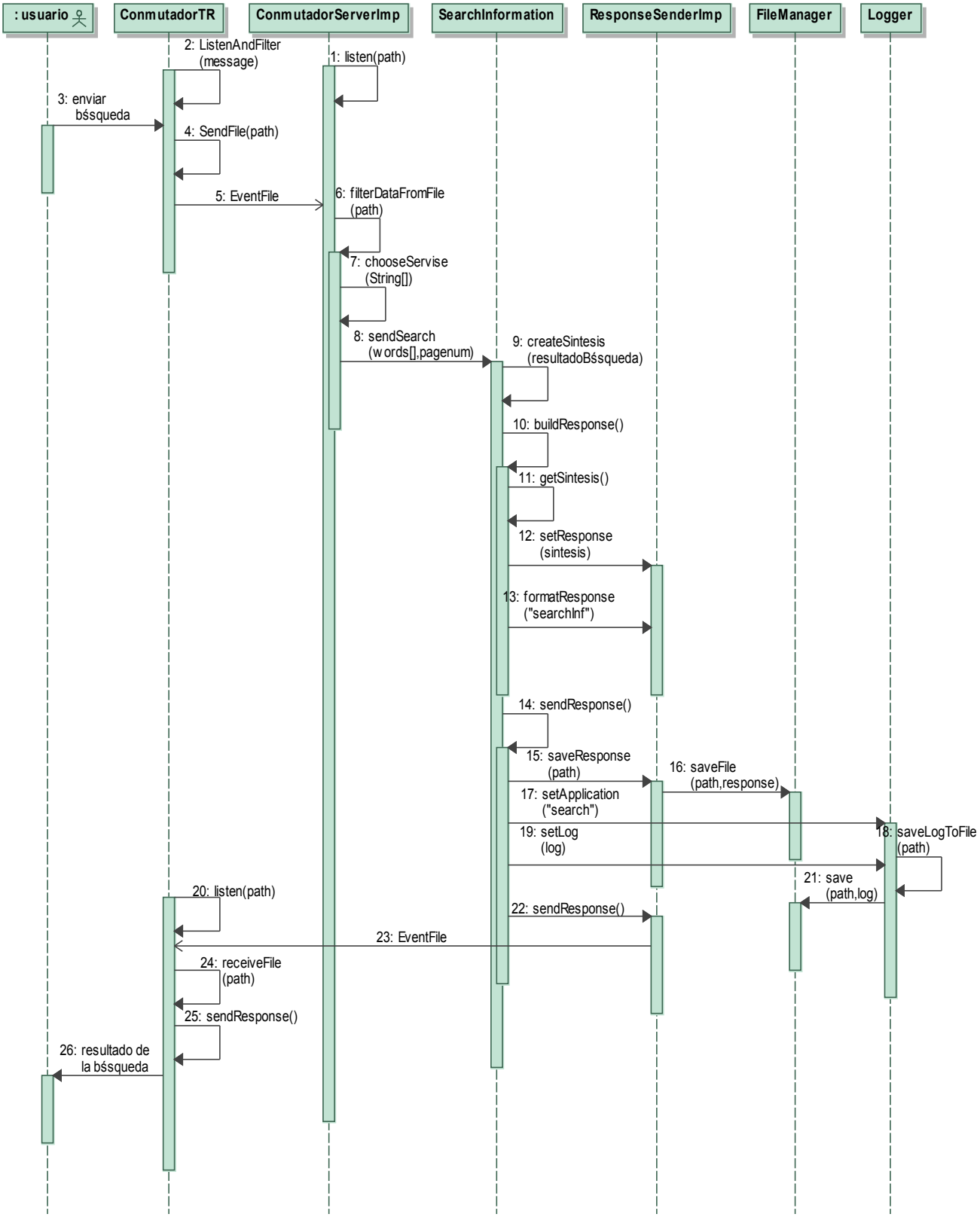
La sintaxis que el protocolo de comunicación establece para este servicio seria de la siguiente forma:

1. El usuario envía la instrucción buscar información. El envío sería de la siguiente forma:
   a. Srch computación
   b. NMsr 3

Los pasos 2, 3, 4, 5 ,6 y 7 son los mismos pasos que en el escenario de envío de correo, los pasos 9,10 y 11 son análogos al escenario de envío de correo salvo que el objeto que ejecuta las llamadas correspondientes a MailSender los ejecuta SearchInformation; así que solo faltan por describir los pasos 8 y 12.

8. Se ejecuta SearchInformation.sendSearch(words[], pagenum) cuyo propósito es buscar archivos (html,pdf,doc,etc) que contengan información concerniente a la palabra solicitada. Se crea un resumen de la información obtenida mediante la llamada al método SearchInformation.createSintesis(resultadoBusqueda).El resumen será usado posteriormente para el envío de la respuesta del sistema.

12. El diagrama de secuencia para Búsqueda de información en la red es el siguiente.


: usuario
ConmutadorTR
ConmutadorServerImp
SearchInformation
ResponseSenderImp
FileManager
Logger
1: listen(path)
2: ListenAndFilter (message)
3: enviar bśsqueda
4: SendFile(path)
5: EventFile
6: filterDataFromFile (path)
7: chooseServise (String[])
8: sendSearch (words[],pagenum)
9: createSintesis (resultadoBśsqueda)
10: buildResponse()
11: getSintesis()
12: setResponse (sintesis)
13: formatResponse ("searchInf")
14: sendResponse()
15: saveResponse (path)
16: saveFile (path,response)
17: setApplication ("search")
18: saveLogToFile (path)
19: setLog (log)
20: listen(path)
21: save (path,log)
22: sendResponse()
23: EventFile
24: receiveFile (path)
25: sendResponse()
26: resultado de la bśsqueda

Implementación de la arquitectura para transacciones básicas (proyecto PAPIIT)

A continuación se describe la implementación del diseño propuesto por el Dr. Benjamín Macías Pimentel, que esta descrito en los anexos A y B. De éste diseño se implementaron todas clases necesarias para el envío de información desde una terminal remota hacia un servidor.

Implementación de la arquitectura básica para el envío de información desde una terminal remota hacia un servidor

La implementación del sistema que permite transferir información entre el servidor y las terminales remotas a nivel básico, consta de tres "subproyectos" cada uno ubicado en una entidad del sistema.

Los "subproyectos" desarrollados son:

SRII-TRC llamado así porque reside en la terminal remota cliente (TRC), que es la que se encarga de interactuar con el usuario mediante una interfaz gráfica.

SRII-TRS llamado así porque reside en la terminal remota servidor (TRS), que se encarga de interactuar con SRII-TRC mediante la red telefónica y con SRII-SRV mediante *Bluetooth*.

SRII-SRV llamada así porque reside en el servidor (SRV). Se encarga de recibir las peticiones realizadas por SRII-TRS y de proveer el mecanismo necesario para responder a TRC usando como puente a TRS.

SRII-TRC

Este subproyecto se encarga de interactuar con el usuario mediante una interfaz gráfica, ésta se encuentra implementada en Java J2ME.

Para desarrollar aplicaciones en J2ME se deben tener en cuenta varias consideraciones que dependen totalmente de la gama de dispositivos a la cual va dirigida la aplicación.

Las consideraciones que deben tomarse son: la configuración, el perfil y los paquetes opcionales que se van a utilizar en la aplicación. Para el desarrollo de SRII-TRC se utilizó la configuración CLCD-1.0 (*Connected Limited Configuration Device 1.0*), el perfil MIDP-2.0 (*Mobile Information Device Profile 2.0*) y como paquetes opcionales los APIs de mensajería (*Wireless Messaging API 2.0*) que es la especificación JSR-120 y los APIs de *Bluetooth* (*Java APIs for Bluetooth*) especificados en JSR-82.

Las clases implementadas para SRII-TRC son EnviadorSMS y Ventana ambas empaquetadas en el paquete transporte.

Componente de Transporte

Componente que se encarga de interactuar con el usuario y de mandar la información desde TRC hacia TRS.

Paquete transporte

Este paquete agrupa las clases necesarias para conformar el componente de transporte. Las clases incluídas en este paquete son: EnviadorSMS y la clase Ventana.

EnviadorSMS

La función de esta clase es enviar mensajes SMS a otra terminal remota, para lograr su cometido utiliza las APIs de mensajería (JSR-120) las cuales deben ser soportadas por el dispositivo móvil

La manera en como EnviadorSMS opera puede describirse en tres pasos.

1. Abrir una conexión con la dirección del dispositivo meta.
2. crear el mensaje.
3. enviar el mensaje creado mediante la conexión que se tiene.

la dirección debe tener el siguiente formato: "sms://"+númeroDeTelefono:puerto" por ejemplo "sms://+5532118304:6222".

Implementación:

```
public class EnviadorSMS implements Runnable
{

  /**Puerto del teléfono al cual se enviará  el mensaje*/
      private  String port;
      /**Mensaje que será enviado*/
 private String message;
 /**Número de teléfono al cual se enviará el mensaje*/
      private String phoneNumber;

      public void run()
      {
             String address = "sms://" + phoneNumber+ port;
             MessageConnection smsconn = null;
             try
             {
                    smsconn = (MessageConnection)Connector.open(address);

                   TextMessage txtmessage =
                   (TextMessage)smsconn.newMessage(MessageConnection.TEXT_MESSAGE);

                    txtmessage.setPayloadText(message);
                    smsconn.send(txtmessage);
             }
             catch (Exception e)
             {
                    e.printStackTrace();
             }

             if (smsconn != null) {
                    try
                    {
                           smsconn.close();
```

```
                    }
                    catch (IOException ioe)
                    {
                           ioe.printStackTrace();
                    }
             }
       }


        /**Método que abre la conexión del servicio y envía el mensaje dado
         *@param message - mensaje que será enviado
         *@param phoneNumber - número del destinatario
         *@param port - puerto del teléfono en que será enviado el mensaje,
           si null se  envía sin un puerto específico
         */
             public synchronized void  connectAndSend(String message,
                                                      String phoneNumber,String  port)
             {
             this.message = message;
             this.phoneNumber = phoneNumber;
                this.port=port==null?"":":"+port;
             Thread t = new Thread(this);
             t.start();
             }

}
```

Ventana

Esta clase es una interfaz gráfica que permite al usuario seleccionar los servicios proporcionados por el servidor.

Esta clase fue modelada por el editor de interfaces gráficas para J2ME proporcionada por Netbeans y el código generado por esta herramienta se adapto a nuestros fines.

Esta clase invoca a EnviadorSMS para el envío de las peticiones del usuario.

Implementación:

```
package transporte;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author Huilver Nolasco Aguilar
 */
public class Ventana extends MIDlet implements CommandListener {


  public Ventana() {
        initialize();
    }

    private List servicios;
    private Command elegir;
    private Form ecorreo;
    private Command ecorreoregresar;
    private Form ocorreo;
    private Command ecorreoRegresar;
    private Form ecomando;
    private Form busqueda;
    private Command ecomandoregresar;
    private Command busquedaregresar;
```

```
private TextField ecorreoRem;
private TextField ecorreoPassword;
private TextField ecorreoDest;
private TextField ocorreoDir;
private TextField ocorreoPassword;
private TextField cmmdServidor;
private TextField cmmdPassword;
private TextField cmmdComando;
private TextField busquedaPalabra;
private Command salirBusqueda;
private Command salirEcomando;
private Command salirOcorreo;
private Command salirEcorreo;
private TextBox mensaje;
private Command okEcorreo;
private Command okCommand2;
private Command mensajeRegresar;
private Command exitCommand1;
private Command itemCommand1;
private Command okMensaje;
private Command okOcorreo;
private Command okEcomando;
private Command okBusqueda;
private Alert alerta;
private Command okCommand1;

private EnviadorSMS esms;
private final String    TELEFONOSERVIDOR="5536684795";
private final String    SMS_PORT="6222";
private final int       ENVIOCORREO=1;
private final int       BUSQUEDA=2;
private final int       OBTENERCORREO=3;
private final int       EJECUTARCOMANDO=4;



public void commandAction(Command command, Displayable displayable) {
    if (displayable == servicios) {
        if (command == servicios.SELECT_COMMAND) {
            switch (get_servicios().getSelectedIndex()) {
                case 0:
                    getDisplay().setCurrent(get_ecomando());
                    break;
                case 1:
                    getDisplay().setCurrent(get_busqueda());
                    break;
                case 2:
                    getDisplay().setCurrent(get_ocorreo());
                    break;
                case 3:
                    getDisplay().setCurrent(get_ecorreo());
                    break;
            }
        } else if (command == elegir) {
            switch (get_servicios().getSelectedIndex()) {
                case 0:
                    getDisplay().setCurrent(get_ecomando());
                    break;
                case 1:
                    getDisplay().setCurrent(get_busqueda());
                    break;
                case 2:
                    getDisplay().setCurrent(get_ocorreo());
                    break;
                case 3:
                    getDisplay().setCurrent(get_ecorreo());
                    break;
            }



        } else if (command == itemCommand1) {
            exitMIDlet();
```

```
    }
} else if (displayable == ecorreo) {
    if (command == ecorreoregresar) {
        getDisplay().setCurrent(get_servicios());
    } else if (command == okEcorreo) {
        getDisplay().setCurrent(get_mensaje());
    }
} else if (displayable == ecomando) {
    if (command == ecomandoregresar) {
        getDisplay().setCurrent(get_servicios());
    } else if (command == okEcomando) {

    //Obtencion de datos de ejecutar comando
        String servidor=this.get_cmmdServidor().getString();
        String passwd=this.get_cmmdPassword().getString();
        String cmmd=this.get_cmmdComando().getString();
        String info=servidor+"|"+passwd+"|"+cmmd;

        if((servidor!=null&&servidor.length()>0)&&
           (passwd!=null&&passwd.length()>0)&&
           (cmmd!=null&&cmmd.length()>0)){
        esms.connectAndSend(this.EJECUTARCOMANDO+"|"+info,TELEFONOSERVIDOR,SMS_PORT);
        }
        else
            getDisplay().setCurrent(get_alerta(), get_ecomando());

    }
} else if (displayable == busqueda) {
    if (command == busquedaregresar) {
        getDisplay().setCurrent(get_servicios());
    } else if (command == okBusqueda) {
        //Obtenemos el texto de la palabra a buscar
        String palabraBuscar=this.get_busquedaPalabra().getString();

        if (palabraBuscar!=null && palabraBuscar.length()>0)
        esms.connectAndSend(this.BUSQUEDA+"|"+palabraBuscar,TELEFONOSERVIDOR,SMS_PORT);
        else
            getDisplay().setCurrent(get_alerta(), get_busqueda());
    }
} else if (displayable == ocorreo) {
    if (command == ecorreoRegresar) {
        getDisplay().setCurrent(get_servicios());
    } else if (command == okOcorreo) {
        //Obtencion de datos para obtencion de correo
        String correo=this.get_ocorreoDir().getString();
        String passwd=this.get_ocorreoPassword().getString();
        String info=correo+"|"+passwd;

        if((correo!=null&&correo.length()>0)&&
           (passwd!=null&&passwd.length()>0))
        esms.connectAndSend(this.OBTENERCORREO+"|"+info,TELEFONOSERVIDOR,SMS_PORT);
        else
            getDisplay().setCurrent(get_alerta(), get_ocorreo());

    }
} else if (displayable == mensaje) {
    if (command == mensajeRegresar) {
        getDisplay().setCurrent(get_ecorreo());
    } else if (command == okMensaje) {
        /* Se obtienen los datos para el envió de correo*/
        String rem= this.get_ecorreoRem().getString();
        String passwd=  this.get_ecorreoPassword().getString();
        String dest=this.get_ecorreoDest().getString();
        String mensaje=this.get_mensaje().getString();
        String info=rem+"|"+passwd+"|"+dest+"|"+mensaje;

        if((rem!=null&&rem.length()>0)&&
                (passwd!=null&&passwd.length()>0)&&
                (dest!=null&&dest.length()>0)&&
                (mensaje!=null&&mensaje.length()>0))
        esms.connectAndSend(this.ENVIOCORREO+"|"+info,TELEFONOSERVIDOR,SMS_PORT);
        else
            getDisplay().setCurrent(get_alerta(), get_mensaje());
    }
```

```
        }
}

    /** This method initializes UI of the application.//GEN-BEGIN:MVDInitBegin
     */
    private void initialize() {
        getDisplay().setCurrent(get_servicios());
    }

    /**
     * This method should return an instance of the display.
     */
    public Display getDisplay() {
        return Display.getDisplay(this);
    }

    /**
     * This method should exit the midlet.
     */
    public void exitMIDlet() {
        getDisplay().setCurrent(null);
        destroyApp(true);
        notifyDestroyed();
    }


    public List get_servicios() {
        if (servicios == null) {
            servicios = new List("Servicios", Choice.EXCLUSIVE, new String[] {
                "Ejecutar Comando",
                "Buscar",
                "Obtener Correos",
                "Enviar Correo"
            }, new Image[] {
                null,
                null,
                null,
                null
            });
            servicios.addCommand(get_elegir());
            servicios.addCommand(get_itemCommand1());
            servicios.setCommandListener(this);
            servicios.setSelectedFlags(new boolean[] {
                true,
                false,
                false,
                false
            });


        }
        return servicios;
    }

    public Command get_elegir() {
        if (elegir == null) {
            elegir = new Command("elegir", Command.OK, 1);
        }
        return elegir;
    }

    public Form get_ecorreo() {
        if (ecorreo == null) {
            ecorreo = new Form("Envio de correo", new Item[] {
                get_ecorreoRem(),
                get_ecorreoPassword(),
                get_ecorreoDest()
            });
            ecorreo.addCommand(get_ecorreoregresar());
            ecorreo.addCommand(get_okEcorreo());
            ecorreo.setCommandListener(this);
        }
        return ecorreo;
    }
```

```
public Command get_ecorreoregresar() {
    if (ecorreoregresar == null) {
        ecorreoregresar = new Command("Regresar", Command.BACK, 1);
    }
    return ecorreoregresar;
}



public Form get_ocorreo() {
    if (ocorreo == null) {
        ocorreo = new Form("Obtener Correo", new Item[] {
            get_ocorreoDir(),
            get_ocorreoPassword()
        });
        ocorreo.addCommand(get_ecorreoRegresar());
        ocorreo.addCommand(get_okOcorreo());
        ocorreo.setCommandListener(this);
    }
    return ocorreo;
}

public Command get_ecorreoRegresar() {
    if (ecorreoRegresar == null) {
        ecorreoRegresar = new Command("Regresar", Command.BACK, 1);
    }
    return ecorreoRegresar;
}

public Form get_ecomando() {
    if (ecomando == null) {
        ecomando = new Form("Ejecutar comando", new Item[] {
            get_cmmdServidor(),
            get_cmmdPassword(),
            get_cmmdComando()
        });
        ecomando.addCommand(get_ecomandoregresar());
        ecomando.addCommand(get_okEcomando());
        ecomando.setCommandListener(this);
    }
    return ecomando;
}


public Form get_busqueda() {
    if (busqueda == null) {
        busqueda = new Form("Busqueda", new Item[] {get_busquedaPalabra()});
        busqueda.addCommand(get_busquedaregresar());
        busqueda.addCommand(get_okBusqueda());
        busqueda.setCommandListener(this);
    }
    return busqueda;
}


public Command get_ecomandoregresar() {
    if (ecomandoregresar == null) {
        ecomandoregresar = new Command("Regresar", Command.BACK, 1);
    }
    return ecomandoregresar;
}


public Command get_busquedaregresar() {
    if (busquedaregresar == null) {
        busquedaregresar = new Command("Regresar", Command.BACK, 1);
    }
    return busquedaregresar;
}


public TextField get_ecorreoRem() {
    if (ecorreoRem == null) {
```

```
                    ecorreoRem = new TextField("Remitente", null, 25, TextField.EMAILADDR);
                                    ecorreoRem.setLayout(Item.LAYOUT_RIGHT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER);
                }
                return ecorreoRem;
            }


            public TextField get_ecorreoPassword() {
                if (ecorreoPassword == null) {
                        ecorreoPassword = new TextField("Contrase\u00F1a", "", 40, TextField.ANY |
TextField.PASSWORD);
                                    ecorreoPassword.setLayout(Item.LAYOUT_LEFT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER);
                }
                return ecorreoPassword;
            }


            public TextField get_ecorreoDest() {
                if (ecorreoDest == null) {
                    ecorreoDest = new TextField("Destinatario", null, 25, TextField.EMAILADDR);
                                    ecorreoDest.setLayout(Item.LAYOUT_LEFT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER);
                }
                return ecorreoDest;
            }

            public TextField get_ocorreoDir() {
                if (ocorreoDir == null) {
                    ocorreoDir = new TextField("Correo", null, 120, TextField.EMAILADDR);
                                     ocorreoDir.setLayout(Item.LAYOUT_LEFT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER);
                }
                return ocorreoDir;
            }

            public TextField get_ocorreoPassword() {
                if (ocorreoPassword == null) {
                        ocorreoPassword = new TextField("Contrase\u00F1a", "", 10, TextField.ANY |
TextField.PASSWORD);
                                    ocorreoPassword.setLayout(Item.LAYOUT_LEFT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER);
                }
                return ocorreoPassword;
            }


            public TextField get_cmmdServidor() {
                if (cmmdServidor == null) {
                    cmmdServidor = new TextField("Servidor", "", 40, TextField.ANY);
                                    cmmdServidor.setLayout(Item.LAYOUT_LEFT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER);
                }
                return cmmdServidor;
            }


            public TextField get_cmmdPassword() {
                if (cmmdPassword == null) {
                        cmmdPassword = new TextField("Contrase\u00F1a", null, 10, TextField.ANY |
TextField.PASSWORD);
                                    cmmdPassword.setLayout(Item.LAYOUT_LEFT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER);
                }
                return cmmdPassword;
            }


            public TextField get_cmmdComando() {
                if (cmmdComando == null) {
                    cmmdComando = new TextField("Comando", "", 120, TextField.ANY);
                                     cmmdComando.setLayout(Item.LAYOUT_LEFT   |   Item.LAYOUT_TOP   |
Item.LAYOUT_NEWLINE_AFTER | Item.LAYOUT_EXPAND);
```

```
            }
            return cmmdComando;
        }


        public TextField get_busquedaPalabra() {
            if (busquedaPalabra == null) {
                busquedaPalabra = new TextField("Palabra a buscar", null, 120, TextField.ANY);
                                busquedaPalabra.setLayout(Item.LAYOUT_LEFT  |  Item.LAYOUT_TOP  |
Item.LAYOUT_NEWLINE_AFTER);
            }
            return busquedaPalabra;
        }


        public Command get_salirBusqueda() {
            if (salirBusqueda == null) {
                salirBusqueda = new Command("Salir", Command.EXIT, 1);
            }
            return salirBusqueda;
        }


        public Command get_salirEcomando() {
            if (salirEcomando == null) {
                salirEcomando = new Command("Salir", Command.EXIT, 1);
            }
            return salirEcomando;
        }


        public Command get_salirOcorreo() {
            if (salirOcorreo == null) {
                salirOcorreo = new Command("Salir", Command.EXIT, 1);
            }
            return salirOcorreo;
        }


        public Command get_salirEcorreo() {
            if (salirEcorreo == null) {
                salirEcorreo = new Command("Salir", Command.EXIT, 1);
            }
            return salirEcorreo;
        }


        public TextBox get_mensaje() {
            if (mensaje == null) {
                mensaje = new TextBox("Mensaje", "", 120, TextField.ANY);
                mensaje.addCommand(get_mensajeRegresar());
                mensaje.addCommand(get_okMensaje());
                mensaje.setCommandListener(this);
            }
            return mensaje;
        }


        public Command get_okEcorreo() {
            if (okEcorreo == null) {
                okEcorreo = new Command("Ok", Command.OK, 1);
            }
            return okEcorreo;
        }


        public Command get_okCommand2() {
            if (okCommand2 == null) {
                okCommand2 = new Command("Ok", Command.OK, 1);
            }
            return okCommand2;
        }
```

```
    public Command get_mensajeRegresar() {
        if (mensajeRegresar == null) {
            mensajeRegresar = new Command("Regresar", Command.BACK, 1);
        }
        return mensajeRegresar;
    }


    public Command get_exitCommand1() {
        if (exitCommand1 == null) {
            exitCommand1 = new Command("Exit", Command.EXIT, 1);
        }
        return exitCommand1;
    }


    public Command get_itemCommand1() {
        if (itemCommand1 == null) {
            itemCommand1 = new Command("Salir", Command.EXIT, 1);
        }
        return itemCommand1;
    }


    public Command get_okMensaje() {
        if (okMensaje == null) {
            okMensaje = new Command("Ok", Command.OK, 1);
        }
        return okMensaje;
    }


    public Command get_okOcorreo() {
        if (okOcorreo == null) {
            okOcorreo = new Command("Ok", Command.OK, 1);
        }
        return okOcorreo;
    }


    public Command get_okEcomando() {
        if (okEcomando == null) {
            okEcomando = new Command("Ok", Command.OK, 1);
        }
        return okEcomando;
    }


    public Command get_okBusqueda() {
        if (okBusqueda == null) {
            okBusqueda = new Command("Ok", Command.OK, 1);
        }
        return okBusqueda;
    }


    public Alert get_alerta() {
        if (alerta == null) {
             alerta = new Alert(null, "Toda la informaci\u00F3n es requerida. Llene todos los
campos del formulario por favor", null, AlertType.INFO);
            alerta.setTimeout(-2);
        }
        return alerta;
    }


    public Command get_okCommand1() {
        if (okCommand1 == null) {
            okCommand1 = new Command("Ok", Command.OK, 1);
        }
        return okCommand1;
    }

    public void startApp() {
```

```
            esms=new EnviadorSMS();
        }

        public void pauseApp() {
        }

        public void destroyApp(boolean unconditional) {
        }

    }
```

SRII-TRS

La función original era la de comunicar a TRC con nuestro servidor en las dos direcciones envío y recepción, sin embargo, debido al esquema de seguridad que Java J2ME especifica en JSR-120, se requiere una confirmación forzosa del usuario para el envío de mensajes SMS ya que éste servicio genera costos. Por lo tanto, el envío de respuestas del servidor hacia TRC no pudo realizarse de manera automática solo usando Java J2ME, razón por la cual algunas clases implementadas para SRII-TRS quedaron sin uso o con uso parcial.

Las clases pertenecientes a SRII-TRS son: EnviadorSMS, ObexServer, Receptor y RecibidorSMS. Todas estas clases están bajo el paquete de transporte.

Componente de Transporte

Este componente recibe la información de TRC y la envía a SRV, es decir se encarga del flujo de información de envío desde TRC hacia SRV.

Paquete transporte

Contiene las clases necesarias que conforman al componente de transporte, las clases que este paquete incluye son: ObexServer, EnviadorSMS y Receptor.

ObexServer

Esta clase implementa un servidor Obex y además nos permite enviar información hacia SRV mediante *Bluetooth*.

El objetivo de tener un servidor Obex era recibir información de SRV vía *Bluetooth* mediante el protocolo Obex, esta funcionalidad quedó sin uso como consecuencia de la limitación de J2ME descrita en 5.1.2; quedando como única funcionalidad la de envío de información hacia SRII-SRV mediante *Bluetooth*.

La forma en la cual ésta clase envía información al servidor es la siguiente:

1. Se crea una sesión con la dirección *Bluetooth* del servidor. Un ejemplo de dirección es

btgoep://0009DD6000A7:3;authenticate=false;encrypt=false;master=false

2. Se conecta a la dirección
3. se crean los encabezados a partir de la información que se desea enviar.
4. Se colocan los encabezados en la sesión.
5. Se ejecuta la operación de escritura

Implementación:

```
package transporte;


import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.UUID;
import javax.microedition.io.Connection;
import javax.microedition.io.Connector;
import javax.obex.*;
import java.io.*;


/**
 *
 * @author Huilver Nolasco Aguilar
 */


public class ObexServer  extends ServerRequestHandler implements  Runnable {


    final private String serverURL;
    private boolean cancelWaitingInvoked = false;
    private Thread waitingThread;
    private SessionNotifier notifier;
    private Connection con;
    private Receptor receptor;
    private EnviadorSMS esms;


    public ObexServer(Receptor receptor,String url) {
        this.receptor=receptor;
        serverURL=url;
        esms=new EnviadorSMS();
    }

    /*iniciador de servidor*/
    public void abrir() {
        cancelWaitingInvoked = false;
        waitingThread = new Thread(this);
        waitingThread.start();
    }

    /**cierra el servidor*/
    public void cerrar() {
        cancelWaitingInvoked = true;
        try {
            notifier.close();
        } catch (IOException e) {
            receptor.getPantalla().addMsg(e.getMessage());
        }
    }

    public synchronized void run() {
        try {
            LocalDevice local = LocalDevice.getLocalDevice();
            local.setDiscoverable(DiscoveryAgent.GIAC);
        } catch (Exception e) {
            receptor.getPantalla().addMsg(e.getMessage());
            return;
        }
```

```
    try {
        notifier = (SessionNotifier) Connector.open(serverURL);
    } catch (IOException e2) {
        receptor.getPantalla().addMsg(e2.getMessage());
    }
    while (!cancelWaitingInvoked) {
        try {
            this.createHeaderSet();
            con = notifier.acceptAndOpen(this);
            wait();
            try {
                con.close();
            } catch (IOException e0) {
            }
        } catch (Exception e1) {
            receptor.getPantalla().addMsg(e1.getMessage());
            return;
        }
    }

}


public int onPut(Operation op) {
    try{
        InputStream in = op.openInputStream();
        byte[] fullResult = null;
        byte[] buf = new byte[256];
        ByteArrayOutputStream bout = new ByteArrayOutputStream(2048);
        for (int len = in.read(buf); len >= 0; len = in.read(buf))
            bout.write(buf, 0, len);
        fullResult = bout.toByteArray();
        //lo que sigue es para obtener el telefono y el mensaje que llego del servidor
        String mensajeCompleto=new String(fullResult);
        int inicioTel=mensajeCompleto.indexOf("//")+2;
        int finTel=mensajeCompleto.indexOf(":",inicioTel);
        int inicioMsg=mensajeCompleto.lastIndexOf(':');

        String telefono=mensajeCompleto.substring(inicioTel,finTel);
        String mensaje=mensajeCompleto.substring(inicioMsg);

        receptor.getPantalla().addMsg(mensaje);
        receptor.getPantalla().addMsg(telefono);
        //enviamos el mensaje enviado por el servidor al telefono que nos hizo la solicitud
        esms.connectAndSend(mensaje,telefono,null);

        return  ResponseCodes.OBEX_HTTP_OK;
    }catch(IOException ioe){
        return ResponseCodes.OBEX_HTTP_INTERNAL_ERROR;
    }
}

public int onConnect(HeaderSet request,
        HeaderSet reply){
    try{
        reply.setHeader(HeaderSet.NAME,request.getHeader(HeaderSet.NAME));
        return ResponseCodes.OBEX_HTTP_OK;
    }catch(IOException ioe){
        receptor.getPantalla().addMsg("error en  onConnect");
        receptor.getPantalla().addMsg(ioe.getMessage());
        return    ResponseCodes.OBEX_HTTP_INTERNAL_ERROR;
    }
}

public synchronized void onDisconnect(HeaderSet request, HeaderSet reply) {
    super.onDisconnect(request, reply);
    notify();// detiene  el threat
}

public void onAuthenticationFailure(byte[] userName) {
    receptor.getPantalla().addMsg("entro a onAuthenticationFailure");
}
public int       onGet(Operation op){
    receptor.getPantalla().addMsg("entro a onGet");
```

```
            return  ResponseCodes.OBEX_HTTP_NOT_IMPLEMENTED;
        }
        public void setConnectionID(long id){
            receptor.getPantalla().addMsg("entro a setConnectionID");
        }


        /**
          envía físicamente el mensaje recibido a la máquina servidor
        ***/
        public void enviaBluetooth(String message,String destinationAddress,String nameOuput,String
url){
            DataInputStream in = null;
            DataOutputStream out = null;
            ClientSession session =null;
            Operation operation=null;
            try {
                session = (ClientSession) Connector.open(url);
                HeaderSet response = session.connect(session.createHeaderSet());
                int responseCode = response.getResponseCode();
                if(responseCode != ResponseCodes.OBEX_HTTP_OK)
                    System.err.println("No se pudo completar la conexion: "+responseCode);

                String dat=destinationAddress+":"+message;

                byte[] data = dat.getBytes();
                HeaderSet headers = session.createHeaderSet();
                headers.setHeader(HeaderSet.LENGTH, new Long(data.length));
                headers.setHeader(HeaderSet.NAME, nameOuput);


                operation = session.put(headers);
                out = operation.openDataOutputStream();
                out.write(data);
                out.close();
                int code = operation.getResponseCode();
                if( code != ResponseCodes.OBEX_HTTP_OK){
                    System.err.println("No se pudo completar la transferencia: "+responseCode);
                }
            } catch(IOException e) {
                e.printStackTrace();

            } finally {
                try {
                    if(in != null)
                        in.close();
                    if(out != null)
                        out.close();
                    if(operation!=null)
                        operation.close();
                    if(session!=null)
                        session.close();
                } catch(IOException e) {e.printStackTrace();}
            }
        }

    }
```

RecibidorSMS

Esta clase tiene como propósito permanecer a la espera de mensajes SMS provenientes de SRII-TRC, esto se logra mediante el uso del API de mensajería.

Para la recepción de mensajes SMS se deben efectuar lo siguiente:

1. Crear una conexión en modo servidor SMS, es decir poner el dispositivo en modo servidor (ejemplo de dirección modo servidor "sms://:6222").
2. Pasarle a la conexión el receptor de eventos de recepción de mensaje.

3. Poner a la clase a recibir mensajes SMS.

Implementación:

```
package transporte;


import javax.wireless.messaging.*;
import java.io.*;
import javax.microedition.io.*;



/**
 *
 * @author Huilver Nolasco Aguilar
 */

public class RecibidorSMS implements Runnable {

    private MessageConnection smsconn;
    private Thread receiverThread;
    private MessageListener listener;
    private  String message;
    private Receptor receptor;



   /**contructor de clase,
      @param listener - es un receptor de eventos tipo mensaje
      @param receptor - Objeto del la clase Receptor
      */
    public RecibidorSMS(MessageListener listener,Receptor receptor) {
        this.listener=listener;
        this.receptor=receptor;
    }
     /**Abre la conexión en modo servidor SMS
      @param - puerto en el que se abrirá la conexión */
    public void abrir(String port) {
        String url = "sms://:" + port;

        try {
            smsconn = (MessageConnection) Connector.open(url);
            smsconn.setMessageListener(listener);
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
    /**Recibe  mensajes
      @param - conn  la conexión en la cual se recibirá el mensaje*/
    public void recibir(MessageConnection conn) {
        new Thread(this).start();
    }

    public void run() {
        Message msg = null;
        String senderAddress = null;
        String receivedMessage = null;

        try {
            msg = smsconn.receive();
            if (msg != null) {
                senderAddress = msg.getAddress();
                if (msg instanceof TextMessage) {
                    receivedMessage = ((TextMessage)msg).getPayloadText();
                    setMessage(receivedMessage);

                }

            }
        } catch (IOException e) {
```

```
                System.out.println("IOException" + e.getMessage());
                e.printStackTrace();
            }
            receptor.getPantalla().addMsg(getMessage());
           receptor.enviaEntrada(getMessage(),senderAddress,"sms.txt");
        }

        /**Cierra la conexión */
        public void cerrar() {
            if (smsconn != null) {
                try{smsconn.close();
                } catch(IOException ioe){
                    ioe.printStackTrace();
                }
            }
        }
        /**Obtiene el mensaje actual recibido*/
        public String getMessage(){
            return this.message;
        }
       /**Fija el contenido del mensaje actual recibido, el cual es actualizado al recibir un
       mensaje*/
        public synchronized void setMessage(String msg){
            this.message=msg;
        }
    }
```

Receptor

Esta clase es un demonio[1] que permanece en espera de mensajes SMS y además integra las funcionalidades de las otras clases de este subproyecto. Así que la función de esta clase es la de recibir la información enviada por SRII-TRC y enviarla hacia SRII-SRV.

El inicio del demonio proporcionado por Receptor es mediante una interfaz gráfica provista por la misma clase.

Implementación:

```
    /*Implementación  de la arquitectura diseñada por el Dr. Benjamín Macías Pimentel para proveer
servicios.*/
    package transporte;

    import javax.microedition.midlet.*;
    import javax.microedition.lcdui.*;
    import javax.wireless.messaging.*;
    import java.io.*;
    import javax.microedition.midlet.MIDlet;
    import javax.bluetooth.*;
    import javax.microedition.io.*;
    import javax.obex.*;


    /**
     * @author Huilver Nolasco Aguilar
     */

    public class Receptor extends MIDlet implements CommandListener, MessageListener {
        //private EnviadorBluetooth ebluetooth;
        private EnviadorSMS esms;
        private RecibidorSMS rsms;
        private ObexServer oserver;
        private Pantalla  pantalla;
        private Display display;
```

1 Un demonio o demon o daemon (Disk And Execution Monitor) es "un programa que no es invocado explícitamente, que permanece inactivo a la espera de que alguna condición ocurra "[4]

```
/**Direccion de la tarjeta bluetoth del servidor
 urlC="btgoep://0009DD6005B6:3;authenticate=false;encrypt=false;master=false"; */
 private String     URLC=
 "btgoep://0009DD6000A7:3;authenticate=false;encrypt=false;master=false";
/**servidor bluetooth OBEX que reside en el celular*/
private String URLS=
"btgoep://localhost:" + new UUID(0x1105).toString(+";name=myObexPush";
/**puerto en el que se reciben los mensajes sms*/
private String SMS_PORT="6222";

private boolean started=false;


public void startApp() {
    display = Display.getDisplay(this);
    esms=new EnviadorSMS();
    rsms=new RecibidorSMS(this,this);
    pantalla=new Pantalla(this,this);
    oserver=new ObexServer(this,URLS);
    display.setCurrent(pantalla);
}


public void pauseApp() {
}



public void destroyApp(boolean unconditional) {
    if(unconditional==true){
        pantalla.addMsg("Servidor terminado");
        rsms.cerrar();
        pantalla=null;
        esms=null;
        display=null;
        notifyDestroyed();

    }
}


public void commandAction(Command command, Displayable displayable) {
    if(command.getCommandType()==Command.EXIT){
        destroyApp(true);
    }else
        if(command.getCommandType()==Command.OK&&!started) {
        rsms.abrir(SMS_PORT);//servidor sms abierto
       oserver.abrir();//servidor obex abierto
        pantalla.addMsg("Servidor Iniciado");
        started=true;
        display.setCurrent(pantalla);
        }
}


public Display getDisplay() {
    return display;
}


public Pantalla getPantalla(){
    return this.pantalla;
}


/** Mandado por la interfaz  MessageListener*/
public void notifyIncomingMessage(MessageConnection conn) {
    recibeEntrada(conn);
}
```

```
      /**Clase Interna que extiende a un texbox*/
    public class Pantalla extends TextBox {
        private Command start;
        private Command end;
        private MIDlet midlet;
        private CommandListener listener;



        public Pantalla(MIDlet midlet,CommandListener listener) {
            super("Servidor: ",null,4000,TextField.UNEDITABLE);
            this.midlet=midlet;
            this.listener=listener;
            start = new Command("Iniciar",Command.OK,1);
            end = new Command("Terminar",Command.EXIT,2);
            addCommand(start);
            addCommand(end);
            setCommandListener(listener);
        }

        public synchronized void addMsg(String msg) {
            String current = getString();
            if(current.length()==4000) {
                //making room for the new msg
                delete(current.length()-1000,1000);
            }
            insert(msg + "\n",current.length());
        }
    }


    /**métodos espacificicados en el diseño**/
    /**detecta el envío de un mensaje por parte de un TR enviado por la red de
     * telefonía*/
    public void recibeEntrada(MessageConnection conn){
        rsms.recibir(conn);
    }

    /**envía físicamente el mensaje recibido a la máquina servidor*/
    public void enviaEntrada(String message,String destinationAddress,String nameOuput){
        oserver.enviaBluetooth(message,destinationAddress,nameOuput,this.URLC);
    }

    /**detecta el envío de un mensaje por parte de Transporte*/
    public void recibeSalida(String mensajeSalida){
        enviaSalida("salida");
    }

    /**envía físicamente el mensaje por la red de telefonía pública*/
    public void enviaSalida(String mensajeSalida){}
}
```

Implementación de algunos servicios

La implementación de la arquitectura diseñada por el Dr. Benjamín Macías Pimentel para proveer servicios, se encuentra en el subproyecto SRII-SRV.

El objetivo de este subproyecto es la recepción de la información enviada por SRII-TRS, manipulación de dicha información para atender el servicio pedido y enviar respuesta del servicio solicitado a la terminal remota cliente.

En esta sección sólo se describe el núcleo de la arquitectura y la implementación del servicio de búsqueda de información en el Internet, particularmente de la wikipedia (http://www.wikipedia.org/).

Componente de manipulación

Este componente es el encargado de proveer los servicios solicitados. Por ejemplo si el servicio que se pide es el envío de un correo, este componente debe contener una clase específica para este servicio, si la petición es buscar una palabra, el componente debe contener una clase que efectué dicho servicio, y así para cada tipo de servicio que se pretenda brindar.

Paquete manipulador

Este paquete va a contener todas y cada una de las implementaciones de servicios que se propone brindar.

Manipulador

Es una interfaz que debe ser implementada por cualquier clase que quiera brindar un servicio.

Los métodos que esta interfaz tiene son:

- ejecutarServicio(): este método permite la ejecución de un servicio. Debe generar un Log de estado de la operación y debe generar un archivo de respuesta al servicio.
- imprimeSalida(String pathFile): genera el archivo de respuesta del servicio.
- regresaLog(): escribe a la bitácora el estado de la operación

Implementación:

```
package manipulador;

/**
 *
 * @author Huilver Nolasco Aguilar
 */
public interface Manipulador {

    /*
     *
     *  Este metodo es invocado para ejecutar el servicio
     *  @return true si fue ejecutado correctamente, false en otro caso
     */
    public boolean ejecutarServicio();

    /*
     *
     *Regresa el log de salida del servicio
     *@return El log de salida del servicio
     */

    public String getLog();

    /*
     *Este metodo implementa la escritura de un archivo con ubicación prefedefinida
     *@param la ruta absoluta del archivo de salida
     *
     */
    public void imprimeSalida(String pathFile);

}
```

Paquete manipulador.search

Este paquete contiene algunas implementaciones correspondientes al servicio de búsqueda.

Clase abstracta Searcher.

Clase abstracta correspondiente a un buscador de información cuyos dos únicos métodos son:

- diplayUrl(String strUrl):este método regresa una cadena con el contenido del url especificado como parámetro
- search(String word):método abstracto que cualquier clase de búsqueda debe de implementar.

Implementación:

```
/**
 *
 * @author Huilver Nolasco Aguilar
 */
package manipulador.search;

import java.io.InputStream;
import java.net.URL;
import java.util.Vector;
import utilerias.StackTraceUtil;


public abstract class Searcher {

    protected StackTraceUtil stacktrace= new StackTraceUtil();
    protected StringBuffer Log=new StringBuffer();


    protected String diplayUrl(String strUrl){
        StringBuffer res=new StringBuffer("");
        try {
            URL url = new URL(strUrl);
            InputStream in = url.openStream();
            byte[] buf = new byte[1024];
            int len;
            while ((len = in.read(buf)) > 0) {
                for (int i = 0; i < len; i++) {
                    res.append((char)buf[i]);
                }
            }

            in.close();
        } catch (Exception e) {
            Log.append(stacktrace.getCustomStackTrace(e));
            System.out.println(Log);
        }
        return res.toString();
    }

    public String getLog(){
        return Log.toString();
    }
```

```
    public abstract Vector<String> search(String word);
}
```

GoogleSearcher

Clase que hereda de la clase Searcher y por tanto implementa el método search, pero lo hace mediante el servicio de búsqueda de Google.

Implementación:

```
/**
 *
 * @author Huilver Nolasco Aguilar
 */
package manipulador.search;
import com.google.soap.search.*;
import java.util.Vector;
import utilerias.filter.HtmlFilter;


public class GoogleSearcher extends Searcher {
    private final String CLIENT_KEY = "Msxvmo5QFHJCVPseiXVS9BL1xwHiErR/";
    private final String DIRECTIVE = "search";
    private int maxResults;


    public GoogleSearcher(int maxResults){
        this.maxResults=maxResults;
    }


    public Vector<String> search(String word){
        GoogleSearch gs = new GoogleSearch();
        gs.setKey(CLIENT_KEY);
        gs.setQueryString(word);
        gs.setMaxResults(this.maxResults);
        gs.setLanguageRestricts("lang_es");
        Vector <String>res=new Vector<String>();
        try{
            GoogleSearchResultElement [] googleRes=gs.doSearch().getResultElements();
            for(GoogleSearchResultElement el:googleRes)
                res.add(el.getURL());
        }catch(GoogleSearchFault gsf){
            Log.append((stacktrace.getCustomStackTrace(gsf)));
            System.out.println(Log);
        }
        return res;
    }


}
```

YahooSearcher

Esta clase, al igual que GoogleSearcher, es una clase de búsqueda que implementa a la clase Searcher, pero que implementa al método search mediante el servicio de búsqueda de Yahoo.

Implementación:

```
package manipulador.search;
/**Classe que permite la busqueda mediante el engine de yahoo
 * @author Huilver Nolasco Aguilar
 */
import java.io.IOException;
import java.util.Vector;
```

```
        import utilerias.filter.HtmlFilter;

        import com.yahoo.search.SearchClient;
        import com.yahoo.search.SearchException;
        import com.yahoo.search.WebSearchRequest;
        import com.yahoo.search.WebSearchResult;
        import com.yahoo.search.WebSearchResults;


        public class YahooSearcher extends Searcher{
                                        private        final        String        CLIENT_KEY        =
"Nzh_2YjV34HbMGS8_WaRXwZ0EqsGHE5ri7jEbZFyk_b2X6fc9RVzcbLt.ffqMcIe";
            private int maxResults;


            public YahooSearcher(int maxResults){
                this.maxResults=maxResults;
            }

            public Vector<String> search(String toSearch){
                Vector <String>res=new Vector<String>();
                SearchClient client = new SearchClient(CLIENT_KEY);
                WebSearchRequest request = new WebSearchRequest(toSearch);
                request.setResults(this.maxResults);
                try {
                    WebSearchResults results = client.webSearch(request);
                    for (int i = 0; i < results.listResults().length; i++) {
                        WebSearchResult result = results.listResults()[i];
                        res.add(result.getUrl());
                    }
                } catch (IOException e) {
                    Log.append(stacktrace.getCustomStackTrace(e));
                    System.out.println(Log);
                } catch (SearchException e) {
                    Log.append(stacktrace.getCustomStackTrace(e));
                    System.out.println(Log);
                }
                return res;
            }

        }
```

Paquete utilerias.filter

Este paquete está pensado para contener todos los posibles filtros de documentos obtenidos en Internet, como documentos en formato pdf, html, doc, etc. Por el alcance de este proyecto sólo se implementa una clase que filtra documentos html.

HtmlFilter

Clase que tiene como objetivo extraer la información contenida en un documento html, es decir eliminar cualquier contenido que corresponda al lenguaje html.

Los métodos que esta clase posee son:

filtraHtml(CharSequence input): Se encarga de realizar la extracción de la información del parámetro de entrada, es decir, elimina cualquier código html contenido en el documento dejando así sólo el texto plano.

eliminaInvalido(String input, String qoute, String toFind):determina si la cadena toFind se encuentra encerrado por qoute en la cadena input y

de ser así elimina dicha secuencia.

String eliminaHtml(String input,String tag):elimina todas las ocurrencias de tag en la cadena input.

Implementación:

```
package utilerias.filter;

/**
 * @author Huilver Nolaso Aguilar
 * @version 0.1 - 3
 *
 */
public class HtmlFilter {

		/**
		 * @param args
		 */

		public String filtraHtml(CharSequence input){
			String cadena=input.toString().toLowerCase();
			String [] htmlTags={"script","style"};
			cadena=eliminaInvalido(cadena,"\"",">");
			cadena=eliminaInvalido(cadena,"\"","<");
			cadena=eliminaInvalido(cadena,"'",">");
			cadena=eliminaInvalido(cadena,"'","<");
			for(String tag:htmlTags)
			cadena=eliminaHtml(cadena,tag);

			return eliminaHtml(cadena,null);
		}



		/**Determina si la cadena toFind se encuentra encerrado por qoute en la cadena input y de ser asi elimina dicha secuencia*/
		public String eliminaInvalido(String input, String qoute, String toFind) {
			StringBuffer cadena=new StringBuffer(input);
			int indexI = cadena.indexOf(qoute);
			int indexF;
			String temporal;
			while (indexI!=-1&&cadena.length() > indexI) {
				indexF = cadena.indexOf(qoute, indexI+1);
				if (indexF == -1) break;
				temporal = cadena.substring(indexI, indexF + 1);
				if (temporal.indexOf(toFind) > 0){
					cadena.delete(indexI, indexF + 1);
					indexI = cadena.indexOf(qoute,indexI);
				}
				indexI = cadena.indexOf(qoute,indexF+1);
			}
			return cadena.toString();
		}

		/**elimina los tags de html*/
		public String eliminaHtml(String input,String tag) {
			StringBuffer cadena=new StringBuffer(input);
			int indexI=-1;
			int indexF;
			if (tag==null||tag.length()==0)
				indexI = cadena.indexOf("<");
			else indexI = cadena.indexOf("<"+tag);

			while (indexI!=-1&&cadena.length() > indexI) {
				if (tag==null||tag.length()==0)
					indexF = cadena.indexOf(">", indexI);
				else
					if((indexF=cadena.indexOf("</"+tag,indexI))!=-1)
						indexF-=1;
```

```
                                else{
                                        indexF = cadena.indexOf("/>", indexI);
                                        int idxTmp=cadena.indexOf(">", indexI);
                                        if(indexF+1==idxTmp) indexF+=1;
                                        else indexF= idxTmp;
                                }
                        if (indexF < 0) break;

                    if (tag==null||tag.length()==0)
                        cadena.delete(indexI, indexF+1);
                        else
                                cadena.delete(indexI, indexF+1);
                        if (tag==null||tag.length()==0)
                                indexI = cadena.indexOf("<",indexI);
                        else indexI = cadena.indexOf("<"+tag,indexI);
                }

              String res=cadena.toString().replaceAll("(\t|\f|\r)","\n");
              res=res.replaceAll("\n{2,}","\n");

                    return res;
            }


}
```

Conclusiones

Llevé a cabo las labores de implementación del módulo de transporte (dirección celular – servidor) del proyecto PAPIIT, así como la implementación de algunos servicios (búsquedas en Internet). Esto se hizo utilizando el diseño propuesto por el Dr. Benjamín Macías Pimentel. La implementación fue probada y se demostró que cumplía con las metas establecidas. Además, en colaboración con Abraham Valle Alvarez, llevé a cabo un análisis y diseño alternativos partiendo de la especificación original.

La realización del sistema que permite el envío de información desde un dispositivo móvil hacia un servidor requirió de servicios tales como SMS y *Bluetooth*. El uso de estos servicios resulta en una forma muy barata de establecer comunicación entre dispositivos móviles y computadoras. Además se obtiene una comunicación asíncrona que era uno de los objetivos buscados.

El uso de la comunicación asíncrona entre dispositivos móviles resulta ser más barata con el uso de mensajes cortos (SMS). Es también hasta cierto grado más practica debido que no se requiere de una comunicación constante durante todo el tiempo con el servidor; sólo se solicita el servicio con la comodidad de un SMS y en cuanto la petición del servicio pedido esté listo, regresa como respuesta en forma de mensaje corto también.

Anexo A

Especificación de un Sistema Informático para el Manejo de Información a Través de Terminales Remotas

Autor: Benjamín Macías

Proyecto: PAPIIT UNAM ("Sistemas Remotos de Interacción con Internet")
Entregable: IS-3 1
Elaboración: 10 de julio 2006
Última revisión: 30 de agosto de 2006
Estatus: Pendiente

0. Introducción

Este documento presenta una especificación de los requerimientos básicos del sistema de transporte de información asociado con el proyecto. Se usa para ello la metodología de estudio de escenarios ("use cases") de orientación a objetos (e. g. Jacbson et al. 1999). De acuerdo a esta metodología, se espera que el proyecto se desarrolle en varios ciclos. Por ello, mas que presentar una especificación completa, nos concentramos en los casos centrales del sistema, con el propósito de producir un prototipo del sistema entero funcionando a la brevedad posible.

Seguiremos al proyecto original en identificar informalmente a los componentes principales del sistema como "transporte", "control", "almacenamiento", y así sucesivamente. En el documento de análisis y diseño daremos una descripción más adecuada de ellos.

1. Requerimientos Funcionales

1.1 Escenario 1.

Transferencia de mensajes de correo electrónico del servidor al cliente. En su presentación, el proyecto esboza el siguiente escenario de uso:

1. La usuaria final envía un mensaje SMS o de correo electrónico solicitando la lectura de su buzón de correo electrónico: "LEE CORREO NUEVO"
2. El componente de Transporte recibe el mensaje, y lo almacena.
3. El componente de Transporte notifica a Control la llegada del mensaje.
4. Control recupera los datos del usuario e indica al Manipulador que inicie un diálogo con el servidor de correo electrónico y baje los correos nuevos (i.e. comportándose como un cliente POP3).

5. El Manipulador transfiere los datos al Archivista.
6. Control solicita al Archivista la producción de sumarios para pasar a la salida.

7. Dependiendo del caso y de la configuración del sistema, la respuesta del Archivista

   puede ser alternativamente:

   (a) "NO HAY MENSAJES NUEVOS"
   (b) lista de encabezados, ordenados tal vez por importancia del que los envió.
   (c) Contenido sintetizado de un único, o más importante, mensaje.

8. Control instruye al Transportista, quien envía el mensaje SMS final a la usuaria.

De esta descripción se deriva el siguiente escenario (cada enunciado en cursivas representa un enunciado del esbozo recién mencionado; las líneas en paréntesis completan la descripción pero se consideran como no relevantes para propósitos de esta especificación):

1.1.1 Paso 1

La usuaria final envía un mensaje SMS o de correo electrónico solicitando la lectura de su buzón de correo electrónico.

(a) La usuaria enciende su TR.
(b) La usuaria abre la ventana en su TR que activa el sistema de intercambio de información.
(c) La usuaria selecciona la opción de lectura de buzón de correo.
(d) La usuaria completa los datos necesarios.
(e) La usuaria selecciona la opción de enviar el comando.
(f) La ventana envía el mensaje en forma de mensaje SMS.
(g) La usuaria apaga su TR.

1.1.2 Paso 2

El componente de Transporte recibe el mensaje, lo almacena, y notifica a Control la llegada del mensaje.

(a) El componente de transporte se encuentra activado, y listo para recibir mensajes de un cliente.
(b) El componente de transporte recibe un mensaje.
(c) El componente de transporte notifica al componente de control el mensaje recibido.
(d) El componente de transporte regresa a su estado inicial, listo para recibir mensajes nuevos.

1.1.3 Paso 3

El componente de control entra en un diálogo con los componentes de manipulación y de almacenamiento para elaborar una respuesta; eventualmente, el componente de control es notificado que la respuesta ha

sido generada. El contenido de la respuesta está en el archivista.

(a) El componente de control se encuentra activado, y listo para recibir mensajes del componente de transporte.
(b) El componente de control solicita al componente de almacenamiento que guarde el mensaje recibido.
(c) El componente de control instruye al componente de manipulación sobre las acciones a realizar sobre el mensaje recibido.
(d) El componente de control regresa a su estado inicial, listo para recibir nuevos mensajes.
(e) Los componentes de manipulación colaboran para producir una respuesta. Cuando la respuesta está lista, el componente de manipulación notifica al componente de control.

1.1.4 Paso 4

Control instruye al componente de transporte, quien envía el mensaje SMS final a la usuaria.

(a) El componente de control se encuentra activado, y listo para recibir mensajes del componente de manipulación.
(b) El componente de control recibe el mensaje del componente de manipulación de que el mensaje está listo para ser entregado.
(c) El componente de control recupera los detalles de la dirección de destino del mensaje.
(d) El componente de control le da al mensaje de salida su formato final (usando c.).
(e) El componente de control envía el mensaje de respuesta en forma de mensaje SMS a la dirección deseada.
(f) La usuaria recibe el mensaje SMS con la respuesta.
(g) La usuaria enciende su TR y arranca la ventana para recibir mensajes.
(h) La usuaria lee el mensaje de respuesta.

Clase Usuario

Esta clase representa al dueño del TR; no es necesario modelarlo.

Clase Ventana

Esta clase representa al navegador ejecutando en el TR del cliente. Se introduce esta clase para propósitos de análisis; es posible que su funcionalidad sea redefinida para simplificarla en el diseño. Esta clase es la única clase ejecutando en el TR del cliente.

Nota: La implementación de esta clase es opcional. Para el caso de los teléfonos celulares, éstos ya cuentan con software para enviar mensajes. Sin embargo, es conveniente contar con un cliente en el TR del usuario para poder formatear mensajes,hacer parsing sobre los mensajes recibidos, etc.

Métodos

abrir():activa el sistema de intercambio de información cliente que pide el servicio; solicita: Usuario; aparece: en 1.b.

int seleccionaFuncion(): se selecciona el tipo de función a realizar; solicita: Usuario; aparece: 1.c.

String[] completaDatos(): se completa los datos necesarios; solicita: Usuario; aparece: 1.d.

solicitaEnvio(String): ordenar el envío del mensaje; solicita: Usuario; aparece: 1.e.

envia(String): envío de mensaje SMS al componente de control; solicita: Ventana; aparece: 1.f

recibe(): despliega un mensaje recibido en el TR; solicita: Usuario.

Nota 1: el mensaje ha llegado a través de la red telefónica; la función de este método (que se ejecuta una vez que se ha abierto la ventana y seleccionado la recepción de un mensaje) sucede una vez que el TR ha recibido el mensaje y lo almacenado.

Nota 2: Se usan clases terminales como String en este punto porque se presume que la funcionalidad del TR impide o hace impráctico usar clases para envolver datos tales como mensajes.

Encadenamiento(pendiente).

Clase Receptor

Esta clase instancia la primera parte del componente de transporte. Su misión es recibir mensajes remotos de TRs de usuarios a través de telefonía pública y canalizarlos a la otra parte del componente de transporte. Esta clase reside en el teléfono celular conectado al servidor. La clase Transporte (ver abajo) instancia el resto de la funcionalidad del componente de transporte.

Métodos

recibeEntrada(): detecta el envío de un mensaje por parte de un TR enviado por la red de telefonía; invoca a enviaEntrada(); solicita: Ventana; aparece: 2.a, 2.b.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: esta función aparece mencionada una vez en la especificación; ha sido desdoblada en dos para propósitos del análisis; ver Transporte.recibe().

enviaEntrada(MensajeEntrada): envía físicamente el mensaje recibido a la

máquina servidor; invoca a Transporte.recibeEntrada(); solicita: recibeEntrada(); aparece: no aparece

Nota 1: esta función conceptualmente trabaja como un evento que se dispara cada vez que recibeEntrada() se activa.

Nota 2: el envío se hace físicamente conectando el celular con el puerto USB del servidor a través de un cable o de tecnología *Bluetooth.*

recibeSalida(MensajeSalida): detecta el envío de un mensaje por parte de Transporte; invoca a enviaSalida(); solicita: Transporte.enviaSalida(); aparece: 4.e

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: esta función recibe físicamente el mensaje conectando el celular con el puerto USB del servidor a través de un cable o de tecnología *Bluetooth.*

enviaSalida(MensajeSalida): envía físicamente el mensaje por la red de telefonía pública; solicita: recibeSalida(); aparece: 4.e.

Clase Transporte

Esta clase instancia la segunda parte del componente de transporte. Su misión es recibir mensajes de la clase Receptor y canalizarlos al resto de los componentes del sistema.

Esta clase reside en la máquina servidor.

Métodos

recibeEntrada(MensajeEntrada): detecta el envío de un mensaje de entrada por parte de la clase Receptor; invoca a enviaEntrada(); solicita: Receptor.envia(); aparece: 2.a, 2.b

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: esta función aparece mencionada una vez en la especificación; ha sido desdoblada en dos para propósitos del análisis; ver Receptor.recibe().

enviaEntrada(MensajeEntrada): envía el mensaje recibido al componente de control para su proceso; invoca a Control.recibeEntrada(); solicita: recibeEntrada(); aparece: 2.c.

Nota 1: esta función conceptualmente trabaja como un evento que se dispara cada vez que recibeEntrada() se activa.

recibeSalida(MensajeSalida): detecta el envío de un mensaje de salida por parte de la clase Control; invoca a enviaSalida(); solicita:

Control.procesaMensajeSalida(); aparece: 4.e.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: la funcionalidad descrita en 4.e esta repartida entre Control.procesaMensajeSalida(), este método, recibeSalida() y Receptor.recibeSalida().

enviaSalida(MensajeSalida)
función: envía el mensaje recibido al Receptor; invoca a Receptor.recibeSalida(); solicita: recibeSalida(); aparece: 4.e.

Clase Control

Esta clase instancia el componente de control. Su tarea es coordinar a los componentes que almacenan el mensaje, lo procesan, y producen la respuesta. Puede pensarse en él como el control del tráfico que fluye hacia el servidor desde los TRs y desde los TRs hacia el servidor. Esta clase reside en la máquina servidor.

Métodos

recibeEntrada(MensajeEntrada): detecta el envío de un mensaje por parte de la clase Transporte; invoca a procesaMensajeEntrada(); solicita: Transporte.envia(); aparece: 2.c.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

procesaMensajeEntrada(MensajeEntrada): organiza el proceso de un mensaje de entrada:

1. Debe de almacenar los detalles generales relativos al mensaje de entrada (hora de llegada, identificador del TR, longitud, identificador interno, etc).
2. Ver que se almacene.
3. Establecer el tipo de servicio solicitado en el mensaje.
4. Ponerse en contacto con el componente relevante para brindar el servicio.

Solicita: recibeEntrada().
Aparece: 3.b, 3.c.

Nota 1: esta función conceptualmente trabaja como un evento que se dispara cada vez que recibe() se activa.

Nota 2: la especificación menciona solamente un componente de manipulación. Para propósitos de análisis podemos suponer que hay un componente para cada tipo servicio que se presta, todos implementando la misma interfase.

Nota3: hay una clase extra implícita para representar el mensaje después de que ha sido procesada. La llamaremos MensajeEntradaInterno.

recibeSalida(MensajeSalidaInterno): detecta que algún componente de manipulación a terminado de elaborar un mensaje de salida, por lo que éste está listo para ser enviado; invoca a procesaMensajeSalida(); solicita: Manipulador.envia(); aparece: 3.d.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

procesaMensajeSalida(MensajeSalidaInterno):organiza el proceso de producción de un mensaje de salida:

1. Almacena los detalles generales del mensaje de salida (componente que lo produjo, hora, identificador del mensaje).
2. Almacena el mensaje de salida (esto es, transfiere una copia del mensaje que llego del Manipulador al Almacenamiento).
3. Establece los detalles del TR receptor.
4. Formatea el mensaje en su forma final y lo envía al Transporte. Invoca a Transporte.recibeSalida().

Solicita: recibeSalida().
Aparece: 4.b, 4.c, 4.d, 4.e.

Clase Almacenamiento

Esta clase es una clase auxiliar de Control; su función es servir como la memoria permanente del sistema.

Métodos
(pend)

Interface Manipulador

Esta clase abstracta define el API entre Control y cada una de sus subclases. Cada subclase implementa un servicio (lectura de correo, agenda, etc) específico del sistema.

Métodos
(pend)

Otras clases (pend):

MensajeEntrada
MensajeSalida
MensajeEntradaInterno
MensajeSalidaInterno

Anexo B

Análisis y Diseño de un Sistema Informático para el Manejo de Información a Través de Terminales Remotas

Autor: Benjamín Macías
Proyecto: PAPIIT UNAM ("Sistemas Remotos de Interacción con Internet")
Entregable: IS-3 2
Elaboración: 31 de julio 2006
Última revisión: 30 de agosto de 2006

0. Introducción

Este documento presenta el análisis y diseño del sistema de transporte de información asociado con el proyecto. Este documento está basado en la especificación de requerimientos del sistema contenida en el documento IS-3 1. Como dicho documento, el análisis y diseño usan de manera general la metodología de orientación a objetos (e. g. Jacbson et al. 1999), aunque adaptada a nuestras necesidades.

1. Análisis

1.1 Clases y su funcionalidad

El elemento principal del análisis queda dado por las clases que se derivan de la especificación. Nótese que estas clases se han introducido para propósitos de replantear la descripción de la especificación en términos computacionales. Al momento del diseño, algunas de ellas pueden desaparecer o redefinirse en otras clases.

1.1.1 Clase Usuario

Esta clase representa al dueño del TR; no es necesario modelarlo.

1.1.2 Clase Ventana

Esta clase representa al navegador ejecutando en el TR del cliente. Se introduce esta clase para propósitos de análisis; es posible que su funcionalidad sea redefinida para simplificarla en el diseño. Esta clase es la única clase ejecutando en el TR del cliente.

Nota: La implementación de esta clase es opcional. Para el caso de los teléfonos celulares, éstos ya cuentan con software para enviar mensajes. Sin embargo, es conveniente contar con un cliente en el TR del usuario para poder formatear mensajes, hacer parsing sobre los mensajes recibidos, etc.

Métodos

abrir(): activa el sistema de intercambio de información del cliente que solicita el servicio: Usuario; aparece mencionada en la especificación en: 1.b

int seleccionaFuncion()
función: se selecciona el tipo de función a realizar solicita: Usuario aparece: 1.c.

String[] completaDatos()
función: se completa los datos necesarios; solicita: Usuario; aparece: 1.d

solicitaEnvio(String)
función: ordenar el envío del mensaje; solicita: Usuario; aparece: 1.e

envia(String)
función: envío de mensaje SMS al componente de control; solicita: Ventana; aparece: 1.f.

recibe()
función: despliega un mensaje recibido en el TR; solicita: Usuario.

Nota1: el mensaje ha llegado a través de la red telefónica; la función de este método (que se ejecuta una vez que se ha abierto la ventana y seleccionado la recepción de un mensaje) sucede una vez que el TR ha recibido el mensaje y lo almacenado.

Nota2: Se usan clases terminales como String en este punto porque se presume que la funcionalidad del TR impide o hace impráctico usar clases para envolver datos tales como mensajes.

1.1.3 Clase Receptor

Esta clase instancia la primera parte del componente de transporte. Su misión es recibir mensajes remotos de TRs de usuarios a través de telefonía pública y canalizarlos a la otra parte del componente de transporte. Esta clase reside en el teléfono celular conectado al servidor. La clase Transporte (ver abajo) instancia el resto de la funcionalidad del componente de transporte.

Métodos

recibeEntrada()
función: detecta el envío de un mensaje por parte de un TR enviado por la red de telefonía; invoca a enviaEntrada(); solicita: Ventana; aparece: 2.a, 2.b.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: esta función aparece mencionada una vez en la especificación; ha sido desdoblada en dos para propósitos del análisis; ver Transporte.recibe().

enviaEntrada(MensajeEntrada): envía físicamente el mensaje recibido a la máquina servidor; invoca a Transporte.recibeEntrada(); solicita: recibeEntrada(); aparece: no aparece.

Nota 1: esta función conceptualmente trabaja como un evento que se dispara cada vez que recibeEntrada() se activa.

Nota 2: el envío se hace físicamente conectando el celular con el puerto USB del servidor a través de un cable o de tecnología *Bluetooth.*

recibeSalida(MensajeSalida): detecta el envío de un mensaje por parte de Transporte; invoca a enviaSalida();solicita:Transporte.enviaSalida(); aparece: 4.e.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: esta función recibe físicamente el mensaje conectando el celular con el puerto USB del servidor a través de un cable o de tecnología *Bluetooth.*

enviaSalida(MensajeSalida): envía físicamente el mensaje por la red de telefonía pública solicita: recibeSalida(); aparece: 4.e.

1.1.4 Clase Transporte

Esta clase instancia la segunda parte del componente de transporte. Su misión es recibir mensajes de la clase Receptor y canalizarlos al resto de los componentes del sistema. Esta clase reside en la máquina servidor.

Métodos

recibeEntrada(MensajeEntrada): detecta el envío de un mensaje de entrada por parte de la clase Receptor; invoca a enviaEntrada(); solicita: Receptor.envia(); aparece: 2.a, 2.b.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: esta función aparece mencionada una vez en la especificación; ha sido desdoblada en dos para propósitos del análisis; ver Receptor.recibe().

enviaEntrada(MensajeEntrada)
función: envía el mensaje recibido al componente de control para su proceso; invoca a Control.recibeEntrada(); solicita: recibeEntrada(); aparece: 2.c

Nota 1: esta función conceptualmente trabaja como un evento que se dispara cada vez que recibeEntrada() se activa.

recibeSalida(MensajeSalida): detecta el envío de un mensaje de salida por parte de la clase Control; invoca a enviaSalida(); solicita: Control.procesaMensajeSalida(); aparece: 4.e.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

Nota 2: la funcionalidad descrita en 4.e esta repartida entre Control.procesaMensajeSalida(), este método, recibeSalida() y Receptor.recibeSalida().

enviaSalida(MensajeSalida): envía el mensaje recibido al Receptor; invoca a Receptor.recibeSalida(); solicita: recibeSalida(); aparece: 4.e.

1.1.5 Clase Control

Esta clase instancia el componente de control. Su tarea es coordinar a los componentes que almacenan el mensaje, lo procesan, y producen la respuesta. Puede pensarse en él como el control del tráfico que fluye hacia el servidor desde los TRs y desde los TRs hacia el servidor. Esta clase reside en la máquina servidor.

Métodos

recibeEntrada(MensajeEntrada): detecta el envío de un mensaje por parte de la clase Transporte; invoca a procesaMensajeEntrada(); solicita: Transporte.envia(); aparece: 2.c.

Nota 1: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

procesaMensajeEntrada(MensajeEntrada): organiza el proceso de un mensaje de entrada:

9. Debe de almacenar los detalles generales relativos al mensaje de entrada (hora de llegada, identificador del TR, longitud, identificador interno, etc).
10. Ver que se almacene.
11. Establecer el tipo de servicio solicitado en el mensaje.
12. Ponerse en contacto con el componente relevante para brindar el servicio.

Solicita: recibeEntrada().
Aparece: 3.b, 3.c.

Nota 1: esta función conceptualmente trabaja como un evento que se dispara cada vez que recibe() se activa.

Nota 2: la especificación menciona solamente un componente de manipulación. Para propósitos de análisis podemos suponer que hay un componente para cada tipo servicio que se presta, todos implementando la misma interfase.

Nota3: hay una clase extra implícita para representar el mensaje después de que ha sido procesada. La llamaremos MensajeEntradaInterno.

recibeSalida(MensajeSalidaInterno): detecta que algún componente de manipulación a terminado de elaborar un mensaje de salida, por lo que éste está listo para ser enviado; invoca a procesaMensajeSalida(); solicita: Manipulador.envia(); aparece: 3.d.

Nota: conceptualmente, esta función trabaja como un demonio en un ciclo infinito.

procesaMensajeSalida(MensajeSalidaInterno): organiza el proceso de producción de un mensaje de salida:

(d) Almacena los detalles generales del mensaje de salida (componente que lo produjo, hora, identificador del mensaje).
(e) Almacena el mensaje de salida (esto es, transfiere una copia del mensaje que llego del Manipulador al Almacenamiento).
(f) Establece los detalles del TR receptor.
(g) Formatea el mensaje en su forma final y lo envía al Transporte. Invoca a Transporte.recibeSalida()

Solicita: recibeSalida().
Sparece: 4.b, 4.c, 4.d, 4.e.

1.1.6 Clase Almacenamiento

Esta clase es una clase auxiliar de Control; su función es servir como la memoria permanente del sistema.

Métodos
(pend)

1.1.7 Interface Manipulador

Esta clase abstracta define el API entre Control y cada una de sus subclases. Cada subclase implementa un servicio (lectura de correo, agenda, etc.) específico del sistema.

Métodos
(pend)

1.1.8 Otras clases (pend):

MensajeEntrada
MensajeSalida
MensajeEntradaInterno
MensajeSalidaInterno

1.2 Flujo de Control (PEND)

2. Diseño

El siguiente código en pseudo-Java especifica la manera de implementar el análisis de la sección 1. Algunos detalles han quedado sin definir, pendiente a los detalles de la implementación del teléfono celular con el que estaremos trabajando, tanto para implementar la clase Ventana en el TR del usuario, como el método Receptor en el celular asociado al servidor.

Sin embargo, debe de haber suficiente detalle aquí para que la implementación sea inmediata.

2.1 Código

```
/*
      Ejemplifica definición de ventana en el TR.
      Detalles de widgets, eventos etc, son ilustrativos, no exactos.
      Reside en el celular del usuario.
*/

Ventana {

      Window v; // ventana
      Botton b; // boton etc

      // Llamada externa
      abrir() {
             v = new Window();
             b = new Botton();
             b.event( accionEvento() ); // ata evento a boton
             v.attach( b );               // ata boton a ventana
             ...
             v.display();                 // abre ventana
      }

      // Obtiene id función de ventana
      int seleccionaFuncion() {
             return Integer.parseInt(v.value("FUNCION"));
      }

      // Obtiene datos de ventana
      String seleccionaDatos() {
             return Integer.parseInt(v.value("DATO"));
      }

      ...

      // Se acciona con el boton de la ventana
      accionEvento() {
             int i = v.seleccionaFuncion();
             // selecciona accion con base a valor función
             switch (i) {
                    case Constantes.ENVIA_CORREO_E:
                           String mensaje = v.seleccionaDatos();
                           envia( cifrado (i,mensaje) );
                    break;
                    ...
             }
      }

      // Cifrado: función opcional para formatear mensaje etc
      String cifrado(int,String) {
             ...
      }

      // Envía un mensaje vía red telefónica
      envia(String) {
             // código nativo
```

```
            ...
      }

      ...
}

/*
      Modela 1a parte del componente de transporte.
      Reside en el celular del servidor.
*/
Receptor {

      Sistema s;

      Receptor(Sistema s) {
            this.s = s;
            recibeEntrada();                    // se activa al arrancar el sistema
      }

      // Crea un evento escuchando a nuevos mensajes
      // Ilustrativo, detalles no son exactos
      recibeEntrada() {
            Event e = new Event( System.incomingMessage(), enviaEntrada() );
            e.start();
      }

      // Se activa con nuevo mensaje
      enviaEntrada() {
            MensajeEntrada mensaje =
                  new MensajeEntrada (System.incomingMessage().contents() );
            s.t.enviaEntrada(mensaje);
      }

      ...

}

/*
      Modela 2a. parte del componente de transporte.
      Reside en el servidor.
*/
Transporte {

      Sistema s;

      Transporte(Sistema s) { this.s = s;}

      recibeEntrada(...){}                      // innecesario

      enviaEntrada(MensajeEntrada m) {
            s.c.procesaMensajeEntrada(m);
      }

      ...
}

/*
      Modela componente de control.
      Reside en el servidor.
*/
Control {

      Sistema s;

      Control (Sistema s) { this.s = s;}

      recibeEntrada(MensajeEntrada){}           // innecesario

      procesaMensajeEntrada(MensajeEntrada m) {
            // almacena mensaje entrada
            Hora t1 = System.timeIs();
            Emisor s1 = m.EmisorEs();
            int long1 = m.longIs();
            ...
```

```
                MensajeEntradaInterno m2 =
                        new MensajeEntradaInterno(t1,s1,long1,...);
                s.a.guarda(m2);

                // decide tipo proceso
                int i = m.accionEs();
                switch (i) {
                        case Constantes.ENVIA_CORREO_E:
                                s.mS[0].servicio(m);          // no espera por
                                                                        // respuesta!
                        break;
                        ...
                }
        }

        ...
    "Cualquier combinación de sistemas que pueden colaborar entre si para dar a los usuarios toda
la información que ellos necesiten sin que tengan que saber donde esta ubicada"
    }

    /*
        Modela el componente de permanencia ("persistence") del sistema.
        Reside en el servidor.
    */
    Almacen {

        Sistema s;

        Almacen (Sistema s) { this.s = s;}

        guarda(MensajeEntradaInterno) {...}          // almacena mensaje

        ...

    }

    /*
        Clase auxiliar; wrapper de todos los componentes y ayuda a intercomunicarlos.
        Reside en el servidor; se arranca antes de comenzar el proceso.
        Nota: en este diseño, la clase Receptor ya esta andando al arrancar el sistema
        en el servidor.
    */
    Sistema {

        Receptor r;
        Transporte t;
        Control c;
        Almacen a;
        Manipulador[] mS;

        Sistema() {
                r = conectaConReceptor(); // Receptor esta en el celular
                t = new Transporte(this);
                c = new Control(this);
                a = new Almacen(this);
                mS = new Manipulador[] {
                                new ManipuladorCorreo(this), ...
                        };
        }

        main() {
                ...
        }
    }
    /*
        Fin de diseño
    */
```

Referencias

[1]	Análisis, diseño e implementación de un sistema para envío de información de un servidor a una terminal remota. Abraham Valle Alvarez. Agosto 2008
[2]	Sistemas Remotos de Interacción con Internet. Benjamín Macías Pimentel . PAPIIT 2006
[3]	CUERPO SUPERIOR DE ADMINISTRADORES DE LA JUNTA DE ANDALUCIA. ESPECIALIDAD ADMINISTRADORES GENERALES (A.1100). TEMARIO. VOLUMEN II pag. 518. 2006
[4]	http://foldoc.org/index.cgi?daemon

Bibliografía

http://www.it.uc3m.es/mcfp/docencia/si/material/1_cli-ser_mcfp.pdf
http://es.wikipedia.org/wiki/Cliente-servidor
http://www.monografias.com/trabajos14/tipos-redes/tipos-redes.shtml
http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml
http://en.wikipedia.org/wiki/Short_message_service
http://www.wirelessdevnet.com/channels/sms/features/sms.html
http://en.wikipedia.org/wiki/Short_message_service
http://www.ietf.org/rfc/rfc2616.txt
http://es.wikipedia.org/wiki/Http
http://www.ietf.org/rfc/rfc959.txt
http://en.wikipedia.org/wiki/File_Transfer_Protocol
http://www.faqs.org/rfcs/rfc1939.html
http://tools.ietf.org/html/821#page-19
http://www.microsoft.com/technet/prodtechnol/exchange/ES/Guides/E2k3TransnRouting/12f765d7-2678-4491-8af8-2
http://java.sun.com/javame/technologies/index.jsp
http://www.lcc.uma.es/~galvez/J2ME.html
http://www.iec.org/online/tutorials/gsm/
http://es.wikipedia.org/wiki/Canal_de_comunicaciones
http://www.radioptica.com/Radio/telefonia_movil.asp
http://www.iec.org/online/tutorials/gsm/
http://www.omg.org/gettingstarted/what_is_uml.htm
http://jcp.org/aboutJava/communityprocess/final/jsr120/
http://www.ibm.com/developerworks/wireless/library/wi-extendj2me/
http://jcp.org/aboutJava/communityprocess/review/jsr082/index.html
http://developers.sun.com/mobility/midp/articles/bluetooth1/
http://www.ub.uib.no/elpub/2004/h/413009/Masteroppgave.pdf
http://archive.wirelesspronews.com/wirelesspronews-14-20041213IntroductiontoBluetoothandJ2ME.html
http://developer.yahoo.com/java/