INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

SISTEMA DE ASIGNACIÓN DE RECURSOS PARA ENTIDADES MÓVILES BASADO EN UNA INFRAESTRUCTURA DE TELEFONÍA CELULAR

T E S I S

QUE PARA OBTENER EL GRADO DE MAESTRA EN CIENCIAS PRESENTA:

ING. LAURA ALVAREZ ROBLES

DIRECTOR DE TESIS: M. EN C. ROLANDO MENCHACA MÉNDEZ

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
MÉXICO, D. F.
SEPTIEMBRE DE 2003

Doy gracias a Dios por darme la vida y permitirme alcanzar una meta más.

Agradezco infinitamente al Instituto Politécnico Nacional por brindarme una excelente preparación y por todos los apoyos otorgados.

A mis padres por su apoyo, cariño y dedicación que hicieron que éste camino fuera más fácil. A mi hermano por su ejemplo, amistad e invaluable ayuda. A toda mi familia y amigos por sus buenos deseos.

A Miguel por brindarme apoyo incondicional, por el cariño y motivación infinitos que me han servido de inspiración.

Al M. En C. Rolando Menchaca Méndez por su gran ayuda en el desarrollo de esta tesis.

Al Dr. Jesús Figueroa Nazuno por todo el apoyo recibido desde mi ingreso a este centro.

A mis sinodales Dr. Agustín Gutiérrez Tornés, Dr. Hugo César Coyote Estrada, M. En C. Sandra Dinora Orantes Jiménez y M. En C. Rubén Peredo Valderrama por las aportaciones que hicieron a este documento.

A todos los que de alguna forma contribuyeron en la realización de este trabajo.

Muchas gracias.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

Capítulo 5. Implementación y pruebas

RESUMEN

En esta tesis se presenta la arquitectura de un Sistema de Asignación de Recursos para Entidades Móviles basado en una Infraestructura de Telefonía Celular. La arquitectura se divide en tres capas: comunicación inalámbrica, AGR (Administrador Global de Recursos) y CR (Contenedores de Recursos). La arquitectura propuesta se basa en el modelo de funcionamiento WAP, el cual es una adaptación para dispositivos móviles de la arquitectura definida para el Web.

Los CR son servidores independientes que administran recursos localmente. Por lo tanto, los RDBMS que utilizan, así como su estructura de información pueden ser diferentes. Para lograr que el AGR pueda interactuar con múltiples CR sin necesidad de realizar adecuaciones a sus bases de datos, se diseñó un documento XML que contiene la información necesaria para que el AGR se comunique con los CR. La información que intercambian es de carácter privado y para lograr una transmisión segura se utiliza el protocolo SSL.

Una característica de los dispositivos móviles es que a diferencia de los sistemas de escritorio no manejan conexiones permanentes. Debido a esto, se implementó un mecanismo que guarda el estado de la petición hasta que el recurso se asigne. De tal forma que si ocurre una desconexión, cuando el usuario reingrese al sistema, este último le indica que tiene una petición pendiente con la opción de continuarla o cancelarla.

Para demostrar la funcionalidad del sistema, se planteó un escenario circunscrito al área del Sector Salud. En este escenario, el sistema administra las camas disponibles de los hospitales del Distrito Federal en cada especialidad. Las peticiones se hacen desde una ambulancia por medio de un teléfono celular obteniendo por respuesta el hospital que puede atender al paciente.

Algunas de las ventajas de un sistema de este tipo son:

- Acceso ubicuo a la información. Se puede acceder desde cualquier lugar.
- Puede llegar a un amplio espectro de personas porque su uso es simple e intuitivo.
- Es económico comparado con sistemas de escritorio.
- Indirectamente puede llegar a mejorar la calidad en la atención a los pacientes. Esto debido a que se reduce el tiempo en el que se canaliza un paciente a un hospital.

ABSTRACT

The architecture of an Assignment Resources System for Mobile Entities based on Cellular Telephony Infrastructure is presented in this work. The architecture has three layers: wireless communication, AGR (*Administrador Global de Recursos*, Global Resources Administrator) and CR (*Contenedor de Recursos*, Resources Containers). The system architecture is based on the WAP model, which is an adaptation of the Web architecture for wireless devices.

CR's are independent servers that manage resources locally. Thus, their information structure as well as the RDBMS they use to implement it can be different. In order to establish communication between the AGR and the heterogeneous CR's, we design an XML document which stores the all the information needed. Because this information is confidential, the SSL protocol is used to set up a secure communication channel.

One characteristic of mobile devices, unlike the desktop systems, is that they don't have fixed connections. Due this, we implement a mechanism that keeps the request status until the resource assignment is completed. If a disconnection occurs and when the user logs in back, the system notifies that there is a pending request with the option to continue or cancel it.

In order to demonstrate the system functionality, we implement a system prototype based on a common scenario in the Healthcare area. The system manages the available beds of the different Mexico City's hospitals. A user request comes from a cellular phone in an ambulance and the response is the hospital that can assist the patient.

Some of the advantages of a system like this are:

- Ubiquitous access information. The information can be accessed from any place.
- A wide spectrum of people can use it because is simple and intuitive.
- It is less expensive than desktop systems.
- Because the system decreases the time in which a patient is accepted in a hospital. The attention quality is improved.

INTRODUCCIÓN

El cómputo móvil ha adquirido una importancia creciente y se está extendiendo, a los sistemas de cómputo distribuido actuales. Aunque los estándares de red no se diseñaron con la capacidad de soportar la demanda de movilidad, se requiere cada vez más de un acceso continuo a la información, independiente del punto de enlace.

Debido a esto, las aplicaciones Web actuales enfrentan problemas que no habían sido considerados. Sistemas que se distribuyen a grandes distancias deben ejecutarse rápidamente y sin fallos. Datos de sistemas heterogéneos, bases de datos, servicios de directorio y aplicaciones, deben transferirse sin que se pierda información.

Aunado a esto, hay una gran diversidad de clientes, que pueden ser además de navegadores Web que soportan HTML, teléfonos móviles que soportan el Protocolo de Aplicación Inalámbrica (WAP, *Wireless Application Protocol*) u organizadores portátiles, con diferentes lenguajes de marcado (*markup*). Los datos y la transformación de éstos, se han vuelto el centro crucial de cada aplicación que se desarrolla en la actualidad.

Para solucionar estos problemas, se introdujeron nuevas tecnologías como el cómputo móvil, que permite la creación y utilización de nuevos servicios y aplicaciones, al proporcionar un ancho de banda capaz de soportar "casi" cualquier servicio. La velocidad de transmisión que se puede lograr, permite que se empiecen a contemplar como factibles una serie de aplicaciones (principalmente de tipo audiovisual y multimedia) que hace muy poco tiempo resultaban imposibles de implantar eficientemente, en las redes actuales.

PLANTEAMIENTO DEL PROBLEMA

Actualmente el uso de dispositivos móviles ha aumentado y existe la infraestructura necesaria para soportar cualquier servicio, por consiguiente, se pueden cubrir nuevas necesidades, como la asignación de recursos, cuyo problema consiste en que exista un conjunto finito de recursos agrupados en contenedores distribuidos sobre un área geográfica.

Por otra parte, existe un conjunto de entidades móviles que eventualmente requieren obtener uno o más recursos pertenecientes a cualquiera de estos contenedores. Una solución es que cada entidad móvil solicite el recurso al contenedor más cercano; si éste no lo puede otorgar, dirige su petición a otro contenedor. Esta actividad se ejecuta hasta que algún contenedor le otorgue el recurso deseado, procedimiento que consume mucho tiempo.

Por esta razón, es necesario implementar un mecanismo que permita administrar globalmente los recursos de cada contenedor, así, cada entidad móvil que requiera un recurso tiene que dirigir su petición a este mecanismo y éste, buscará en los contenedores disponibles, los que pueden otorgar el recurso.

La solución propuesta debe satisfacer dos requisitos: primero, la exclusión mutua, la cual, requiere que un mismo recurso no sea asignado a más de una entidad a la vez. Segundo, debe ser libre de inanición, es decir, que las entidades no esperen para siempre por los recursos[1].

[1] Bajo el supuesto de que eventualmente habrá los recursos suficientes para ser asignados a todas las entidades.

En la figura P-1, se ilustra cómo se realizará la asignación, en un caso donde se tienen varios Contenedores de Recursos (CR) distribuidos en un área geográfica, un Administrador Global de Recursos (AGR) y varias Entidades Móviles (EM) que requieren de uno o más recursos.

El esquema puede interpretarse de la siguiente manera:

1. Cada CR se registra en el AGR indicándole su nombre, los recursos que puede ofrecer y su disponibilidad.
2. Una EM requiere de un recurso, entonces hace la petición al AGR indicándole que tipo necesita.
3. El AGR le responde con el o los nombres de los CR que pueden ofrecer el recurso solicitado.
4. La EM le indica al AGR, de cual CR tomará el recurso.
5. El AGR se comunica con el CR para solicitárselo.

6, 7 y 8. El AGR enlaza a la EM con el CR para que pueda utilizarlo.

Para ilustrar la asignación de recursos se trabajará en el área del Sector Salud, ya que a partir de 1999, particularmente en el Distrito Federal y Área Metropolitana, han surgido diversos problemas al respecto. "Socorristas y paramédicos del ERUM y la Cruz Roja Mexicana coinciden en señalar que cada vez son más los casos en los que tienen que recorrer varios hospitales hasta que encuentren uno en el que puedan atender al paciente"[2]. De esta forma, se pierde tiempo valioso, que en el caso de lesiones graves disminuye la probabilidad de que una persona salve la vida y la utilización de la telefonía celular puede contribuir a reducir tiempos.

[2] Mario Torres. El Universal. Viernes 15 de octubre de 1999. Ciudad, página 8.

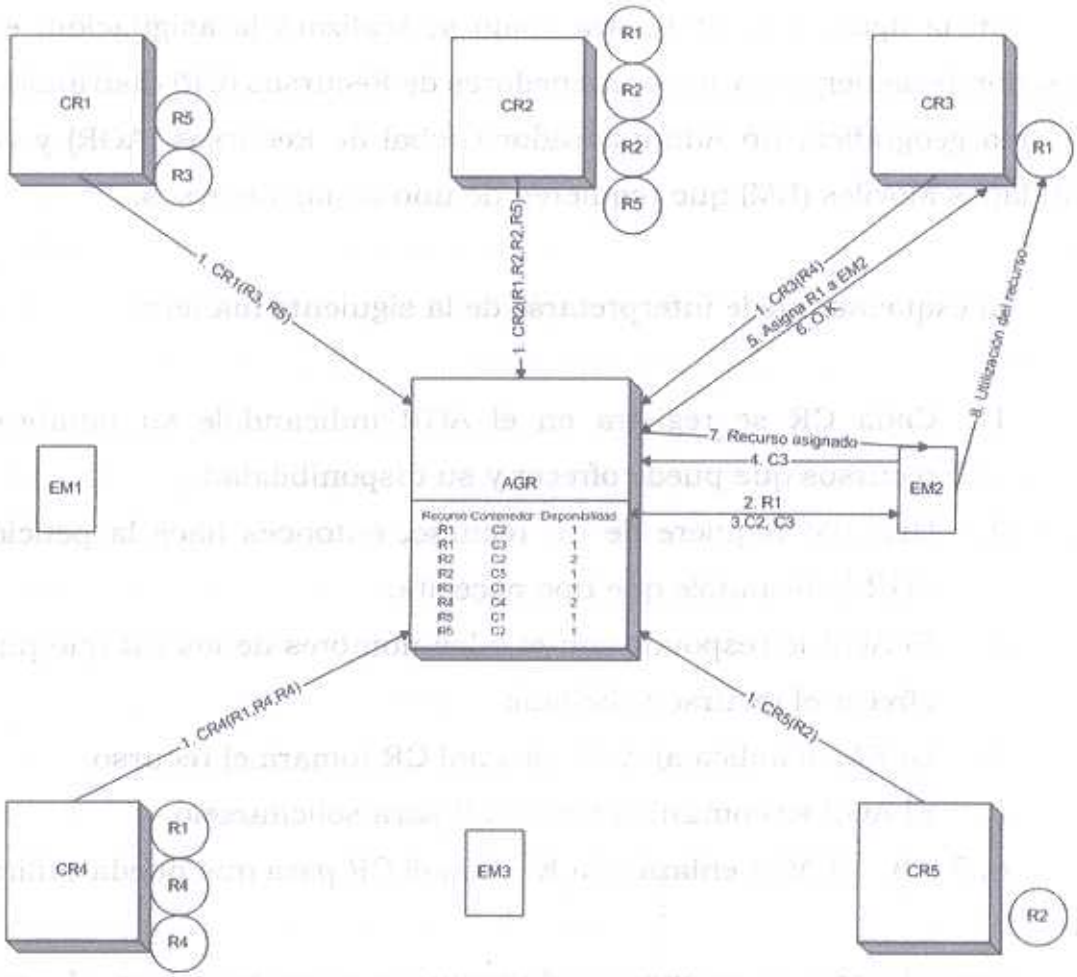

Figura P-1. Esquema de la asignación de recursos para entidades móviles.

Para realizar el sistema es necesario solucionar los siguientes problemas:

- El acceso a las Bases de Datos. Se proyecta que participen todas las instituciones de salud en el Distrito Federal; Instituto Mexicano del Seguro Social (IMSS), Instituto de Seguridad y Servicios Sociales de los Trabajadores del Estado (ISSSTE), Secretaría de Salubridad y Asistencia (SSA), Cruz Roja, Hospitales privados, y los que pertenecen al gobierno del Distrito Federal, lo cual, trae como consecuencia que las Bases de Datos tengan diseños diferentes y por lo tanto, se dificulta la

generalización y el acceso. Ligado a este problema, se encuentra el hecho de que existen variedades de Sistemas Manejadores de Bases de Datos Relacionales (RDBMS, *Relatinal Database Management System*) y que, las Bases de Datos participantes, se encuentran distribuidas físicamente, esto se soluciona, personalizando el acceso a cada una mediante la utilización de XML (*eXtensible Markup Language*, Lenguaje Extensible de Marcado); específicamente, el servidor de Base de Datos de cada hospital contendrá un Documento XML con la información necesaria para conocer la estructura de la Base de Datos y de esta forma, se podrán formular consultas SQL (*Structured Query Language*, Lenguaje de Consulta Estructurada) válidas para cada hospital, logrando enviar y obtener la información relevante para asignar y en su momento liberar, los recursos que cada uno ofrece. La razón por la cual se utilizará XML, es porque tiene la capacidad de integrar el sistema propuesto con los sistemas de software de los hospitales por medio de Internet logrando así, interoperabilidad sobre las plataformas, aplicaciones y lenguajes de programación. Las aplicaciones pueden definir una DTD (*Document Type Definition*, Definición de Tipo de Documento) en la que se incluya la información relevante a ésta, de tal forma que para accederlas, sólo se tienen que escribir clases que concuerden con ellas, sin tener que hacer modificaciones en los sistemas. Además, XML y Java, permiten que la interacción se haga de manera fácil, rápida y a un costo bajo, gracias al uso de Servlets, así con estas dos tecnologías, se puede crear una aplicación capaz de integrar todos los recursos remotos, para que puedan ser asignados a las entidades móviles.

- Con la solución planteada en el punto anterior, surge otro problema, la seguridad. El documento XML contiene información privada y debe asegurarse, que ésta sea obtenida sólo por personas autorizadas y que no pueda ser cambiada durante la transferencia. Para solucionarlo, se utiliza JSSE (*Java Secure Socket Extensión*, Extensión de Sockets Seguros de Java) que permite comunicaciones seguras por Internet y garantiza la encriptación de los datos y la autenticación, para restringir el acceso a entidades no autorizadas.

- Deben evitarse accesos innecesarios a las Bases de Datos de los hospitales, por consiguiente, se crea una Base de Datos intermedia, en la cual, se almacena información proveniente de las Bases de Datos de los hospitales y que es necesaria para realizar la asignación de recursos: datos generales del hospital, información sobre el RDBMS que utiliza y el diseño de la Base de Datos; también incluye una tabla con los recursos disponibles, el hospital que los ofrece y su disponibilidad, de esta forma, cuando un usuario ingrese al sistema, no se conectará con todos los servidores, sino solamente con aquellos que tengan mayor disponibilidad y por lo tanto, mayores probabilidades de éxito al buscar el recurso necesitado. Con esto se reduce el tiempo de respuesta del sistema.

- La utilización de dispositivos móviles, presenta algunas limitaciones y la más importante, es que los teléfonos celulares cuentan con pantallas reducidas y además, se dificulta la introducción de información via teclado; para disminuirla, se realizan interfaces por selección de opciones cuidando que se presente en la pantalla sólo información relevante.

- Para probar la aplicación se utilizan además de teléfonos celulares, el simulador que proporciona Openwave[3], debido a la gran variedad de teléfonos celulares existentes, donde cada dispositivo cuenta con capacidades y propiedades diferentes que afectan la forma en que despliegan información. Aunque los dispositivos WAP (*Wireless Applcation Protocol*, Protocolo de Aplicación Inalámbrica) se han diseñado para ser independientes y el código que se escribe para ellos debería poder ejecutarse de la misma forma en cualquier dispositivo, no siempre ocurre así; cada dispositivo implementa sus propiedades por lo que el entorno de desarrollo es inconstante. Por ello, resulta necesario probar el código escrito en la mayor cantidad de dispositivos posibles; por la dificultad para afrontar la compra de varios dispositivos WAP, se presenta la solución de utilizar simuladores, los cuales han sido diseñados para imitar el comportamiento y funcionalidad de los dispositivos móviles. WAP se utiliza, porque es el estándar de facto actual del cómputo móvil (en particular para aplicaciones de telefonía celular) y es un protocolo para el transporte de datos. Se ha diseñado para el mundo creciente de dispositivos móviles y su finalidad es adaptarse a las limitaciones propias de este tipo de cómputo. Además, prácticamente todos los teléfonos celulares cuentan con soporte WAP y solo algunos, con soporte Java; por esta razón, se trabajará con este protocolo del lado de los clientes (teléfonos celulares) y se utilizará Java y XML del lado del servidor.

[3] Openwave, www.openwave.com

JUSTIFICACIÓN

Con la amplia difusión de la telefonía celular y con el hecho de que el tráfico telefónico se va desplazando progresivamente desde el teléfono fijo al teléfono celular y los atributos del cómputo móvil como: omnipresencia, alcance, seguridad, conveniencia, localización, conectividad instantánea y personalización, permiten prever un importante incremento del uso de servicios sobre telefonía celular y el desarrollo de nuevas aplicaciones que complementen a las existentes. Debido a esto, resulta atractivo solucionar el problema de la asignación de recursos, para entidades móviles utilizando la infraestructura de telefonía celular existente, la cual, proporciona una eficiencia probada.

Por otra parte, la Secretaría de Salud del Distrito Federal, se encuentra preocupada por el incremento en los casos en que un paciente pasa horas en una ambulancia recorriendo hospitales hasta encontrar uno en el que lo puedan atender y ha hablado de la necesidad de articular un sistema efectivo de urgencias, para que los heridos sean canalizados en un tiempo no mayor a 10 minutos a una unidad médica.

Por esta razón, como tema de tesis se decide resolver este problema particular como caso de prueba del problema general: la asignación de recursos y así, contribuir reduciendo los tiempos de respuesta y evitar los rechazos de pacientes, empleando tecnología celular y permitiendo a cada ambulancia contar con un teléfono celular, por medio del cual, se conectará al sistema y una vez ingresando las características del paciente como: su padecimiento y el tipo de servicio médico con el que cuente. se le indicará, cual o cuales hospitales, cuentan con ese servicio en ese momento, con todo esto, también se reducen los costos de operación, al reducir los tiempos de arribo y traslado al lugar del accidente y a los hospitales.

OBJETIVOS

OBJETIVO GENERAL

- Implementar un sistema de asignación de recursos para entidades móviles basado en una infraestructura de telefonía celular.

OBJETIVOS ESPECÍFICOS

- Integrar las tecnologías de cómputo móvil y las tecnologías para la interoperabilidad de bases de datos.
- Implementar un mecanismo que permita integrar bases de datos heterogéneas.
- Realizar un DTD en XML que describa la estructura de la base de datos e información relacionada con las bases de datos de los contenedores de recursos.
- Implementar un mecanismo que permita un acceso seguro a los sistemas de información de los proveedores de recursos.
- Construir una interfaz intuitiva que se apegue a las restricciones de los dispositivos móviles.
- Generar una serie de recomendaciones relacionadas con la tecnología utilizada y el desarrollo de la aplicación.

ORGANIZACIÓN DE LA TESIS

El trabajo se organiza en 6 capítulos:

Capítulo 1. Tecnologías para Internet Móvil: donde se expone el estado del arte del cómputo móvil y las tecnologías involucradas en la realización de esta tesis.

Capítulo 2. Arquitectura: aquí se describe la arquitectura del sistema, la cual, se descompondrá en módulos, analizando cada uno de éstos con el fin de entender el comportamiento del sistema y proponer las tecnologías que se utilizarán en la implementación del sistema.

Capítulo 3. Análisis: se realiza una definición detallada del problema a resolver como caso de prueba. Se explican los módulos en los que se divide el sistema, así como la interacción entre ellos.

Capítulo 4. Diseño: se presenta la descomposición del sistema en subsistemas, se definen las clases que lo forman y se muestra, la arquitectura del caso de prueba.

Capítulo 5. Implementación y Pruebas: en este capítulo se presenta el desarrollo del sistema y los resultados de las pruebas a las que fue sometido.

Capítulo 6. Conclusiones y Trabajos futuros: donde se reportan las conclusiones obtenidas con la realización de esta tesis. También se describen algunos trabajos, que se pueden realizar basándose en la presente tesis.

Capítulo

1

Tecnologías para Internet Móvil

Resumen

En este capítulo se mencionan las principales tecnologías para la realización de aplicaciones en el área del cómputo móvil.

1.1. INTRODUCCIÓN

En este capítulo se mencionan las principales tecnologías para la realización de aplicaciones en el área del cómputo móvil. El capítulo se organiza de la siguiente manera:

Sección 1.2. Cómputo Móvil: donde se expone el estado del arte del cómputo móvil.

Sección 1.3. Sistema Global para comunicaciones Móviles: aquí se mencionan las principales características de esta tecnología.

Sección 1.4. Protocolo de Aplicación Inalámbrica: donde se explica el protocolo WAP, ya que este protocolo es la plataforma para aplicaciones de Internet a la que se puede acceder a través del teléfono móvil.

Sección 1.5. Lenguaje de Marcado Extensible: se explica el lenguaje XML. Este lenguaje es tan flexible que se puede utilizar para varias cosas. Se puede utilizar para describir metacontenidos respecto a documentos o recursos en línea, para publicar e intercambiar contenidos de bases de datos, y como formato para sistemas de mensajería con el fin de permitir la comunicación entre programas de aplicación. Además en esta sección, se menciona WML. Este lenguaje fue creado con XML y se utiliza para describir la estructura de los documentos que se distribuirán a través de dispositivos móviles. Se adapta a las limitaciones propias de los teléfonos celulares y otros dispositivos móviles.

Sección 1.6. Interconexión de Bases de datos: se describe la forma en la que se pueden interconectar Bases de Datos heterogéneas con XML. También se menciona JDBC, ya que ésta API de Java permite ejecutar sentencias SQL con cualquier tipo de Base de Datos por medio de un manejador JDBC.

Sección 1.7. Java Inalámbrico: aquí se menciona la tecnología para la creación de aplicaciones inalámbricas con Java para dispositivos pequeños como localizadores, teléfonos celulares, PDA, etc. Utilizando la plataforma *Java2 Micro Edition* (Edición Micro de Java 2).

Sección 1.8. Extensión de Socket Seguro de Java (JSSE): se explica el marco de Trabajo JSSE para desarrollar aplicaciones que requieran una comunicación segura entre clientes y servidores que utilicen cualquier protocolo de aplicación como HTTP.

Sección 1.9. Trabajos Relacionados: se mencionan las características de algunos sistemas de cómputo móvil existentes y que tienen alguna relación con esta tesis.

Sección 1.10 Conclusiones: después de haber estudiado las tecnologías existentes para la creación de aplicaciones para dispositivos móviles se mencionará cuales se van a utilizar para la realización de esta tesis y de que manera se van a relacionar.

Sección 1.11 Referencias: aquí se enumera el material consultado para la realización de éste capítulo.

1.2. CÓMPUTO MÓVIL [1], [2], [3]

El término cómputo móvil se usa para describir una serie de tecnologías que soportan movilidad personal. Tal movilidad puede manifestarse con usuarios moviéndose entre terminales fijas en cualquier parte del mundo o usuarios que utilizan dispositivos móviles en cualquier lugar. En ambos casos, se debe proporcionar al usuario un ambiente de trabajo consistente con el acceso a sus archivos, correo electrónico, etc. La movilidad debe apoyar por consiguiente el movimiento de las personas, datos, y/o aplicaciones entre diferentes lugares.

Las tecnologías requeridas son una combinación de varios medios de comunicación, servidores de datos, y computadoras (portátiles y/o de escritorio). Las comunicaciones móviles presentan las siguientes características:

- Movilidad
- Posicionamiento
- Personalización
- Seguridad
- Comunicaciones por voz

Fundamentalmente, el cómputo móvil tiene como meta proporcionar un ambiente totalmente omnipresente donde las personas pueden trabajar donde sea, y a cualquier hora. Esto va mas allá de convertir en móviles los servicios actuales. Aunque el cómputo móvil seguirá en un futuro previsible marcado por las limitaciones que plantean el ancho de banda y las pantallas de las terminales, presenta ventajas específicas de este entorno, por ejemplo, la personalización, y la dependencia del lugar.

En el desarrollo del cómputo móvil, los servicios están pasando por tres tendencias sucesivas y, en algunos casos, superpuestas. A continuación se mencionan estas tendencias:

- Acceso a los contenidos que existen actualmente en Internet para su uso desde un dispositivo móvil.
- Aplicaciones móviles específicas, como WAP.
- Personalización de información y contenidos de acuerdo al lugar donde se encuentre el usuario.

La primera tendencia, empezó con los servicios de mensajes cortos (SMS, *Short Message Service*) y la transmisión de datos. Un ejemplo de esto es el uso de computadoras portátiles junto con un teléfono móvil para enviar y recibir mensajes de correo electrónico o acceder a intranets. Los usuarios pueden tener acceso a los servicios de Internet en el aeropuerto, en el tren o en el parque. Los principales requisitos de infraestructura en este sentido consisten en transferir los datos a alta velocidad de una manera eficaz y en formato comprimido para el usuario. Los servicios de la segunda etapa han presentado problemas técnicos para el desarrollo masivo de éstos al tener una latencia excesiva y un ancho de banda limitado para cada comunicación (9600 bps máximo para transmisión de datos sobre el sistema global para comunicaciones móviles (GSM, *Global System for Mobile comunication*) y 160 bytes para cada transacción por SMS).

La segunda tendencia, es la que lleva los servicios de Internet a los dispositivos móviles. Las aplicaciones están especialmente adaptadas para funcionar en dispositivos móviles con pantallas pequeñas, por ejemplo, con el WAP utilizando las redes actuales GSM/GPRS (*General Packet Access Service,* Servicio General de Acceso de Paquetes), un avance de lo que son las de tercera generación UMTS (*Universal Mobile Telecomunications System,* Sistema Universal de Telecomunicaciones Móviles), que permitirán aplicaciones multimedia en el teléfono móvil.

La tercera tendencia permitirá obtener el máximo aprovechamiento del cómputo móvil. Las aplicaciones, los contenidos y los servicios pondrán de manifiesto todo su potencial debido a la movilidad, la ubicación y la preferencia del usuario, con lo que su experiencia pasará a ser "situacional". Esta posibilidad permitirá crear servicios más valiosos y personalizados. Por ejemplo, el servicio puede estar basado en la información sobre su ubicación actual, los datos de la agenda diaria y sus preferencias personales.

1.3. SISTEMA GLOBAL PARA COMUNICACIONES MOVILES (GSM, *GLOBAL SYSTEM FOR MOBILE COMUNICATIONS*) [4]

GSM es un sistema digital celular desarrollado por CEPT (*Groupe Special Mobile of Conference European des Postes et Telecomunications*) y su sucesor ETSI (*European Telecomunications Standard Institute*). Uno de los objetivos más importantes del proceso de desarrollo GSM fue ofrecer compatibilidad de servicios celulares entre países Europeos. GSM es una tecnología revolucionaría que combina TDMA (Time División Múltiple Access) y FDMA (*Frecuency División Múltiple Access*) Con TDMA, la radiobase puede ser compartida entre múltiples usuarios. En GSM, un portador de frecuencia se divide en ocho rebanadas de tiempo donde la tasa de codificación de la transferencia es de 13 kbps. En una estación base GSM, cada par de radio transceptor-receptor soporta ocho canales de voz, mientras que una estación base AMPS necesita un par de radio para cada canal de voz. La interfaz de aire GSM ha evolucionado en EDGE (*Enhanced Data Rate for GSM Evolution*) con una tasa variable de datos y adaptación de enlace. EDGE utiliza modulación altamente eficiente de espectro para tasas de bits más altas que las existentes en la tecnología GSM. Los requerimientos básicos de GSM se describen en cinco aspectos:

Servicios. El sistema proporciona portabilidad de servicio; esto es, que las estaciones móviles (MSs) o los teléfonos móviles pueden utilizarse en todos los países participantes. El sistema ofrece servicios que existen en las redes fijas, además de los servicios específicos para las comunicaciones móviles.

Servicios de calidad y seguridad. La calidad de la voz telefónica GSM es al menos tan buena como la que ofrecen los sistemas análogos sobre el rango operativo práctico. El sistema será capaz de ofrecer cifrado en la información sin afectar significativamente el costo para los usuarios que no requieren esta característica.

Utilización de radio frecuencia. El sistema permite un alto aprovechamiento del espectro. El sistema es capaz de operar en la banda de frecuencia asignada, y coexistir con los sistemas cercanos en la misma banda de frecuencia.

Red. Las redes públicas fijas existentes no son modificadas significantemente.

1.4. PROTOCOLO DE APLICACIÓN INALÁMBRICA [5]

Para adaptar contenidos web estándar a los usuarios inalámbricos, se utiliza el protocolo WAP, el cual se diseñó para lograr la interoperabilidad de diferentes redes inalámbricas, dispositivos, y aplicaciones que usan un conjunto común de aplicaciones y protocolos. Usando la arquitectura WAP, puede desplegarse software inalámbrico intermedio como un cliente en una terminal móvil y como un servidor en el *gateway* o servidor intermedio (figura 1-1).

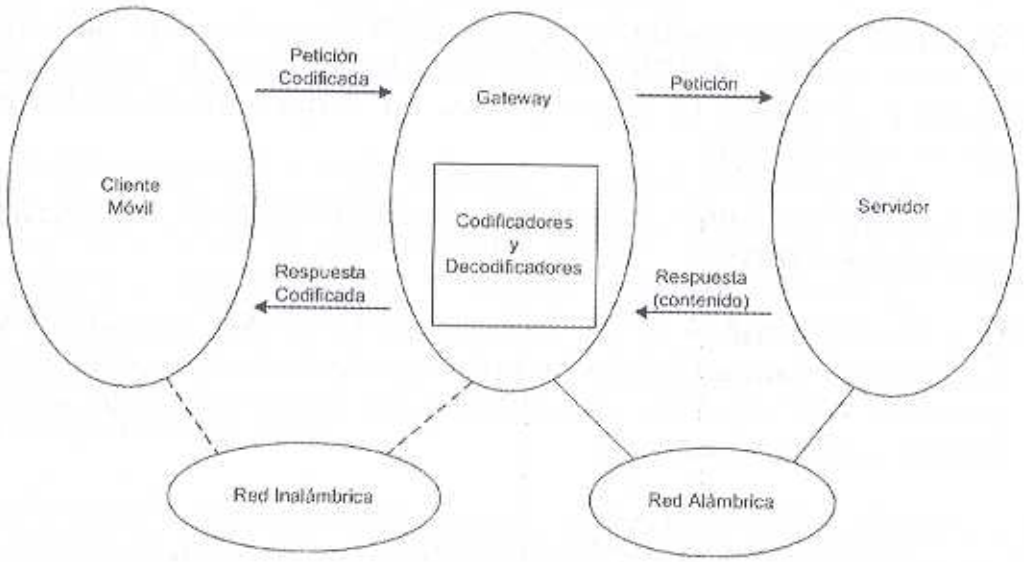

Figura 1-1. Modelo de programación WAP.

WAP usa un micro navegador en el cliente que soporta texto, gráficos, y contenido web estándar. Soportará medios de comunicación por flujos como video en el futuro cercano. En WAP, un *gateway* actúa como un servidor *proxy* para el cliente móvil y traduce las peticiones de la pila de protocolos WAP a la pila de protocolos empleadas por el servidor de información en el otro lado. Los codificadores traducen el contenido que viene del servidor en formatos compactos para reducir el tamaño de los datos sobre la red inalámbrica. Esta infraestructura asegura que el usuario móvil pueda acceder a una amplia variedad de contenidos y aplicaciones y también les permite a los desarrolladores construir aplicaciones de comercio electrónico (usando tecnologías existentes y probadas) que pueden ejecutarse en diversas terminales móviles.

Los pasos que se siguen en una conexión con el protocolo WAP son los siguientes (figura 1-2):

1. El usuario selecciona en su terminal WAP una dirección (URL).

2. La terminal móvil WAP manda esta petición del URL al *Gateway* WAP utilizando el protocolo WAP.

3. El *gateway* convierte esta petición WML (*Wireless Markup Language*, Lenguaje de Marcado Inalámbrico) y/o WMLScript WAP a WML y/o WMLScript HTTP convencional y manda la petición de búsqueda del URL al servidor web.

4. El servidor Web evalúa la petición HTTP y determina que tipo de petición es. Si el URL es un simple archivo, le adjuntará un encabezado HTTP. Si la petición es un *Script* o un CGI el servidor abrirá la aplicación.

5. El servidor Web devuelve una respuesta en WML, o WMLScript con encabezado HTTP.

6. El *gateway* verifica el encabezado HTTP y los contenidos WML, convierte en forma binaria los scripts creando una respuesta para la terminal WAP en WML y/o WMLScript con encabezado WAP y lo manda a la terminal móvil.

7. La terminal WAP recibe la respuesta con el protocolo WAP y evalua los contenidos WML y/o WMLScript y los visualiza según la configuración de la terminal.

El W3C diseñó varias recomendaciones en los intereses de independencia de dispositivos Web, reutilización de código, y codificación de red amigable. Estas recomendaciones incluyen el Lenguaje de Marcado Extensible (XML, *Extensible Markup Language*) para información semántica más rica, mejoró las Hojas de Estilo en Cascada (CSS, *Cascading Style Sheet*) y el Lenguaje de Hoja de Estilo Extensible (XSL, *Extensible Stylesheet Language*) para impulsar la separación del contenido y la presentación, y un Modelo de Objetos de Documento (DOM, *Document Object Model*) qué define una API independiente del lenguaje esas aplicaciones pueden utilizar el acceso y modificar la estructura, contenido, y estilo de documentos HTML y XML. Estas especificaciones, junto con las especificaciones WAP, habilitan un amplio rango de aplicaciones inalámbricas.

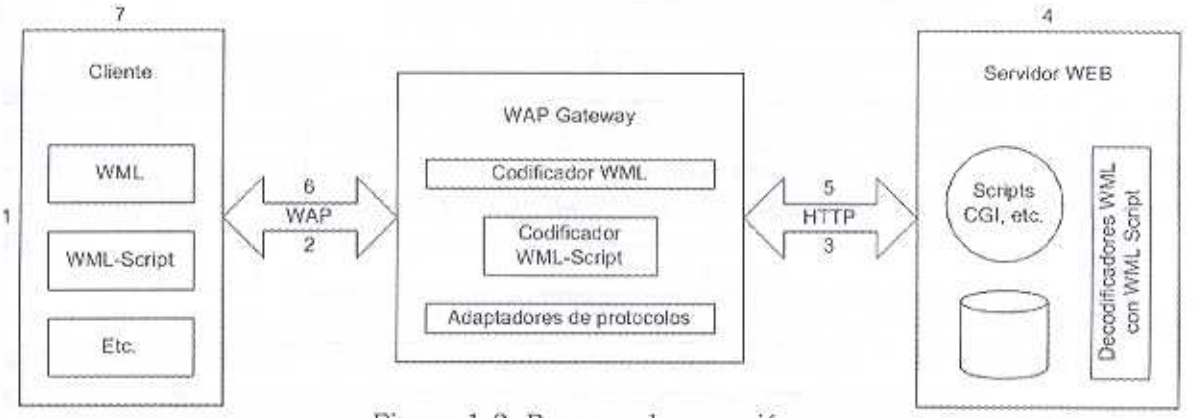

Figura 1-2. Pasos en la conexión.

WAP proporciona seguridad a las aplicaciones a través del protocolo de seguridad WTLS (*Wireless Transaction Layer Secure*, Capa de Transacciones Seguras Inalámbricas) equivalente al SSL (*Secure Socket Layer*, Capa de Socket Seguro) de Internet. WAP es además independiente del mecanismo de transporte y soporta tanto circuitos como GPRS o SMS.

1.4.1. Arquitectura WAP

La arquitectura WAP proporciona un ambiente extensible y escalable para el desarrollo de aplicaciones para dispositivos de comunicación móvil. Esto se alcanza por medio de un diseño a capas de toda la pila de protocolo (figura 1-3). Cada una de las capas de la arquitectura es accesible por las capas superiores, así como también por otros servicios y aplicaciones.

La arquitectura de capas de WAP permite a otros servicios y aplicaciones utilizar las características de la pila WAP a través de un conjunto de interfaces bien definidas. Las aplicaciones externas pueden acceder a las capas de sesión, transacción, seguridad y transporte directamente.

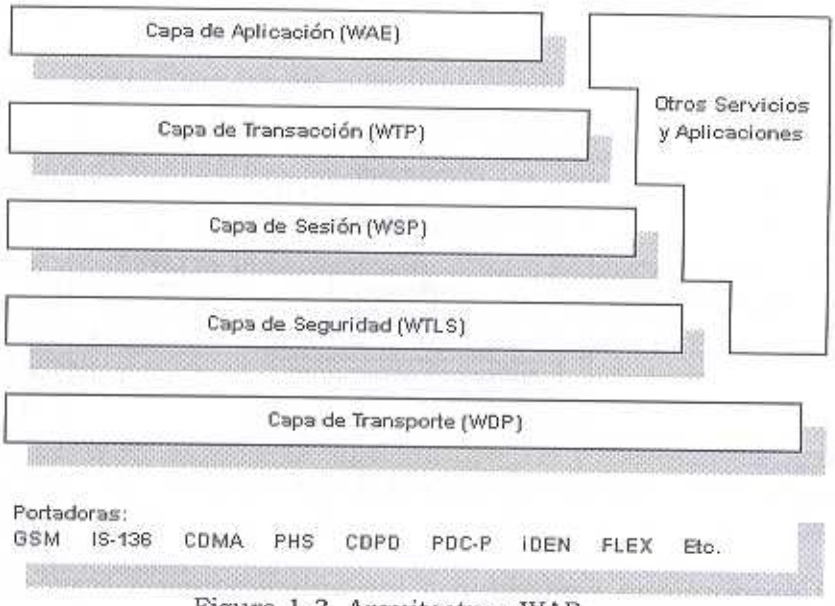

Figura 1-3. Arquitectura WAP.

1.4.1.1. Ambiente de Aplicación Inalámbrica (WAE, *Wireless Application Environment*)

El ambiente de aplicación inalámbrica (WAE) es un ambiente de aplicación de propósito general basado en una combinación de las tecnologías WWW y telefonía móvil. Su objetivo es establecer un ambiente interoperable que permita a los operadores y proveedores de servicio construir aplicaciones y servicios que puedan alcanzar una amplia variedad de plataformas inalámbricas de una forma eficiente y útil. WAE incluye un ambiente de micro-navegador que contiene las siguientes funcionalidades:

- **Lenguaje de Marcado Inalámbrico** (WML, *Wireless Markup Language*). Un lenguaje de marcado ligero, similar a HTML, pero optimizado para su uso en terminales móviles.
- ***WMLScript***. Un lenguaje de guión (*script*), similar a *JavaScript*.
- **Aplicación de Telefonía Inalámbrica** (WTA, *Wireless Telephony Application*). Servicios de telefonía e interfaces de programación.

1.4.1.2. Protocolo de sesión inalámbrica (WSP, *Wireless Session Protocol*)

El protocolo de sesión inalámbrica (WSP) proporciona la capa de aplicación del WAP con una interfaz consistente para dos servicios de sesión. El primero es un servicio orientado a conexión que opera sobre la capa del protocolo de transacción WTP. El segundo es un servicio sin conexión que opera sobre un servicio de datagramas seguro o inseguro (WDP).

El protocolo de sesión inalámbrica actualmente consiste de servicios apropiados para visualizar aplicaciones (WSP/B). Proporciona las siguientes funcionalidades:

- Funcionalidad y semántica HTTP/1.1 en una codificación compacta.
- Estado de sesión de larga duración.
- Suspensión y actualización con migración de sesión.
- Una facilidad común para almacenar (*push*) datos confiables y no confiables.
- Protocolo de negociación.

Los protocolos en la familia WSP están optimizados para redes portadoras de bajo ancho de banda con una latencia relativamente larga. WSP/B esta diseñado para permitir a un *proxy* WAP conectarse a un cliente WSP/B que entienda un servidor HTTP.

1.4.1.3. Protocolo de Transacción Inalámbrica (WTP, *Wireless Transaction Protocol*)

El protocolo de Transacción Inalámbrica (WTP) se ejecuta en la parte superior de un servicio de datagramas y proporciona un protocolo ligero orientado a conexión que se adapta para su instalación en clientes pequeños (estaciones móviles). WTP opera eficientemente sobre redes de datagramas seguras o inseguras y proporciona las siguientes características:

- Tres clases de servicios de transacción:
  - Peticiones no confiables en un sentido.
  - Peticiones confiables en un sentido.
  - Transacciones confiables en dos sentidos Petición-Respuesta.
- Confiabilidad opcional usuario-usuario: WTP usuario dispara la confirmación de cada mensaje recibido.
- Reconocimientos opcionales de datos fuera de banda.

- Concatenación y reconocimientos retardados de PDU para reducir el número de mensajes enviados.
- Transacciones asíncronas.

1.4.1.4. Capa de Transporte Seguro Inalámbrico (WTLS, *Wireless Transport Layer Security*)

WTLS es un protocolo de seguridad basado en el estándar del protocolo de Capa de Transporte Seguro (TLS), anteriormente conocido como Capa de *Sockets* Seguros (SSL). Se pretende que WTLS se utilice con los protocolos de transporte WAP y ha sido optimizado para su uso sobre canales de comunicación de banda estrecha. WTLS proporciona las siguientes características:

- **Integridad de datos.** WTLS contiene facilidades para permitir que los datos se envíen entre terminales y un servidor de aplicación es incorruptible y nunca cambia.
- **Privacidad.** WTLS contiene facilidades que permiten que los datos se transmitan entre terminales y un servidor de aplicación es privado y no puede entenderse con ninguna parte intermedia que pudiera haber interceptado el flujo de datos.
- **Autenticación.** WTLS contiene facilidades para establecer la autenticidad de las terminales y el servidor de aplicación.
- **Protección de negativa de servicio.** WTLS contiene facilidades para detectar y rechazar los datos que vuelven a enviarse o que no se verificaron satisfactoriamente. WTLS efectúa muchas negativas de servicio típicas.

WTLS puede además utilizarse para comunicaciones seguras entre terminales, por ejemplo, autenticación de tarjetas electrónicas de intercambio de negocios.

Las aplicaciones son capaces de habilitar o deshabilitar selectivamente las características WTLS dependiendo de sus requerimientos de seguridad y las características de las redes subyacentes.

1.4.1.5. Protocolo de Datagrama Inalámbrico (WDP, *Wireless Datagram Protocol*)

El protocolo de la capa de transporte en la arquitectura WAP se refiere como el Protocolo de Datagrama Inalámbrico (WDP). La capa WDP opera sobre los servicios de portadora aptos para datos que son soportados por varios tipos de redes. Como un transporte de servicio general, WDP ofrece un servicio consistente a los protocolos de capas superiores de WAP y comunica transparentemente sobre uno de los servicios de portadora disponibles.

Debido a que los protocolos WDP proporcionan una interfaz común a los protocolos de capas superiores, las capas de Seguridad, Sesión, y Aplicación son capaces de funcionar independientemente de las redes inalámbricas subyacentes. Esto se realiza para adaptar la capa de Transporte a las características específicas de la portadora subyacente. Manteniendo la interfaz de la capa de Transporte y las características básicas consistentes, la interoperabilidad global puede alcanzarse utilizando *gateways* intermedios.

1.4.1.6. Portadoras

Los protocolos WAP están diseñados para operar sobre una variedad de diferentes servicios de portadoras, incluyendo mensajes cortos, circuitos de conmutación de datos, y paquetes de datos. Las portadoras ofrecen diferentes niveles de calidad de servicio con respecto al rendimiento, tasa de error, y retardos. Los protocolos WAP están diseñados para compensar o para tolerar estos niveles de variación del servicio.

Debido a que la capa WDP proporciona la convergencia entre los servicios de portadora y el resto de la pila WAP, la especificación WDP lista las portadoras que son soportadas y las técnicas utilizadas para permitir a los protocolos WAP ejecutarse sobre cada portadora

1.5. JAVA INÁLAMBRICO [6]

J2ME (*Java 2 Micro Edition*, Edición Micro de Java 2) tiene la característica de tener una parte de su API fija, es decir, aplicable a todos los dispositivos inalámbricos y una parte que es específica para ciertos dispositivos; por ejemplo la API especifica de Palm y la de los teléfonos móviles evidentemente son distintas.

Una configuración es un conjunto mínimo de APIs que son útiles para desarrollar aplicaciones para un conjunto definido de dispositivos. Son muy importantes porque describen las funcionalidades más importantes requeridas para unos dispositivos determinados.

1.5.1. Configuración de Dispositivos de Conexión Limitada [7]

Hay una configuración estándar para dispositivos inalámbricos que se conoce como CLDC (*Connected Limited Device Configuration*, Configuración de Dispositivos de Conexión Limitada). Este estándar describe el conjunto de funcionalidades mínimas de los dispositivos inalámbricos, acorde a su potencia y a sus características. CLDC es en resumen el conjunto de APIs básicas para construir aplicaciones para dispositivos móviles.

CLDC es un estándar que Sun ha especificado. Según se aplique J2ME a nuevas familias de dispositivos, Sun especificará nuevos estándares adecuados a cada familia.

Extendiendo un poco el concepto de CLDC, este estándar especifica los siguientes aspectos de la programación inalámbrica:

- El subconjunto del lenguaje java que puede ser usado.
- El subconjunto de funciones de la Máquina Virtual Java.
- Las APIs fundamentales para esté tipo de desarrollo.
- Los requerimientos de hardware de los dispositivos móviles enfocados a CLDC.

Algunas características de Java han sido deshabilitadas y la razón es sencilla: los dispositivos móviles tienen capacidad limitada.
La función más importante que cumple CLDC es indicar un conjunto de APIs que debe incorporar cualquier dispositivo.

Finalmente, CLDC establece los mínimos de hardware requeridos para J2ME. Estos son los siguientes:

- 160Kb de memoria disponible para Java
- Procesador de 16-bit
- Bajo consumo energético
- Conexión a una red (a menudo de 9600 bps pero puede ser menor el ancho de banda)

Los dispositivos aptos para CLDC son teléfonos móviles, PDAs, ciertos electrodomésticos, etc. Muchos otros se irán incorporando al

estándar CLDC siempre y cuando cumplan los mínimos marcados por el estándar e incorporen la máquina virtual.

Examinando estos requerimientos más a fondo, es conveniente saber qué significa el requerimiento de 160kb de memoria disponible para java. Esta cantidad está compuesta de 128kb de memoria no volátil para la Máquina Virtual Java y las APIs de CLDC y otros 32kb de memoria volátil para el sistema Java *runtime*.

1.5.1.1 Limitaciones de CLDC

La primera limitación es la falta de soporte para los números de punto flotante, de manera que no ofrece soporte para este tipo de operaciones matemáticas. La limitación se impone porque los dispositivos carecen de hardware para estas operaciones y hacerlo vía software sería cargarlos por encima de sus posibilidades.

La siguiente restricción es la eliminación del método Object.finalize(). Este método es llamado para que un objeto sea borrado de memoria y CLDC no lo soporta ni lo requiere.

Una restricción a tener en cuenta es la limitada capacidad de CLDC para el manejo de excepciones. Esto es debido a que gran parte del manejo de excepciones depende exclusivamente de las APIs de cada dispositivo en concreto, pues las excepciones dependen de las características de cada aparato. Por ello CLDC maneja un número limitado de excepciones y delega el resto del manejo de excepciones en las APIs específicas de cada familia de dispositivos.

1.5.1.2 Seguridad en CLDC

Al igual que la versión estándar de Java, J2ME tiene su propio modelo de seguridad. Y es el modelo *sandbox* de seguridad.

La filosofía de este modelo de seguridad es dotar a los desarrolladores de grandes posibilidades para hacer aplicaciones potentes, pero por otra parte minimizar los riesgos que supone un dispositivo que ejecuta aplicaciones que se descargan de la red. Esto define los siguientes puntos:

- Las clases Java deben verificarse como aplicaciones Java válidas.
- Sólo se permite el uso de APIs autorizadas por CLDC.

- No esta permitido cargar clases definidas por el usuario.
- Sólo características nativas que entren dentro del CLDC pueden ser accedidas.

Por lo general, las restricciones impuestas por CLDC son restricciones de bajo nivel y esto es así porque CLDC trabaja a este nivel. Se supone que una capa adicional, que está definida en una configuración, impondrá nuevas restricciones.

1.5.2. Perfil Móvil de Información de Dispositivo [8]

Las configuraciones dentro de la arquitectura J2ME son un conjunto de APIs pensadas para un tipo concreto de dispositivos. De hecho, son una descripción de una familia de dispositivos que añade un conjunto de APIs adicionales al CLDC que se corresponden con las funcionalidades específicas de estos dispositivos.

CLDC es la configuración básica de J2ME. MIDP (*Mobile Information Device Profile*, Perfil Móvil de Información de Dispositivo) lleva CLDC más allá y añade nuevos requerimientos y APIs obligatorios para dispositivos MIDP. Los requerimientos de memoria de MIDP son:

- 128kb de memoria no volátil para las librerías MIDP API.
- 32kb de memoria volátil para el sistema Java runtime.
- 8kb de memoria no volátil para datos de aplicación persistente.

Respecto a CLDC, MIDP sólo incrementa los 8kb destinados a datos persistentes.

Los requerimientos de entrada para MIDP exigen la existencia de un teclado o una pantalla táctil o ambos. No se exige ratón (raro en un teléfono móvil).

Los requerimientos de salida para MIDP son algo más importantes, porque la pantalla es una de las restricciones mayores de los dispositivos móviles. MIDP exige al menos una pantalla de 97 x 54 píxeles (anchura, altura) con 1-bit de profundidad de color (blanco y negro). El ratio de salida debe ser 1:1.

MIDP tiene también requerimientos de red. El mínimo soporte de red exigido es disponer de una conexión inalámbrica de 2 sentidos. Se supone que estos dispositivos pueden tener un ancho de bando limitado (9600bps).

MIDP no establece ninguna obligación respecto a qué sistema operativo debe tener el dispositivo. Esto es posible gracias a que Java es multiplataforma. Sin embargo se esperan ciertos mínimos:

- Un núcleo mínimo que maneje el hardware a bajo nivel.
- Un mecanismo que lea y escriba en memoria persistente o no volátil.
- Un mecanismo de temporización para establecer mediciones temporales y dotar de información de tiempo a datos persistentes.
- Acceso de lectura y escritura hacia la conexión inalámbrica.
- Acceso a la entrada por teclado o pantalla.
- Soporte mínimo para mapas de bits.
- Un mecanismo que controle el ciclo de vida de una aplicación.

1.6. LENGUAJE DE MARCADO EXTENSIBLE [9]

XML fue concebido como solución definitiva a la situación creada por varios años de etiquetas HTML y marcado de presentación de páginas web. XML dota de estructura a un área compuesta tanto por software de contenido como por software de desarrollo. Al hacerlo, XML amplía las posibilidades de ese software mediante saltos y enlaces.

XML es una especificación para diseñar lenguajes de marcado. En otras palabras, XML es un metalenguaje que se usa para describir lenguajes de marcado como HTML. HTML en su forma actual no constituye un lenguaje de marcado XML, aunque se está tratando de modificar HTML para que se ajuste a los límites de XML.

Con XML no se puedan crear lenguajes de marcado, sino que se pueden crear lenguajes de marcado muy estructurados. XML, en realidad, no es más que un formato de texto estandarizado que sirve para representar información estructurada en la Web. Es así de sencillo. Sin embargo, cuando se dota de estructura a un almacén de información masivo como es la Web, todo es posible. Se pueden lograr nuevos niveles de automatización, mientras los datos estructurados van y vienen entre las distintas aplicaciones de la Web. Los motores de búsqueda se vuelven inteligentes de repente y pueden buscar en base al contexto y no sólo en base al contenido. Básicamente, los datos de la Web se describen a sí mismos.

XML es un subconjunto simplificado de SGML (*Standard Generalized Markup Language*, Lenguaje de Marcado Generalizado Estándar) que incorpora muchas de las características de SGML, entre las que se incluyen las tres más importantes: extensibilidad, estructura y validación.

Uno de los principales objetivos de XML consiste en separar el contenido (los datos) de la presentación (cómo se ven los datos) en los documentos web.

XML es una tecnología tan general que parece que sirve para todo. Sin embargo, a diferencia de HTML, XML no es una solución por sí mismo. XML define un marco que se puede usar para crear soluciones, pero aisladamente no hace mucho. Dado que la premisa que subyace a XML es la creación de conjuntos personalizados de etiquetas por medio de una Definición de Tipo de Documento (DTD, *Document Type Definition*) que sirven para codificar tipos específicos de información, no hay un visor XML genérico en el mismo sentido que un navegador web es un visor HTML. Si, hay XML genéricos, pero sólo le permiten ver la estructura del marcado XML y no la visualización de la verdadera significación del contenido XML.

Para ver los datos XML en un contexto significativo, tendrá que describir de cierta forma cómo hay que presentarlo. Hay que recordar que XML trata sobre contenido estructurado y no sobre despliegue. El despliegue de documentos XML se suele llevar a cabo con hojas de estilos, utilizando bien XSL o CSS.

Además de estas útiles capacidades de transformación, los mismos documentos XML y su contenido de datos pueden usarse para transferir información entre aplicaciones. Esta comunicación se logra fácilmente porque los datos XML no se atan a algún tipo de cliente, o incluso a ser usados por un cliente. Además proporciona una representación simple de los datos los cuales son fácilmente trasmisibles sobre una red. Esta característica de comunicación de XML es probablemente la más descuidada y subestimada de los documentos y representaciones de datos XML.

También hay que considerar que XML es una representación completamente textual de datos, es decir que XML traduce los documentos a un código comprensible por la computadora (representación digital). Porque el texto es semejante a una representación de datos ligera y fácilmente serializada, XML proporciona un medio rápido de transmitir datos por una red. Aunque algunos formatos de datos binarios pueden transmitirse eficazmente, las transmisiones de red textuales típicamente son medios más rápidos de comunicación.

1.6.1. Estructura lógica

Un documento se puede dividir en componentes a los cuales se les denomina elementos y cada elemento representa un componente lógico del documento. Los elementos pueden tener otros elementos, frases o palabras que normalmente se consideran el texto del documento o datos de carácter.

La estructura lógica se denomina estructura en árbol del documento. El elemento que contiene a los demás recibe el nombre de elemento raíz o elemento de documento, los elementos que incluye se llaman subelementos, que pueden contener a su vez subelementos. Si es así se denominan ramas, de lo contrario se llaman hojas.

A veces, los elementos incluyen información adicional llamada atributos. Los atributos describen las propiedades de los elementos.

1.6.2. Estructura física

El documento XML se define por una serie de caracteres. XML incluye un mecanismo que permite al texto organizarse de forma no lineal y en varias partes. El analizador sintáctico lo reorganiza en una estructura lineal.

Las entidades tienen nombres. Para poder utilizar una entidad, es necesario insertar una referencia de entidad en alguna parte del documento. Así, el procesador podrá reemplazar la referencia de entidad por la entidad misma, que se denomina texto de reemplazo.

Al igual que los elementos XML describen la estructura lógica del documento, las entidades siguen la pista de la ubicación de los conjuntos de bytes que constituyen el documento XML. A esto se le denomina estructura física del documento.

Los objetos que no son XML se consideran elementos de datos (GIF, JPEG, MPEG, PDF, etc.) porque carecen de cualquier tipo de marcado XML.

1.6.3. Documentos

Un documento XML debe estar bien establecido para poder analizarse y utilizarse correctamente. Un documento bien formado es aquel que para cada etiqueta abierta tiene una que cierra, y no hay ninguna fuera de orden y es sintáticamente correcto respecto a la especificación. La buena formación se centra en su estructura física. Algunas restricciones que determinan la buena formación de un documento se refiere a lo siguiente:

- Los nombres de atributos deben ser únicos dentro de un elemento.
- Los valores de atributos no deben contener el carácter "<".

Un documento válido es aquel que se adapte a su DTD. Una DTD define que etiquetas pueden ir en un documento, en que secuencia, sus atributos y opcionalmente los valores que pueden tener estos atributos. Para que se compruebe la validez de un documento, éste debe incluir la declaración <!DOCTYPE> al principio, donde viene especificada la DTD a la que debe adecuarse el documento para que resulte válido. La validez de un documento tiene que ver con la estructura lógica de los elementos.

Debe tenerse en cuenta que un documento válido es siempre un documento bien formado, pero un documento bien formado no siempre es un documento válido.

1.6.3.1. Interpretación de documentos XML

Existen 2 API (*Application Program Interfaz,* Interfaz para la Programación de Aplicaciones) para analizar los documentos XML; el Modelo de Objeto de documento (DOM, *Document Object Model*) y la API Simple para XML (SAX, *Simple API for XML*). DOM es un estándar del W3C que crea una vista de árbol del documento XML. Proporciona funciones estándar para manipular los elementos en el documento. Es preferible utilizar DOM si el documento XML contiene un documento.

SAX notifica cuando ciertos eventos ocurren al analizar el documento. Cuando se responde a un evento, cualquier dato que no se guarde específicamente se descarta. No proporciona un modelo de objetos. Y consume menos memoria.

1.6.4. Lenguaje de Marcado Inalámbrico (WML, *Wireless Markup Language*)

WML es un lenguaje de marcado ligero similar a HTML. El cual ha sido perfeccionado para usarse por dispositivos portátiles inalámbricos, para satisfacer las necesidades de comunicaciones inalámbricas. WML es un lenguaje derivado de XML y se especifica en un documento DTD.

WML se diseñó para dispositivos de banda estrecha con las siguientes restricciones:

- Área de despliegue pequeña, y facilidades limitadas para entradas de usuario.
- Conexión a redes de banda estrecha.
- Memoria y recursos computacionales limitados.

WML incluye principalmente 4 áreas funcionales:

Diseño y presentación de Texto. WML incluye soporte para imágenes y texto, incluyendo una variedad de comandos para el diseño y formato. Por ejemplo, se puede especificar texto en negritas.

Metáfora organizacional Página/Carta (*Deck/Card*). En WML toda la información esta organizada en una colección de cartas y páginas. Las cartas especifican una o más unidades de interacción de usuario (por ejemplo, un menú, una pantalla de texto, o un campo de entrada de texto). Lógicamente, un usuario navega a través de una serie de cartas WML, revisa los contenidos de cada una de ellas, ingresa la información requerida, hace elecciones, y se mueve a otra carta.

Las cartas se agrupan juntas en una página. Una página WML es similar a una página HTML en la que se identifica por una URL y es la unidad de transmisión de contenido.

Navegación y enlace entre cartas. WML incluye soporte para el manejo de la navegación entre cartas y páginas. Además WML incluye provisiones para el manejo de eventos en el dispositivo, el cual puede utilizarse para propósitos de navegación o para ejecutar *scripts*.

Parametrización de cadenas y administración del estado. Todas las páginas WML pueden parametrizarse utilizando un modelo de estado. Las variables pueden utilizarse en lugar de cadenas y se sustituyen en tiempo de ejecución. Esta parametrización se permite para un uso más efciente de los recursos de red.

1.7. INTERCONEXIÓN DE BASES DE DATOS [10]

1.7.1. Modelo de tres capas

Actualmente, las aplicaciones Web y de cómputo móvil, se construyen siguiendo el modelo de tres capas. Este modelo en tres capas o niveles surgió ante la necesidad de separar la lógica empresarial de la GUI (*Graphical User Interface*, Interfaz Gráfica de Usuario) y la base de datos remota. De acuerdo con este modelo, tres procesos separados y perfectamente definidos o, lo que es lo mismo, tres modelos se ejecutan en plataformas distintas:

1. La GUI, es decir, el navegador que se ejecuta en el dispositivo móvil del usuario.
2. El programa o programas de aplicación que se ejecutan en el servidor web y/o wap y que se encargan de procesar los datos (el nivel lógico empresarial).
3. Un sistema de base de datos que almacena los datos que requiere la capa 2 del modelo.

Este modelo se ilustra en la figura 1-4.

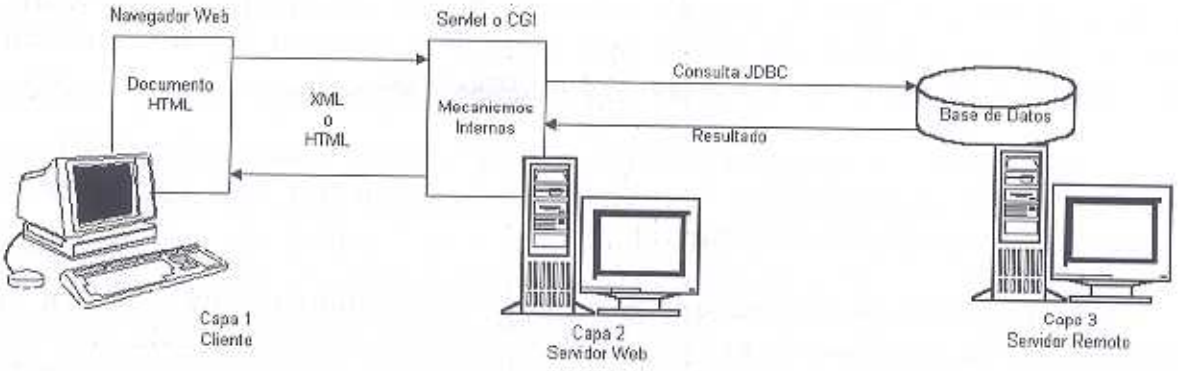

Figura 1-4. Arquitectura de aplicación web para acceder a bases de datos.

1.7.2. XML para representar Bases de datos [11]

Los Sistemas Manejadores de Bases de datos Relaciónales (RDBMS, *Relational Data Base Management System*) resultan muy eficaces para manipular grandes cantidades de datos y para ofrecer robustez, integridad, consistencia y disponibilidad. Sin embargo, cuando se trata de

transmitir datos entre distintos RDBMS, no existe un estándar que establezca un formato común para el intercambio de datos. Es aquí donde se puede utilizar XML para lograr la interoperabilidad. Como formato estándar de datos, XML cuenta con un amplio soporte por parte de los distribuidores.

Para lograr la interoperabilidad de las Bases de Datos con XML, es necesario crear un documento XML que contenga la estructura de la base de datos.

Existen varios mecanismos que permiten obtener un documento XML de una estructura de base de datos relacional. ADO 2.5 regresa un registro en formato XML, SQL Server 2000 permite extraer estructuras unidas como XML. Sin embargo, estas estructuras no pueden manejar situaciones muy complejas, como relaciones muchos a muchos. A continuación se explican 11 reglas definidas para obtener una estructura XML, a partir de una estructura de una base de datos relacional.

1. Elegir que información se va a incluir en la estructura, es decir las tablas y columnas de la base de datos que se requiere incluir en la DTD.
2. Crear un elemento raíz para el documento y declarar qué atributos de ese elemento son requeridos para manejar información adicional.
3. Modelar las tablas de contenido, es decir, crear un elemento en la DTD para cada tabla de la base de datos que se haya elegido para el modelo. Estos elementos deben declararse como EMPTY.
4. Modelar las columnas de las claves no foráneas. Esto se hace creando un atributo para cada columna que se haya decidido incluir en el documento XML (excepto las llaves foráneas). Estos atributos deben aparecer en la declaración !ATTLIST del elemento correspondiente de la tabla en la cual aparecen. Cada uno de estos elementos deben declararse como CDATA, y declararlos como #IMPLIED si aceptan nulos o como #REQUIRED en caso contrario.
5. Agregar un atributo de identificación a cada elemento que se haya creado en la estructura XML (a excepción del elemento raíz), este nuevo atributo debe nombrarse con el nombre del elemento seguido por ID. Los atributos se declaran como tipo ID, y #REQUIRED.
6. Representando tablas de búsqueda. Para cada clave foránea que se haya elegido para incluirse en la estructura XML y que haga referencia a una tabla de búsqueda se debe hacer lo siguiente:
    a. Crear un atributo en el elemento que representa la tabla en la cual se encuentra la llave foránea.
    b. Dar al atributo el mismo nombre de la tabla referenciada por la clave foránea, y hacerla #REQUIRED si la clave foránea no acepta nulos o #IMPLIED en caso contrario.
    c. Hacer los atributos del tipo lista enumerada.

7. Agregar contenido de elementos al elemento raíz, esto se logra agregando uno o varios elementos hijos, para cada tabla que modele el tipo de información que se desea representar en el documento.
8. Agregando relaciones a través de la contención. Para cada relación definida, si la relación es uno a uno o uno a muchos en la dirección en la que es navegada, y no hay ninguna otra relación que se dirija al hijo dentro del subconjunto seleccionado, entonces se agrega el elemento hijo como contenido de elemento del elemento padre con la cardinalidad apropiada (? Para uno a uno y * para uno a muchos).
9. Agregando relaciones utilizando IDREF/IDREFS. Identificar cada relación muchos a uno en la dirección que se ha definido, o cuyo hijo es el hijo en más de una relación definida. Para cada una de esas relaciones, se agrega un atributo IDREF o IDREFS al elemento del lado del padre de la relación, el cual apunta al ID del elemento en el lado hijo de la relación.
10. Agregar los elementos faltantes. Para cada elemento que únicamente es apuntado, agregar ese elemento como los elementos contenidos en el elemento raíz con el sufijo de cardinalidad *.
11. Remover los atributos ID no deseados o que no estén referenciados por atributos IDREF o IDREFS en alguna parte de la estructura XML.

Al seguir estos pasos se obtiene una DTD XML que contiene la estructura de la base de datos relacional. De esta forma se puede lograr el intercambio de información entre aplicaciones que manejen diferentes bases de datos.

1.7.3. Conectividad para Bases de datos en Java (JDBC, *Java Data Base Connectivity*)

Java dispone de una API estándar para acceder a sistemas de bases de datos en formato SQL (*Structured Query Language*, Lenguaje Estructurado de Consulta) denominada JDBC.

JDBC Proporciona una interfaz uniforme para acceder a las distintas bases de datos existentes independientemente de la implementación que tengan. Los distribuidores de bases de datos envían controladores JDBC que permiten acceder a sus bases de datos desde programas escritos en Java. Estos controladores, convierten las invocaciones JDBC en invocaciones nativas del RDBMS al que pertenece el controlador, así como los resultados del RDBMS en estructuras de datos JDBC.

Una vez instalado el controlador, se debe especificar la fuente de datos a la que se desea acceder. En JDBC, una fuente de datos se especifica por medio de un URL con el prefijo de protocolo jdbc:. La sintaxis del URL sería la siguiente:

Jdbc:<*subprotocolo*>:<*subnombre*>

Subprotocolo representa el tipo de fuente de datos, normalmente el nombre del sistema de base de datos, como db2 u oracle. Subnombre, por su parte, ofrece información acerca de la base de datos. El contenido y la sintaxis de subnombre dependen de subprotocolo. Por ejemplo, para acceder a una tabla denominada ejemplo y almacenada en un sistema DB2 local, el URL que tendría que crear sería:

```
String url = "jdbc:db2:ejemplo";
```

Si, en cambio, la base de datos se encuentra en una máquina remota, el URL sería:

```
String url = "jdbc:db2:monet.trl.ibm.com/ejemplo";
```

Para conectar la base de datos con un URL se puede hacer invocando el método getConnection():

```
Connection con = DriverManager.getConnection(url);
```

A menudo, las bases de datos están protegidas con contraseñas e identificadores de usuario para restringir el acceso a las mismas. Con JDBC, el identificador de usuario y la contraseña se especifican al conectarse con una base de datos:

```
String uid = "user";
String passwd = "password";
Connection con = DriverManager.getConnection(url, uid, passwd);
```

Una vez establecida la conexión, se pueden someter consultas a la base de datos. Sin embargo, antes es necesario crear un objeto Statement invocando el método createStatemente() del objeto Connection:

```
Statement stmt = con.createStatement();
```

Ahora ya se puede realizar una consulta SQL para obtener la información deseada.

```
String SQLQuery = "select * form ejemplo"
ResultSet rs = stmt.executeQuery(SQLquery);
```

La clase ResultSet define el número de métodos disponibles para acceder al resultado de la consulta. El resultado es básicamente una secuencia de filas, a las que el usuario puede acceder cuantas veces quiera utilizando el método next(). El resultado de una consulta mantiene un objeto cursor para indicar la fila que se está consultando dentro del conjunto de resultados obtenidos. Las invocaciones reiteradas del método next() permiten desplazar este cursor a la fila siguiente hasta el final de los datos, donde next() devuelve null. Dentro de la fila que incluya el cursor, se puede acceder al valor de cada columna con sólo especificar el número de índice de la columna o el nombre de la misma. El método getXX, en el que XX representa tipos de datos como Int y String, se puede utilizar también para acceder a cada una de las columnas.

1.7.4. Creación de contenido dinámico [12]

Cuando se construye una aplicación Web o de cómputo móvil resulta necesario personalizar la información que se despliega, es decir, que el contenido se genere dinámicamente. Java cuneta con dos tecnologías para lograrlo: los servlets y las JSP (*Java Server Pages*, Páginas Java en Servidor).

Los servlets son clases Java que amplían la funcionalidad de un servidor Web mediante la generación dinámica de páginas Web. Un entorno de ejecución denominado motor de servlets administra la carga y descarga del servlet, y trabaja con el servidor Web para dirigir peticiones a los servlets y enviar la respuesta a los clientes.

Desde su aparición en 1997, los servlets se han convertido en el entorno dominante de la programación Java en servidor y en un portal de uso generalizado para los servidores de aplicaciones. Los servlets aportan varias ventajas clave:

- **Rendimiento.** Las tecnologías anteriores como la Interfaz de pasarela común CGI (Common Gateway Interface) normalmente inician un nuevo proceso para manejar cada petición que les llega. Cuando la red era simplemente un repositorio para investigación académica y científica, no había demasiado tráfico y este sistema funcionaba bien. Los servlets, al contrario, se cargan cuando se solicitan por primera vez y permanecen indefinidamente en la memoria. El motor de servlets carga una instancia de la clase Servlet y le lanza peticiones empleando un conjunto de subprocesos disponibles (threads o hilos). La mejora del rendimiento con los servlets es considerable.

- **Simplicidad.** Los applets Java se ejecutan en una máquina virtual proporcionada por el navegador Web. Esto genera problemas de compatibilidad que incrementan la complejidad y limitan la funcionalidad de los applets. Los servlets simplifican esta situación considerablemente, ya que se ejecutan en una máquina virtual en un entorno de servidor controlado y sólo necesitan el http básico para comunicarse con sus clientes. No es preciso que el cliente tenga un software especial, ni siquiera en el caso de los navegadores antiguos.

- **Sesiones HTTP.** Aunque los servidores http no tienen capacidad para recordar detalles de una petición previa del mismo cliente, la interfaz API Servlet proporciona una clase HttpSession que permite superar esta limitación.

- **Acceso a la tecnología Java.** Al ser aplicaciones Java, los servlets tienen acceso directo a toda la gama de características Java, como el uso de subprocesos, acceso a redes y conectividad a bases de datos.

Los servlets son extensiones a un servidor Web que permiten crear contenido Web dinámicamente como respuesta a una petición del cliente. El motor de servlets los administra, se encarga de cargarlos e inicializarlos, les pasa un cierto número de peticiones a las que deben atender y después los descarga. Los servlets tienen ventajas clave sobre otros entornos de programación en el servidor:

Los servlets operan con un ciclo de vida fijo, que proporciona métodos de retrollamada (*callback*) a un motor de servlets para que los inicialicen, manejen las peticiones y los destruyan. La API proporciona dos modelos de subproceso: uno predeterminado consistente en subprocesos múltiples ejecutados en un solo ejemplar y el modelo de subproceso único alternativo.

Los servlets son la tecnología que sirve de base a las páginas JSP.

Una página JSP es una plantilla para una página Web que emplea código Java para generar un documento HTML dinámicamente. Las páginas JSP se ejecutan en un componente del servidor conocido como contenedor de JSP, que las traduce a servlets equivalentes.

Por esta razón los servlets y las páginas JSP están íntimamente relacionados. Lo que se puede hacer con una tecnología es, en gran medida, también posible con la otra; aunque cada una tiene capacidades

propias. Como son servlets, las páginas JSP tienen todas las ventajas de los servlets, pero además las páginas JSP tienen ventajas propias:

- Se vuelven a compilar automáticamente cuando es necesario.
- Como están en el espacio común de documentos del servidor Web, dirigirse a ellas es más fácil que dirigirse a los servlets.
- Como las páginas JSP con similares al HTML, tienen mayor compatibilidad con las herramientas de desarrollo Web.

1.8. Extensión de Socket Seguro de Java (JSSE, *Java Secure Socket Extension*) [13]

JSSE permite establecer comunicaciones seguras sobre Internet. Proporciona un marco de trabajo, así como una implementación en Java de los protocolos de Capa de Socket Segura (SSL, *Secure Socket Layer*) y de Seguridad en la Capa de Transporte (TLS, *Transport Layer Security*) e incluye funcionalidad para cifrar datos, autentificación del servidor, integridad de los mensajes y autentificación opcional del cliente. Utilizando JSSE se puede implementar comunicación segura entre clientes y servidores que utilicen cualquier protocolo de aplicación como HTTP, Telnet o FTP.

Al presentar una capa de abstracción que oculta la complejidad de los algoritmos de seguridad, así como de los mecanismos de establecimiento de acuerdos (*handshaking*), JSEE minimiza el riesgo de crear vulnerabilidades de seguridad peligrosas. Además, simplifica el desarrollo de aplicaciones al proporcionar un bloque que puede ser integrado directamente en una gran gama de aplicaciones.

1.8.1 Capa de Socket Segura (SSL)

El protocolo SSL es un sistema diseñado y propuesto por *Netscape Communications Corporation*. Se encuentra en la pila OSI entre los niveles de TCP/IP de los protocolos HTTP, FTP, SMTP, etc. Proporciona sus servicios de seguridad cifrando los datos intercambiados entre el servidor y el cliente con un algoritmo de cifrado simétrico, y cifrando la clave de sesión mediante un algoritmo de cifrado de llave pública. La clave de sesión es la que se utiliza para cifrar los datos que vienen del servidor seguro y van al mismo. Se genera una clave de sesión distinta para cada transacción, lo cual permite que aunque sea vista por un atacante en una transacción dada, no sirva para descifrar futuras transacciones.

Proporciona cifrado de datos, autentificación de servidores, integridad de mensajes y, opcionalmente, autentificación de cliente para conexiones TCP/IP.

Cuando el cliente pide al servidor seguro una comunicación segura, el servidor abre un puerto cifrado, manejado por el protocolo SSL Record, situado encima de TCP. Es el protocolo SSL, quien utiliza el Protocolo SSL Record y el puerto abierto para comunicarse de forma segura con el cliente.

1.8.1.1. Protocolo SSL

Durante el protocolo SSL, el cliente y el servidor intercambian una serie de mensajes para negociar las mejoras de seguridad. Este protocolo sigue las siguientes fases (figura 1-5):

1. **Cliente hola.** El cliente envía la versión más alta de SSL, así como una lista de suites de cifrado soportadas por él.

2. **Servidor hola.** El servidor elige la versión más alta de SSL y la mejor suite de cifrado.

3. **Certificado.** El servidor envía un certificado o una cadena de certificados.

4. **Petición de certificado.** Si el servidor necesita autentificar al cliente.

5. **Servidor Intercambio de llaves.** El servidor envía un mensaje de intercambio de llaves, cuando la información del paso 3 no es suficiente.

6. **Servidor hola listo.** El servidor termina con el intercambio inicial de mensajes.

7. **Certificado.** Si el servidor solicitó un certificado en el paso 4, el cliente envía una cadena de certificados similar al del mensaje 3.
8. **Cliente Intercambio de llaves.** El cliente genera información utilizada para crear una llave que se utilizará en el cifrado simétrico. Para RSA el cliente cifra esta información con la llave pública del servidor y la envía.

9. **Verifica Certificado.** Este mensaje se utiliza para completar la autentificación del cliente. El cliente envía información que es

firmada digitalmente usando una función *hash* criptográfica. Cuando el servidor descifra esta información con la llave pública del cliente, el servidor puede autentificar al cliente.

10. **Cambia especificación de cifrado.** El servidor solicita al cliente cambiar el modo de cifrado.

11. **Finalizado.** El servidor informa al cliente que está listo para comenzar la comunicación segura. Esto finaliza el acuerdo *SSL*

Figura 1-5. Mensajes SSL.

1.8.1.2. El Protocolo SSL Record

El Protocolo SSL Record especifica la forma de encapsular los datos transmitidos y recibidos. La porción de datos del protocolo tiene tres componentes:

- MAC-DATA, el código de autenticación del mensaje.
- ACTUAL-DATA, los datos de aplicación a transmitir.
- PADDING-DATA, los datos requeridos para rellenar el mensaje cuando se usa cifrado en bloque.

1.9. TRABAJOS RELACIONADOS

A continuación, se mencionan las características de dos sistemas de cómputo móvil, relacionados con el sistema propuesto en esta tesis. Uno de ellos (INFOPARCO) también realiza asignación de recursos, para un parque natural. El otro trabajo (MediPad), es un sistema de Telemedicina, que principalmente realiza diagnósticos remotos.

1.9.1. INFOPARCO [14]

INFOPARCO es un sistema que mejora el aprovechamiento de los recursos ambientales, arquitectónicos, naturales y económicos del parque natural Velino-Sirente en la región Abruzzo. El sistema soporta acceso a través de diferentes tecnologías, notablemente a través de una interfaz Web y una WAP, asequibles a través de terminales móviles. Los tipos de servicios accesibles a través de la interfaz WAP son varios, de naturaleza estática y dinámica, más detalladamente, se puede acceder a información sobre servicios de uso público, alojamiento, reporte metereológico, descripción de las áreas de descanso naturales e itinerarios, exhibiciones culturales y esparcimiento en general.

1.9.2. MEDIPAD [15]

MediPad es una aplicación de cómputo móvil para intercambiar información medica e imágenes de forma segura, además de obtener diagnóstico, supervisón y monitoreo remoto de los pacientes. Esta aplicación, actualmente está funcionando como un prototipo, cuenta con una plataforma que consiste de terminales móviles y una red de alta velocidad que conecta a siete provincias al Centro Medico de la Universidad de Indiana y Purdue.

1.9.3. Visor PDA para Servicios de Vigilancia Móviles (PDA Watch for Mobile Surveillance Services) [16]

Dotar de servicios de vigilancia en tiempo real a dispositivos de mano (handheld) sobre Internet es una aplicación interesante en el ambiente móvil. Es necesario desarrollar cada sistema de acuerdo a las necesidades de cada aparato, considerar los retos del diseño como la heterogeneidad en las plataformas, y la limitación en el ancho de banda.

En este trabajo, se desarrolló un sistema de vigilancia móvil basado en Java, el cual soporta a usuarios de PDA para superar los retos antes mencionados. Se aplican las tecnologías Java emergentes: J2ME, RMI y JMF para el cómputo móvil, dispositivos de mano, cómputo distribuido y cómputo multimedia. Se diseño un protocolo de comunicación textual basado en el nivel de aplicación.

El objetivo del sistema es proporcionar acceso universal a servicios de vigilancia por quien sea, a la hora que sea y en donde sea.

Basándose en el método de solución de J2ME, se propuso un esquema cliente-servidor el cual facilita el desarrollo del sistema de vigilancia móvil. Éste se compone de objetos distribuidos que se comunican a través de un mecanismo de intercambio de mensajes en el paradigma orientado a objetos.

El sistema de vigilancia móvil es un sistema basado en componentes. La figura 1-6 muestra la arquitectura del servidor de vigilancia. Contiene los siguientes componentes: *Watch Daemon* (Demonio de Observación), *WTP Parser* (Analizador WTP), *SnapshotCapturer* (Capturador de instantáneas), *Message Agent* (Agente de mensajes), *eMail Agent* (Agente de correo), *Fax* y *Page Server* (Servidor de páginas), *Image Manager* (Administrador de imágenes), y *Login Manager* (Administrador de acceso).

1.10. CONCLUSIONES

A lo largo de este capítulo, se han mencionado las tecnologías con las que se pueden desarrollar aplicaciones de cómputo móvil y ahora se puede tomar una decisión sobre cuales utilizar y cómo se van a integrar en el sistema.

En primer lugar se debe elegir una plataforma para los teléfonos móviles. En este capítulo, se han presentado WAP y J2ME. Ambas plataformas tienen ventajas y desventajas. De acuerdo a las características y limitaciones que presentan y al tipo de dispositivos que se van a utilizar en el sistema (teléfonos celulares), se observa lo siguiente. Todos los teléfonos con soporte de navegación en Internet, cuentan con el protocolo WAP implementado. Por otro lado, los teléfonos que cuentan con soporte Java, apenas se están introduciendo al mercado mexicano. Por lo tanto, se trabajará con el protocolo WAP.

En cuanto al acceso a las bases de datos se construirá una DTD XML. En la sección 1.7.2 se habló sobre como crear una DTD XML que contenga la estructura de la base de datos. Siguiendo esa serie de pasos, se representa toda la base de datos. Para el caso de la aplicación que se va a desarrollar, no es necesario conocer a detalle la estructura de la base de datos. Se requiere saber información relacionada con el software de base de datos en la que esta construida. Por lo tanto, es necesario un documento XML de consulta que incluya un nombre de base de datos, el manejador que utiliza, información relevante sobre la estructura de la base de datos, y demás elementos pertinentes. Debido a que la información que contendrá el documento XML es confidencial, será necesario construir un mecanismo de transmisión de datos seguro y para esto se utilizará JSSE.

Por otro lado, la información que se va a desplegar en los teléfonos celulares se genera dinámicamente de acuerdo al tipo de recurso solicitado y a los CR que lo puedan otorgar. En la sección 1.7.4 se mencionaron los servlets y las JSP, ambas tecnologías de Java, plataforma en la cual se desarrollará la aplicación. Se decidió trabajar con servlets por simplicidad, ya que una JSP es en si un servlet, poseen las mismas ventajas y desventajas. Al utilizar servlets se ahorra el paso de traducir el código fuente JSP al código fuente de un servlet Java equivalente y espacio de almacenamiento.

Con esto, se obtendrá una arquitectura flexible en la que se podrán generar consultas dinámicas personalizadas para cada base de datos. Se profundizará más sobre este tema en el siguiente capítulo.

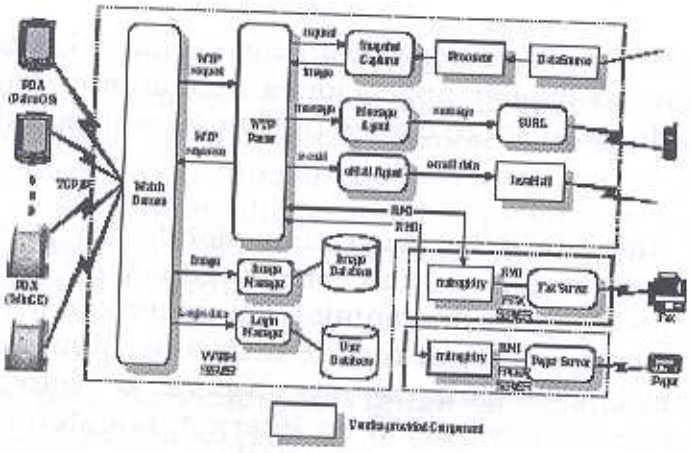

Figura 1-6. Arquitectura del visor PDA.

Para el cliente móvil, se propuso una estructura de capas que se muestra en la figura 1-7. Ésta facilita el desarrollo del monitor móvil de vigilancia (PDA Watcher), el cual permite a los usuarios remotos ingresar al servidor de vigilancia e inspeccionar el punto que se esta monitoreando.

Figura 1-7. Estructura de capas del Visor PDA.

El Protocolo de Transmisión Observada (WTP) es un protocolo como HTTP, el cual define la comunicación entre el servidor de vigilancia y los PDA a través del puerto 4111.

Este sistema, se puede integrar a varias aplicaciones como: monitoreo de tráfico, terapia intensiva en hospitales, catálogos electrónicos, comercio electrónico, etc.

1.11. REFERENCIAS.

[1] Varshney, U. Y Vetter, R. "A Framework for the Emerging Mobile Commerce Applications", 34th Hawaii International Conference on System Sciences, 2001.

[2] Nixon, P. Y Cahill, V. "Mobile Computing: Technologies for a disconnected society", Mobile Computing, Enero – Febrero 1998.

[3] Huidobro, J. "Internet móvil". La nueva era de las comunicaciones personales", Bit, No. 125.

[4] Yi-Bing Lin, Imrich Chlamtac "Wireless and Mobile Network Architectures", Wiley, 2001.

[5] Wiley John, "Official Wireless Application Protocol", Wiley, 1999.

[6] Jonathan Knudsen. "Wireless Java Developing with Java 2 Micro Edition". Apress. 2001.

[7] The CLDC HotSpot, Java 2 Plataform, micro Edition, http://java.sun.com/products/cldc/wp/CLDC_HI_WhitePaper.pd

[8] MIDP APIs for Wireless Applications, http://java.sun.com/products/midp/midp-wirelessapps-wp.pdf

[9] McLaughlin, B., "Java and XML", O'Reilly, Junio 2000.

[10] Hiroshi Maruyama, Kent Tamura, Naohiko Uramoto. "Creación de sitios Web con XML y Java".Prentice Hall, 2000.

[11] Kevin Williams, et al. "Professional XML Databases". Wrox, 2000.

[12] Phil Hanna, Manual de referencia JSP. McGraw-Hill, 2002.

[13] Java Secure Socket Extension (JSSE) Reference Guide for the Java 2 SDK, Standard Edition, v.1.4.

[14] C. Colafigli, P. Inverardi, R. Matricciani, "INFOPARCO: An experience in designing an information system accesible through WEB and WAP interfaces", Proceedings of the 34th Hawaii International Conference on System Sciences, 2001.

[15] MediPad, http://www.cs.purdue.edu/research/cse/mobile/medipad.html

[16] Sheng-Tun Li et.al., "PDA Watch for Mobile Surveillance Services", IEEE Workshop on Knowledge Media Networking, 2002.

Capítulo

2

Arquitectura

Resumen

En este capítulo se explicará en detalle la arquitectura del Sistema de Asignación de Recursos para entidades móviles basado en una infraestructura de telefonía celular..

2.1. INTRODUCCIÓN

En este capítulo se explicará en detalle la arquitectura del Sistema de Asignación de Recursos. Para la realización del sistema, se utilizará la infraestructura de telefonía celular existente, particularmente GSM por ser el estándar de comunicación más utilizado en el mundo y además por cuestiones de seguridad, ya que cuenta con mecanismos para identificar al usuario que se está conectando al sistema.

Los protocolos de comunicación que se utilizan son WAP para la comunicación del cliente con el Administrador Global de Recursos (AGR) y HTTP para la comunicación entre el AGR y los Contenedores de Recursos (CR). Una vez identificada la plataforma de desarrollo, se puede crear la arquitectura del sistema.

La arquitectura del Sistema se basa en el modelo de funcionamiento WAP, la cual es una adaptación de la arquitectura definida para el World Wide Web (WWW), de acuerdo a los requerimientos de las aplicaciones WAP. En la figura 2-1, se muestra el modelo de funcionamiento WAP.

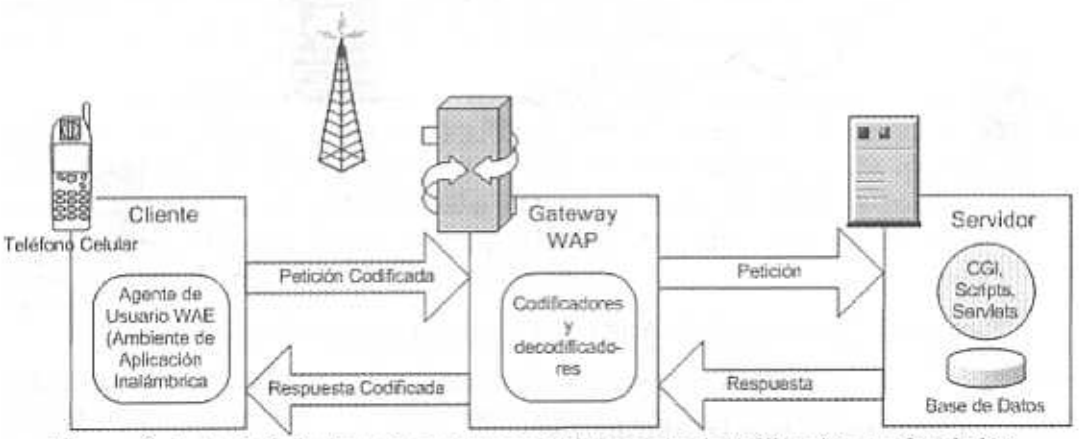

Figura 2-1. Modelo de funcionamiento del Protocolo de Aplicación Inalámbrica.

Como se muestra en la figura 2-1 en el teléfono celular existe un micro navegador[1] encargado de la coordinación con el *gateway* WAP. Éste recibe peticiones de información del micro navegador y agrega a la petición un encabezado HTTP que redirige al servidor de información adecuado. Una vez procesada la petición en el servidor, se envía al *gateway* WAP. Éste la procesa de nuevo para enviarla al teléfono celular ahora con un encabezado WAP.

[1] El micro navegador, actúa como interfaz de usuario de la misma forma en la que lo hacen los navegadores estándar.

2.2. ARQUITECTURA DEL SISTEMA

En la figura 2-2. Se muestra la arquitectura general del sistema la cual se basa en el modelo de funcionamiento WAP explicado en la sección anterior. La arquitectura se divide en 3 capas:

- Comunicación inalámbrica.
- Administrador Global de Recursos.
- Contenedores de Recursos.

En secciones subsecuentes se explican a detalle estas capas.

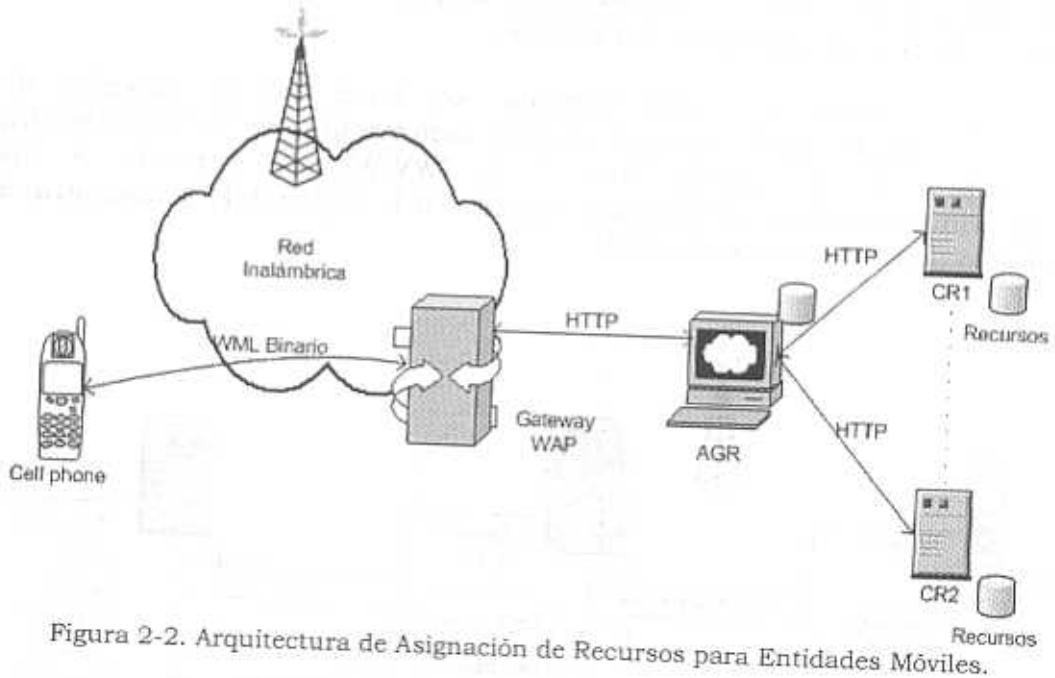

Figura 2-2. Arquitectura de Asignación de Recursos para Entidades Móviles.

2.2.1. Comunicación inalámbrica

En la parte inalámbrica se encuentra el cliente, el cual puede ser cualquier teléfono celular con soporte WAP. Para el desarrollo de esta tesis, se utilizará además de teléfonos celulares, el simulador Openwave SDK WAP Edition 5.0[1]. Este simulador proporciona tanto la funcionalidad del servidor WAP como la funcionalidad de un teléfono celular. Por medio de la simulación se podrá verificar el funcionamiento del sistema en dispositivos de diversos fabricantes.

La forma en la que se inicia la comunicación es por medio del cliente. Éste envía varios parámetros para que en base a estos el AGR logre asignarle el recurso adecuado. Además se implementa un

mecanismo para manejar las desconexiones durante el proceso de una petición. En el caso de que ocurra una desconexión, cuando el usuario reingrese al sistema, éste le informará el estado en el que se encuentra su petición dándole la oportunidad de continuar o cancelar la misma.

Otro actor importante dentro de la comunicación inalámbrica es el Gateway WAP. Como se explicó en la sección anterior encapsula la petición del cliente (que esta en WML), en una petición HTTP. Para que se pueda dar la comunicación entre el cliente y el AGR. Entre la información que agrega el gateway WAP está el MSISDN (*Mobile Station ISDN Number*). Este número es un identificador único del teléfono, con lo cual se puede saber quién está ingresando al sistema y así tener un control de acceso, para evitar el mal uso del sistema.

2.2.2. Contenedores de Recursos

Los CR son servidores que administran recursos localmente. Aunque los recursos que administran son del mismo tipo, son independientes unos de otros. Por esta razón, los tipos de RDBMS que utilizan pueden ser diferentes, al igual que la estructura de información que poseen. Los CR tienen una base de datos, en la cual registran los recursos que poseen, su disponibilidad, y también cuales ya están asignados, y por lo tanto no se pueden otorgar.

Para realizar una mejor asignación de recursos es necesario administrarlos globalmente. Para lograrlo, cada CR debe indicarle de alguna forma al AGR, qué recursos tiene disponibles, y de que manera puede acceder a su información interna. Esto se hace a través de un documento XML en la que se especifica el nombre y la estructura de la base de datos, el RDBMS que utiliza, etc.

Por cada CR se proporcionará una interfaz que le pedirá al administrador del CR la información necesaria para que el AGR pueda comunicarse con éste. Dando por resultado un documento XML con la información del CR.

En el AGR hay un modulo de administración en el que se pueden registrar las URL de cada CR. De esta forma, el AGR conoce la URL de cada CR y puede comunicarse con cada uno para poder descargar y analizar el documento XML. Debido a que éste contiene información de carácter privado como contraseñas para acceder a las Bases de Datos, etc., es necesario utilizar un método para proporcionar medidas de seguridad adicionales. Esto se logra a través del protocolo SSL (Para mayor referencia ver sección 1.8 del capítulo 1).

Para que la información alojada en un dominio pueda verse protegida bajo el protocolo SSL es necesario instalar en dicho domino un Certificado de seguridad. Los datos contenidos en éste permiten cifrar la información intercambiada y la certificación del propietario de la información. Estas dos técnicas son los principios en los que se basa el funcionamiento del protocolo SSL.

Un Certificado de seguridad es un conjunto de documentos electrónicos emitidos por una entidad certificadora, que permiten cifrar la información transmitida e identificar a la fuente de dicha información. Para que el certificado sea fiable, la citada entidad debe ser un organismo de confianza capaz de garantizar la procedencia de la información, que es lo que en cierta medida proporciona la seguridad.

Es necesario, además contar con un Servidor seguro que sea capaz de manejar una comunicación segura con el cliente, para que la información que se transmita esté cifrada, y solo pueda ser obtenida por un cliente certificado.

La función del Servidor Seguro instalado en el CR es esperar a que el Cliente Seguro instalado en el AGR le pida descargar el documento XML. En cuanto reciba la petición, Servidor y Cliente seguirán el protocolo de comunicación SSL (mostrado en la figura 1-5. del capítulo 1). De esta forma, se evita el riesgo de que personas ajenas tengan acceso a la información transmitida. La forma en la que se realiza la comunicación entre El AGR y los CR's se muestra en la figura 2-3.

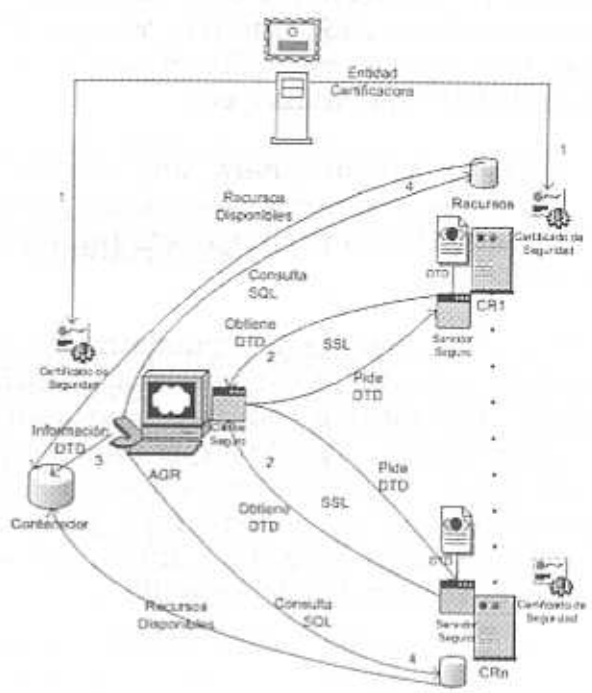

Figura 2-3. Intercambio de Información entre el AGR y los CR's

2.2.3. Administrador Global de Recursos

El AGR se encuentra en un servidor Web, y se divide en dos partes. Una se encarga de hacer la administración de los recursos y la otra administra las peticiones del cliente. En la figura 2-3, se muestra un esquema de cómo funciona el AGR.

El AGR al igual que los CR solicita a una Entidad Certificadora un Certificado de Seguridad (1). Por medio de éste, el servidor ubicado en cada CR puede identificar al AGR como un cliente válido. Una vez identificado, le permite descargar el documento XML (2). Una vez que el AGR puede acceder a éstos, guarda información en su base de datos (3). Después con la información obtenida construye consultas SQL personalizadas para cada base de datos de los CR en las que solicita información sobre los recursos que ofrecen y su disponibilidad (4). Este proceso, se ejecutará automáticamente cada 5 minutos para lograr que la información este actualizada. Esto se hace, porque cada CR es capaz de asignar recursos localmente y en dicha asignación no forma parte este sistema. Así, cuando desde una entidad móvil se solicite un recurso, el AGR obtendrá (en base a su petición) uno o más recursos que cumplan con las características solicitadas. De los recursos enviados por el AGR, el usuario podrá elegir la que más le convenga. El AGR se comunica entonces con el CR al que pertenece el recurso y realiza el proceso de asignación. Con esto, se reducen accesos innecesarios a todos los CR cada vez que se solicite un recurso, y solo se busca en aquellos que mejor se adapten a las necesidades del usuario y tengan disponibilidad.

Cuando el AGR reciba una petición de un cliente móvil, el sistema le pedirá las características del recurso solicitado, con estos parámetros, el servidor buscará en su base de datos cual o cuales CR's pueden otorgarlo y le envía esta información al cliente, para que en caso de que el sistema le presente más de uno, pueda elegir el que más le convenga. Cuando el usuario elige un recurso, el servidor se comunica con el CR elegido para que pueda hacer la asignación. Una vez que se asigno al cliente, tanto el CR como el AGR, eliminan la disponibilidad de éste en sus bases de datos.

2.3. CONCLUSIONES

En éste capítulo se presentó la arquitectura del Sistema de Asignación de Recursos para Entidades Móviles, y se explicó a detalle las capas que la conforman.

En los siguientes capítulos se presentará el análisis, diseño e implementación del caso de prueba elegido en el planteamiento del

problema. Se aplicará el Sistema de Asignación de Recursos para Entidades Móviles al Sector Salud del D. F. y área Metropolitana para lograr una mejor asignación de los recursos médicos y reducir los tiempos en los que se canaliza un paciente a un hospital.

Así tenemos que los CR's son los Hospitales, los Recursos son las Camas, y los Dispositivos Móviles son teléfonos celulares que se encontrarán en las ambulancias.

2.4. REFERENCIAS

[1] Openwave, http://www.openwave.com

Capítulo

3

Análisis

Resumen

En este capítulo se realizará el análisis para obtener el modelo del sistema que describe el dominio de aplicación. También se definirá la manera en la que interactúan los actores y el sistema para manipular el modelo del dominio de aplicación y de esta forma, tener preparada una arquitectura del sistema.

3.1. INTRODUCCIÓN

Como se menciono en el capítulo anterior, en esta tesis, se pretende ofrecer una solución al problema de la asignación de recursos. Específicamente para entidades móviles. Para ilustrar la forma en la que se asignarán los recursos, se trabajará en el Sector Salud por las razones expuestas en la sección Planteamiento del problema.

En este capítulo se realizará el análisis con lo cual se obtendrá el modelo del sistema que describe el dominio de aplicación. También se definirá la manera en la que interactúan los actores y el sistema para manipular el modelo del dominio de aplicación y de esta forma, tener preparada una arquitectura del sistema sobre la que se trabajará en el capítulo 4 (Diseño).

3.2. ESPECIFICACIÓN DEL SISTEMA

Se necesita construir un sistema capaz de administrar los recursos con los que cuentan los hospitales del Distrito Federal en base a su capacidad. En este caso, los recursos son las especialidades que maneja un hospital y la capacidad, es el número de pacientes que se pueden atender en esa especialidad.

El sistema busca reducir el tiempo de asignación. Es decir, el tiempo que pasa desde que una ambulancia recoge a un paciente hasta que es aceptado por un Hospital. De tal forma que, mientras se evalúa el estado del paciente, uno de los tripulantes de la Ambulancia se conecta por medio de un teléfono celular al sistema para localizar un hospital que cuente con los servicios necesarios para la atención del paciente y que tenga la capacidad de atenderlo.

En el sistema existen tres elementos importantes:

- La Interfaz de Usuario y el procesamiento de las peticiones.
- La administración de los recursos que ofrecen los Hospitales.
- El soporte a los administradores.

La Interfaz de Usuario, debe ser muy práctica e intuitiva, debido a que se utilizarán teléfonos celulares que cuentan con pantallas muy reducidas. En promedio en estas pantallas se pueden desplegar 5 líneas de 12 caracteres. Además no es recomendable utilizar interfaces en las que se le pida al usuario ingresar algún dato por medio del teclado, ya que por la

disposición del teclado en los teléfonos celulares, ingresar un simple nombre puede tomar mucho tiempo.

El sistema para poder procesar una petición de usuario, necesita saber, el tipo de hospital al que se va a canalizar el paciente (IMSS, ISSSTE, Privado) de acuerdo a sus prestaciones, si se conocen en ese momento. De lo contrario, se seleccionará Cruz Roja o Salubridad. También se necesita proporcionar una especialidad (Urgencias, General, Ginecología, Traumatología, etc.). Con estos datos, el sistema le podrá presentar al usuario, él o los hospitales en los que pueden recibir y atender al paciente.

Para realizar una correcta administración de los recursos, se debe tener en cuenta que los hospitales manejan información en diferentes formatos (diferentes RDBMS, diferentes diseños, etc.). Por esta razón, se debe construir un mecanismo que permita el intercambio de información entre bases de datos heterogéneas. Una solución a este problema, se mencionó en las conclusiones del capítulo 1. Esta solución, se desarrollará de la siguiente forma. Es necesario que en el servidor de cada Hospital se tenga un documento XML que contenga los datos del hospital, así como la estructura de su Base de Datos. Con esta información se pueden construir consultas SQL personalizadas a cada Hospital. La comunicación se hará de la siguiente manera. El administrador del SAR se conecta por medio de un cliente al servidor del Hospital para descargar el documento XML. Del cual extrae la información contenida que almacena en la base de datos del AGR. Después se conecta a cada hospital para buscar los servicios con los que cuenta además de su disponibilidad y almacena esta información en la base de datos del AGR. Este proceso lo realizará cada 5 minutos para mantener la consistencia en la información. Esto se debe a que cada Hospital también asigna sus recursos localmente.

La forma en la que se dará soporte a los administradores es por medio de interfaces. El administrador del Hospital tiene que generar un Documento XML con la información del Hospital. Así que se le presentará una interfaz para que este proceso sea más sencillo. Por otro lado, el Administrador del Sistema de Asignación de Recursos (SAR) se encargará de registrar en el sistema a los Hospitales y a los Usuarios que formarán parte del SAR. Para que este proceso sea más sencillo, también se le presentará una interfaz de usuario.

3.3. ESPECIFICACIÓN DE REQUISITOS O REQUERIMIENTOS

3.3.1. Requerimientos no funcionales

Los requerimientos no funcionales describen aspectos del sistema visibles para el usuario que no están relacionados en forma directa con el comportamiento funcional del sistema. Los requerimientos no funcionales abarcan varias cuestiones. Desde la apariencia de la interfaz de usuario hasta los requerimientos de tiempo de respuesta y las cuestiones de seguridad.

Interfaz de Usuario y factores humanos. Como se mencionó en la sección anterior, Los usuarios van a utilizar teléfonos celulares para conectarse al sistema, y debido a las características que éstos presentan (pantallas reducidas, dificultad de ingresar texto a través del teclado) es recomendable no manejar más de 9 líneas por cada carta, y utilizar listas desplegables para obtener información del usuario, y para que la navegación sea más intuitiva utilizar las etiquetas "*accept*" y "*options*" en vez de utilizar ligas propias.

Cuestiones de Seguridad. Debido a que el AGR se comunica con los hospitales para obtener la DTD. Y que ésta contiene información privada, relacionada con los hospitales a la que nadie externo debe tener acceso. Se debe proporcionar un mecanismo para que la transferencia de la DTD sea segura. En cuanto al SAR debe ser capaz de reconocer a los dispositivos móviles desde los cuales se hace la petición. También debe validar al usuario que ingresa al sistema.

3.3.2. Requerimientos funcionales.

Manejo de Usuarios. Los usuarios serán identificados por medio del MSISDN (*Mobile Station* ISDN *Number*) que identifica a cada teléfono celular con un número único. Cada ambulancia contará con un teléfono celular, el cual será utilizado por cualquier tripulante de la misma. El Administrador del SAR dará de alta en la base de datos estos números, para que el sistema pueda identificar al dispositivo desde el cual se esta haciendo la petición.

Además, a cada usuario se le asignará una clave numérica de 4 dígitos. Esto es para evitar que alguna persona ajena al sistema y que tenga acceso a un teléfono registrado en el mismo pueda hacer un mal uso del sistema.

Manejo de Hospitales. Los hospitales serán identificados por medio de su URL. El administrador del SAR se encargará de dar de alta en la bese de datos las URL de los hospitales que formaran parte del sistema. En base a su URL el sistema se comunicará con los hospitales y llevará a cabo la administración y asignación de sus recursos.

Manejo de desconexión. El sistema debe ser capaz de guardar el estado de la petición hasta que el recurso se asigne. Así, si ocurre una desconexión, cuando el usuario reingrese al sistema, éste debe indicarle que tiene una petición pendiente y darle la opción de continuarla o cancelarla.

3.4. IDENTIFICACIÓN DE LOS ACTORES

Un actor describe cualquier entidad que interactúa con el sistema (un usuario, otro sistema, el ambiente físico del sistema). Para definir correctamente los alcances del sistema, es necesario identificar a los actores y a los casos de uso. Los actores están fuera de la frontera del sistema, mientras que los casos de uso están dentro de la frontera del sistema.

Los actores que intervienen en el sistema son:

- Usuario móvil
- Administrador del Hospital
- Administrador del SAR
- Hospital

3.4.1. Descripción de los actores

Nombre del Actor:	Administrador del Hospital
Definición:	Es el encargado de solicitar que el Hospital se agregue al SAR. Conoce toda la información relacionada con el Hospital y cómo está constituida su base de datos.
Notas:	➢ Tiene la capacidad de Crear un documento XML que contiene información sobre el Hospital y su Base de Datos. ➢ Ejecuta el Servidor Seguro para la transferencia segura del documento XML.

Nombre del Actor:	Usuario Móvil
Definición:	Se encarga de solicitar un recurso en un hospital. Proporcionando un tipo y una especialidad.
Notas:	- Puede ser cualquier tripulante de la ambulancia siempre y cuando este registrado en la base de datos del sistema.

Nombre del Actor:	Administrador del SAR.
Definición:	Es el encargado de administrar el Sistema y los módulos que lo conforman.
Notas:	➢ Tiene la capacidad registrar a los Hospitales que formarán parte del sistema. ➢ Tiene la capacidad de registrar a los dispositivos móviles desde los cuales se podrá ingresar al sistema. Además de registrar a los usuarios del sistema. ➢ Tiene la capacidad de activar el SAR, el AGR y el cliente Seguro por medio del cual se garantiza la transmisión segura de un documento XML.

3.5. IDENTIFICACIÓN DE CASOS DE USO [1]

Con el Sistema de Asignación de recursos que propone esta tesis, se pueden reducir los tiempos en los que una ambulancia lleva a un herido o paciente a un hospital para su atención. Cualquier tripulante de la ambulancia, puede solicitar una cama en algún hospital desde un teléfono celular ingresando al servicio de Asignación de Recursos (AsignaRecursos en la figura 3-1). Este servicio debe soportar a más de un usuario.

Para que el sistema funcione como se planteó anteriormente, es necesario realizar algunas cosas antes de que el usuario ingrese al sistema.

En la figura 3-2, se muestra el caso de uso ActualizarContenedor al cual se hace referencia en el caso de uso mostrado en la figura 3-1. Una vez que el usuario ha recibido su número de confirmación el AGR busca en su base de datos el nombre del Hospital y el Servicio con los que se hizo la reservación y actualiza el campo de disponibilidad.

Nombre del Caso de Uso:	AsignaRecursos
Actores participantes:	Usuario Móvil y Hospitales
Condición inicial:	1. El Usuario Móvil ingresa al SAR desde su teléfono celular.
Flujo de eventos:	2. El SAR verifica que sea un usuario válido e inicia una sesión. 3. El Usuario Móvil ingresa el tipo de hospital y servicio que requiere. 4. El SAR le muestra el o los hospitales que pueden cubrir su necesidad. 5. El Usuario Móvil elige un hospital. 6. El SAR se comunica con el hospital elegido para hacer la reservación. 7. El hospital le envía al SAR su número de confirmación. 8. El SAR envía al dispositivo móvil el número de confirmación. 9. El hospital actualiza su Base de Datos. 10. Se llama al Caso de Uso ActualizarContenedor.

Figura 3-1. Caso de Uso AsignaRecursos del Sistema de Asignación de Recursos.

Primero, el administrador del SAR deberá registrar en la base de datos del AGR los Hospitales que participarán en el sistema. Desde una PC el administrador del SAR deberá ingresar al servicio de mantenimiento en la página Web (RegistrarHospital en la figura 3-3). La forma en la que se va a registrar un Hospital, es por medio de su URL.

Nombre del Caso de Uso:	ActualizarContenedor
Actores participantes:	Usuario Móvil
Condición inicial:	1. El SAR obtiene del caso de uso AsignaRecurso el nombre del Hospital y Servicio que le fueron asignados al Usuario Móvil.
Flujo de eventos:	2. El SAR busca en su Base de datos los datos obtenidos para poder actualizar la disponibilidad. 3. El SAR actualiza su Base de datos con el nuevo valor para el campo Disponibilidad.

Figura 3-2. Caso de Uso ActualizaContenedor del Sistema de Asignación de Recursos.

Nombre del Caso de uso:	RegistrarHospital
Actores participantes:	Administrador del SAR.
Condición inicial:	1. El Administrador del SAR ingresa al servicio Web de mantenimiento.
Flujo de eventos:	2. La página le pedirá sus datos para comprobar que tiene permiso para utilizar el servicio. 3. Después de registrarse en forma satisfactoria, El servicio le presenta un menú en el que debe indicarle que desea ir a la sección de Hospitales. 4. El servicio le presentará un menú en la que le pedirá la acción a realizar. 5. Una vez que el usuario le indique al servicio que quiere registrar un hospital, le pedirá ingresar la URL del Hospital. 6. El Administrador del SAR guarda los cambios en la base de datos.

Figura 3-3. Caso de Uso RegistrarHospital del Sistema de Asignación de Recursos.

Después, el administrador del SAR deberá registrar en la base de datos del AGR a los Usuarios autorizados para utilizar el sistema. Desde una PC, el administrador del SAR deberá ingresar al servicio de mantenimiento en la Web (RegistrarUsuario en la figura 3-4).

Nombre del Caso de uso:	RegistrarUsuario
Actores participantes:	Administrador del SAR.
Condición inicial:	1. El Administrador del SAR ingresa al servicio Web de mantenimiento.
Flujo de eventos:	2. La página le pedirá sus datos para comprobar que tiene permiso para utilizar el servicio. 3. Después de registrarse en forma satisfactoria, El servicio le presenta un menú en el que debe indicarle que desea ir a la sección de Usuarios. 4. El servicio le un menú en el que le pedirá la acción a realizar. 5. Una vez que el usuario le indique al servicio que quiere registrar un Usuario, le pedirá ingresar el nombre y la clave del usuario que va a registrar. 6. El Administrador del SAR guarda los cambios en la base de datos.

Figura 3-4. Caso de Uso RegistrarUsuario del Sistema de Asignación de Recursos.

Algo muy importante para el correcto funcionamiento del sistema, es que una vez que se registró el Hospital, el Administrador del SAR tiene que descargar el documento XML (DTD) con la definición de la estructura de información de ese hospital. Para lograrlo, el administrador del SAR debe activar un cliente seguro para la transmisión de datos, indicándole la URL del Hospital (DescargarDTD en la figura 3-5).

Nombre del Caso de Uso:	DescargarDTD
Actores participantes:	Administrador del SAR
Condición inicial:	1. El Administrador del SAR se conecta al Hospital por medio de un cliente seguro.
Flujo de eventos:	2. El cliente le pide la URL del Hospital del que se quiere descargar la DTD. 3. El Cliente se conecta con el Servidor del Hospital y verifica si el cliente tiene autorización para descargar la DTD. 4. El Servidor le envía al cliente la DTD.

Figura 3-5. Caso de Uso DescargarDTD del Sistema de Asignación de Recursos.

Una vez hecho lo anterior, el Administrador del SAR puede activar al AGR (InicializarAGR en la figura 3-6), para que el sistema funcione correctamente. Es necesario que este proceso se realice cada 5 minutos, para garantizar que la información contenida en la Base de Datos del AGR sea congruente con la de las Bases de Datos de los Hospitales.

Nombre del Caso de Uso:	InicializarAGR
Actores participantes:	Administrador SAR
Condición inicial:	1. El AGR lee la DTD del Hospital.
Flujo de eventos:	2. Almacena la información en su Base de Datos. 3. El AGR formula una consulta SQL para obtener los servicios que ofrece el hospital y su disponibilidad y guarda esta información en su base de datos.

Figura 3-6. Caso de Uso InicializarAGR del Sistema de asignación de Recursos.

Por último, el administrador del Hospital lo único que tiene que hacer para que se pueda dar la comunicación entre el SAR y el Hospital es crear un documento XML (DTD) que contenga la estructura de información del Hospital. Para facilitarles esta actividad, se puede utilizar la interfaz

CrearDTD en la figura 3-7. La DTD obtenida debe guardarse en donde se encuentre activo el servidor seguro para la transferencia de archivos.

Nombre del Caso de Uso:	CrearDTD
Actores participantes:	Administrador del Hospital
Condición inicial:	1. El Administrador del Hospital activa la opción de Crear DTD.
Flujo de eventos:	2. El sistema le presenta un formulario. 3. El Administrador del Hospital llena el formulario. 4. El sistema crea una DTD con la información que ingreso el Administrador del Hospital.

Figura 3-7. Caso de Uso CrearDTD del Sistema de Asignación de Recursos.

De acuerdo con los Casos de Uso identificados, se pueden presentar los Diagramas de Casos de Uso del sistema en las figuras 3-8 y 3-9.

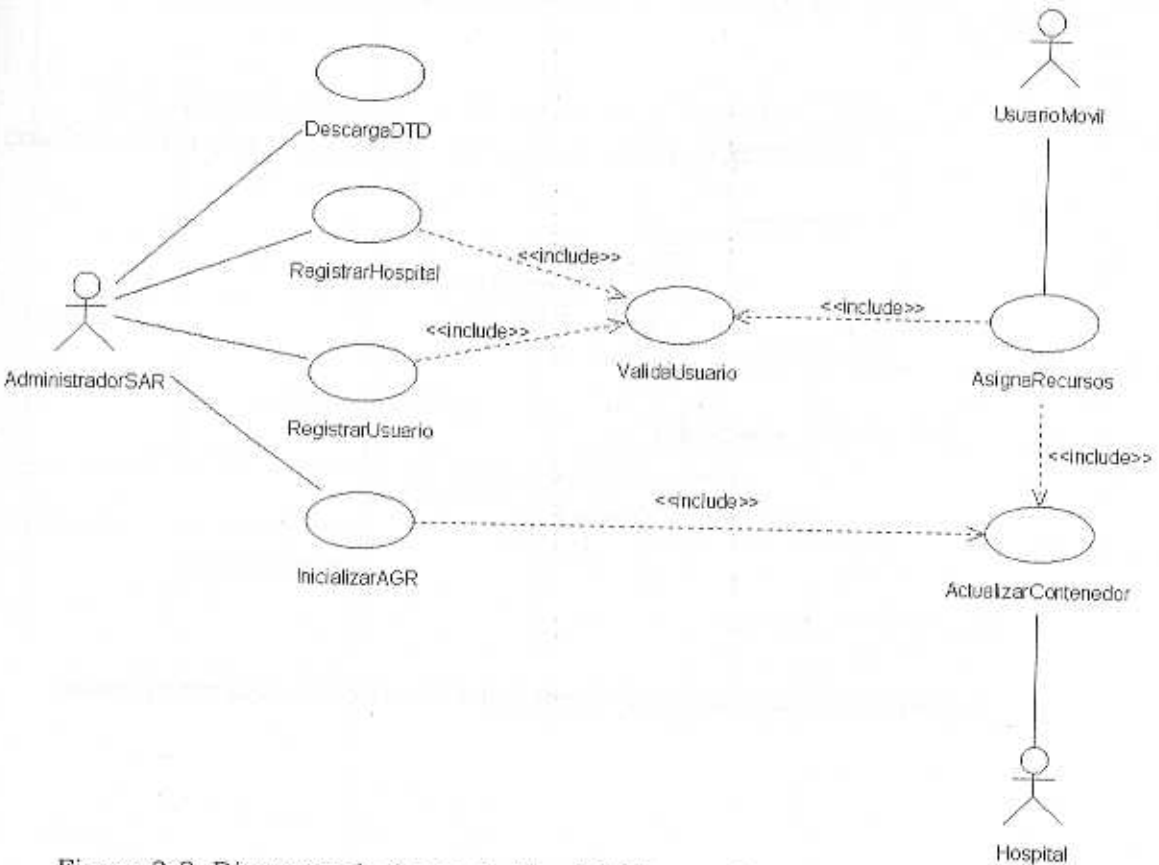

Figura 3-8. Diagrama de Casos de Uso del Sistema de Asignación de Recursos.

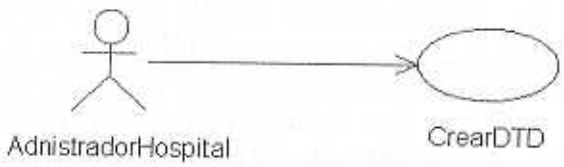

Figura 3-9. Diagrama del Caso de Uso CrearDTD.

Debido a que los diagramas de secuencia y colaboración expresan el flujo en el escenario de un caso de uso en términos de las clases que eventualmente los implementarán se muestran estos diagramas correspondientes al caso de uso Asigna Recursos en las figuras 3-10 y 3-11 respectivamente.

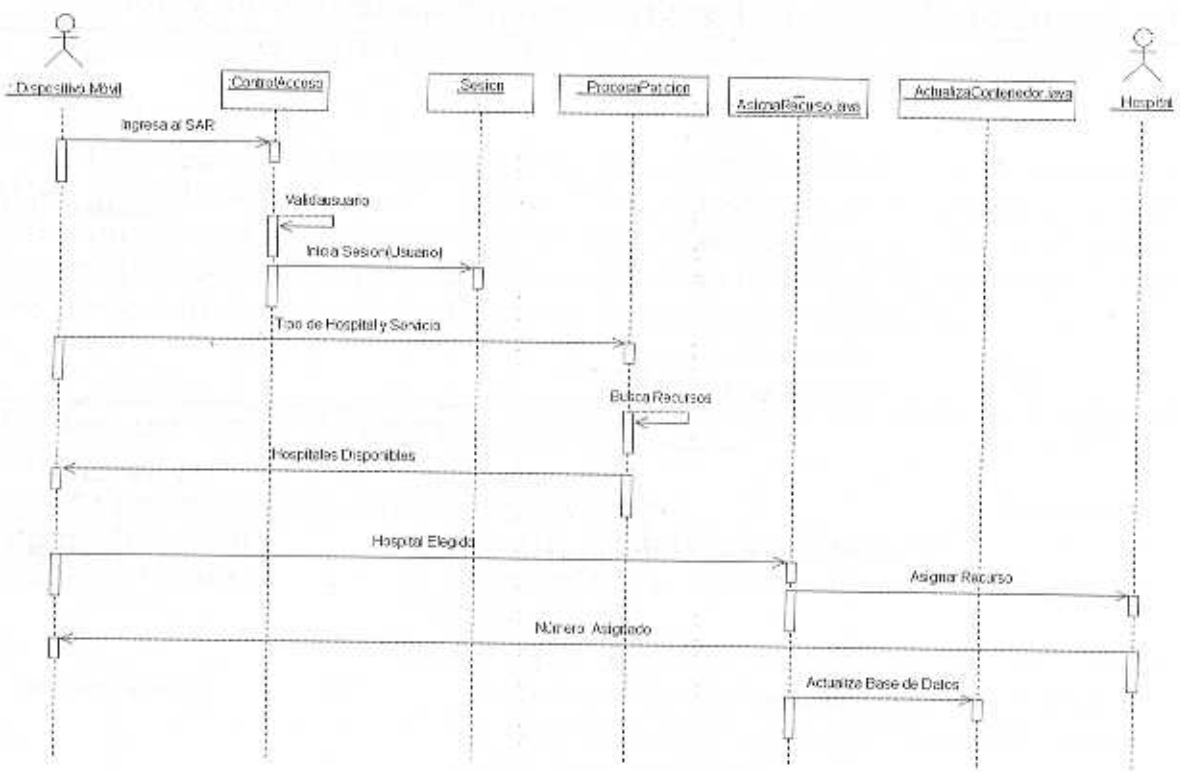

Figura 3-10. Diagrama de Secuencia del Caso de Uso AsignaRecursos.

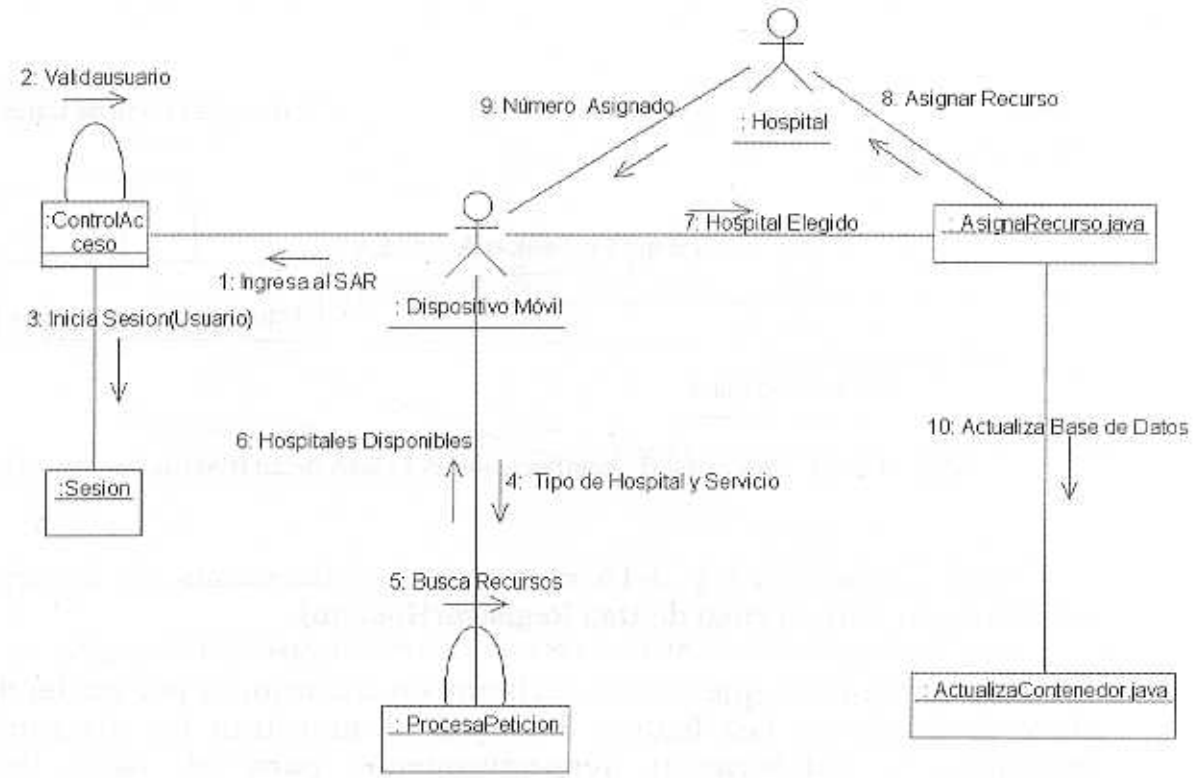

Figura 3-11. Diagrama de colaboración del Caso de Uso AsignaRecursos.

Las figuras 3-12 y 3-13 muestran los diagramas de secuencia y colaboración para el caso de uso ActualizaContenedor.

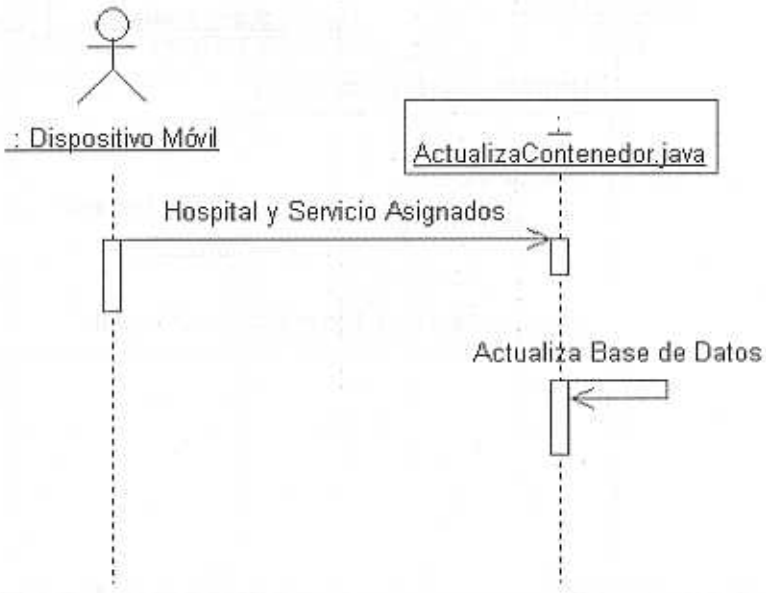

Figura 3-12. Diagrama de Secuencia del Caso de Uso Actualizacontenedor.

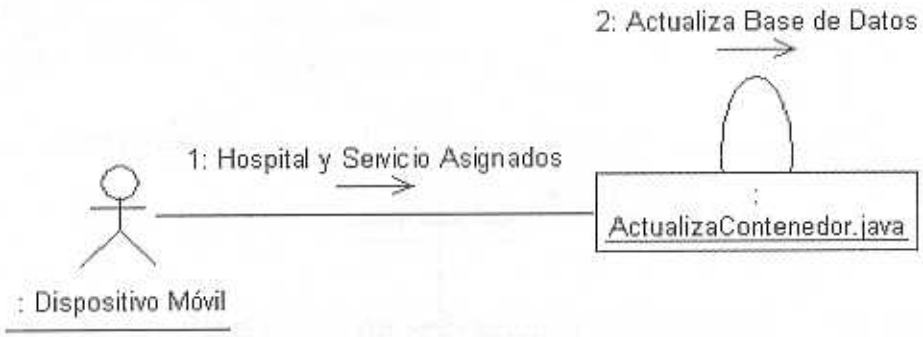

Figura 3-13. Diagrama de colaboración del Caso de Uso Actualizacontenedor.

Las figuras 3-14 y 3-15 muestran los diagramas de secuencia y colaboración para el caso de uso RegistrarHospital.

La forma en la que se va a registrar un Usuario, es por medio de una clave de 8 dígitos. Las figuras 3-16 y 3-17 muestran los diagramas de secuencia y colaboración respectivamente para el caso de uso RegistraUsuario.

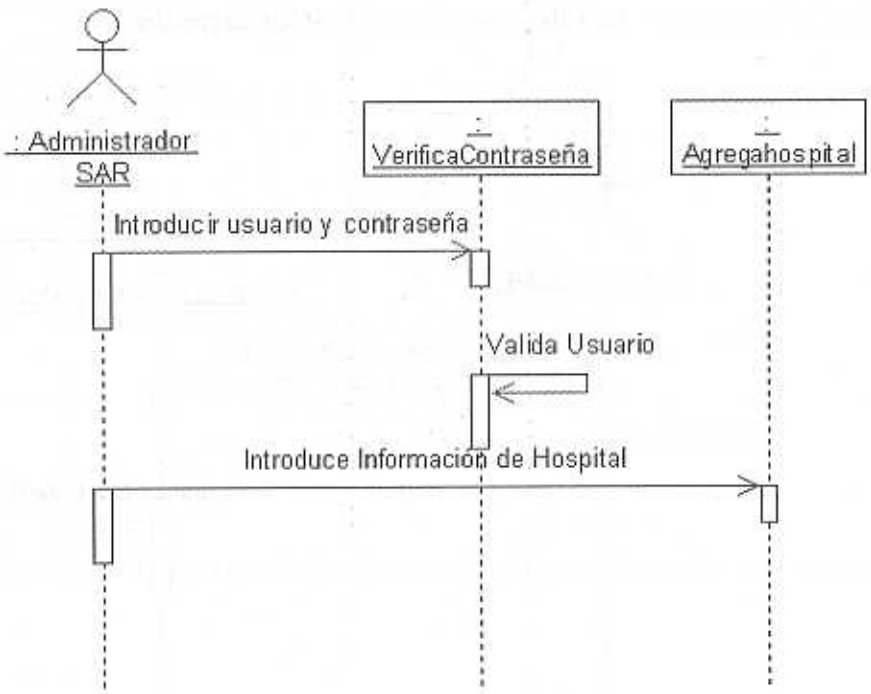

Figura 3-14. Diagrama de Secuencia del Caso de Uso RegistrarHospital.

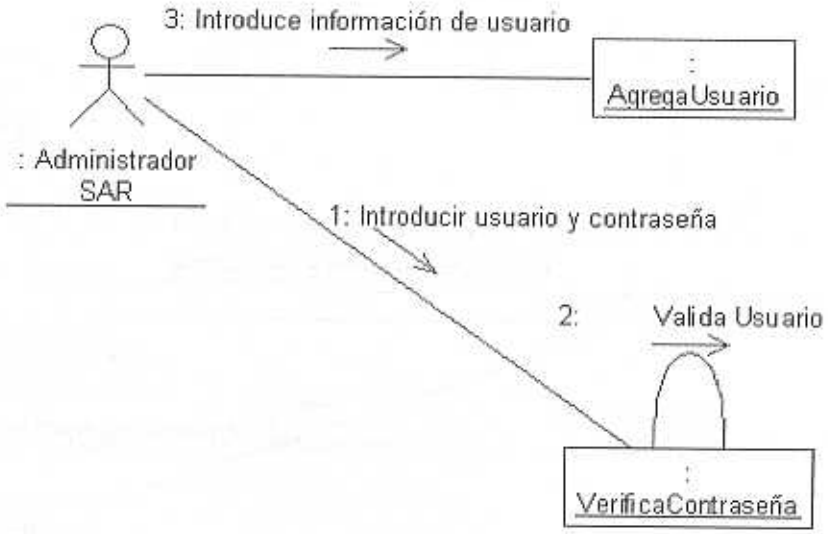

Figura 3-17. Diagrama de Colaboración del Caso de Uso RegistrarUsuario.

Las figuras 3-18 y 3-19 muestran los diagramas de secuencia y colaboración para el caso de uso DescargarDTD.

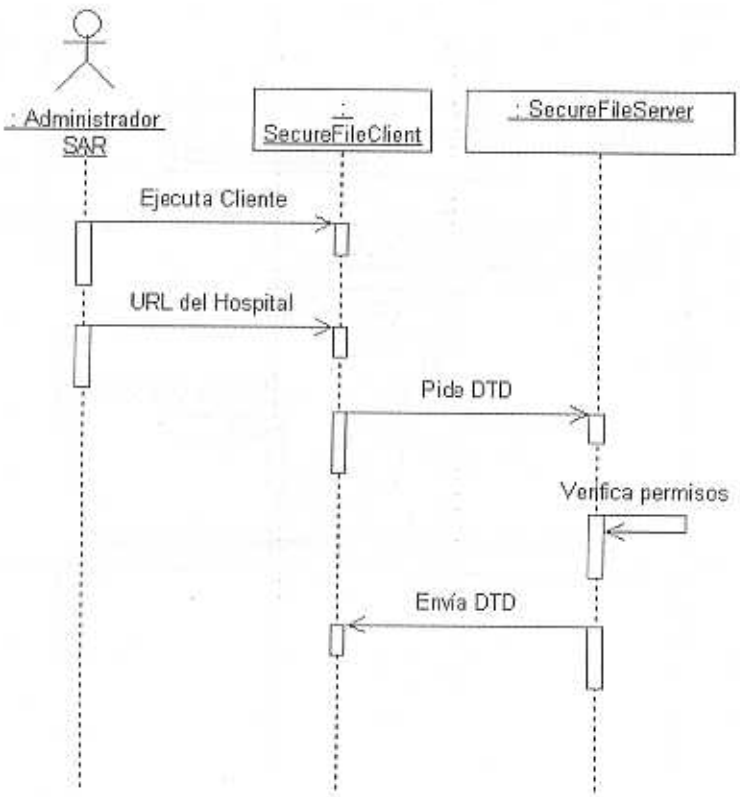

Figura 3-18. Diagrama de secuencia del Caso de Uso DescargarDTD.

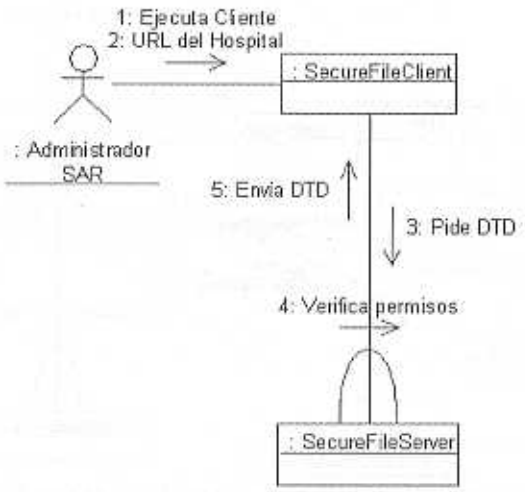

Figura 3-19. Diagrama de colaboración del Caso de Uso descargarDTD.

Las figuras 3-20 y 3-21 muestran los diagramas de secuencia y colaboración para el caso de uso InicializarAGR.

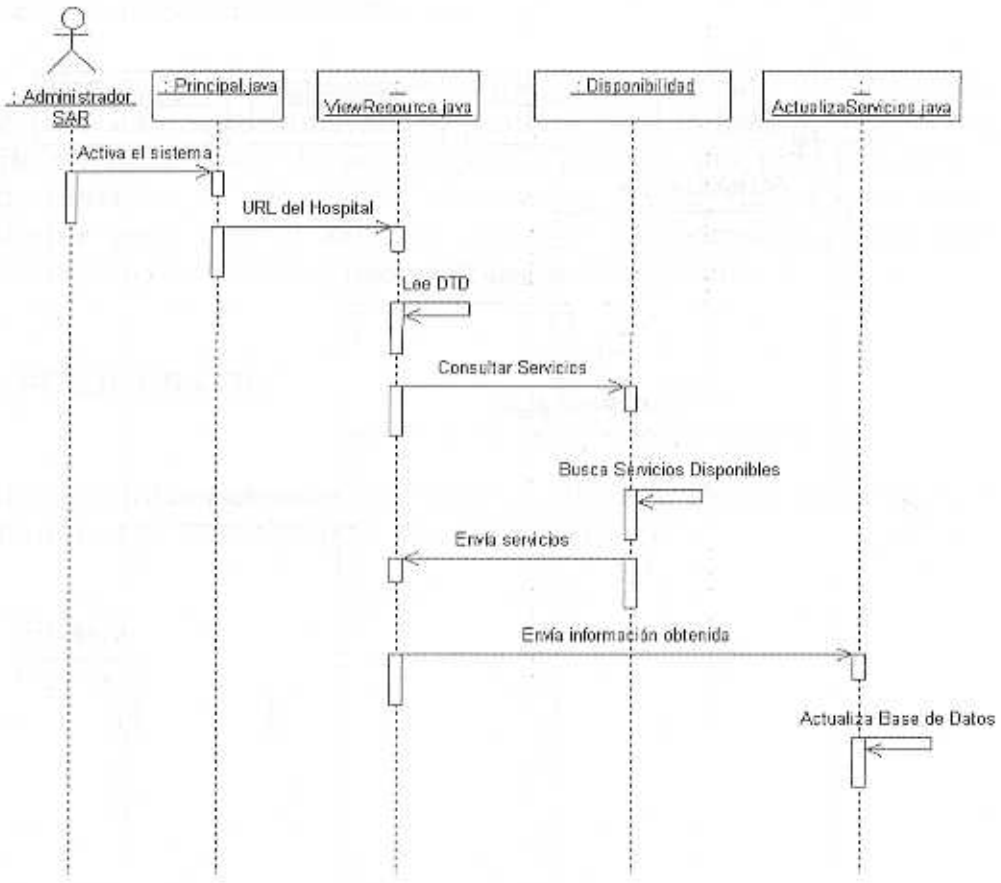

Figura 3-20. Diagrama de Secuencia del Caso de Uso InicializarAGR.

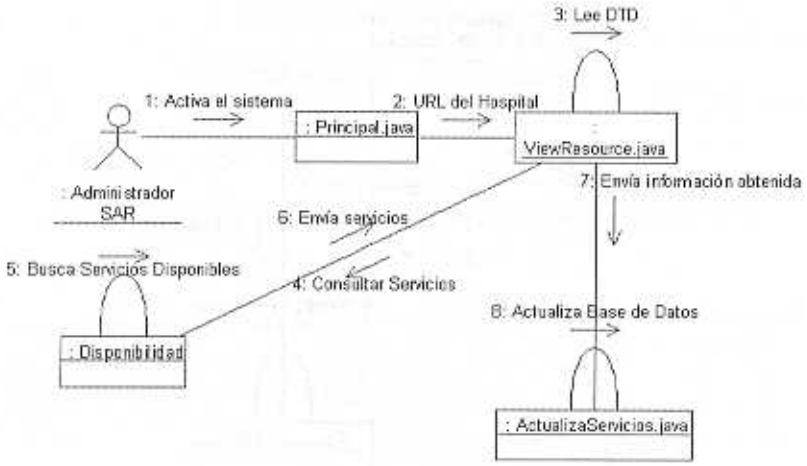

Figura 3-21. Diagrama de Colaboración del Caso de Uso InicializarAGR.

Las figuras 3-22 y 3-23 muestran los diagramas de secuencia y colaboración para el caso de uso CrearDTD.

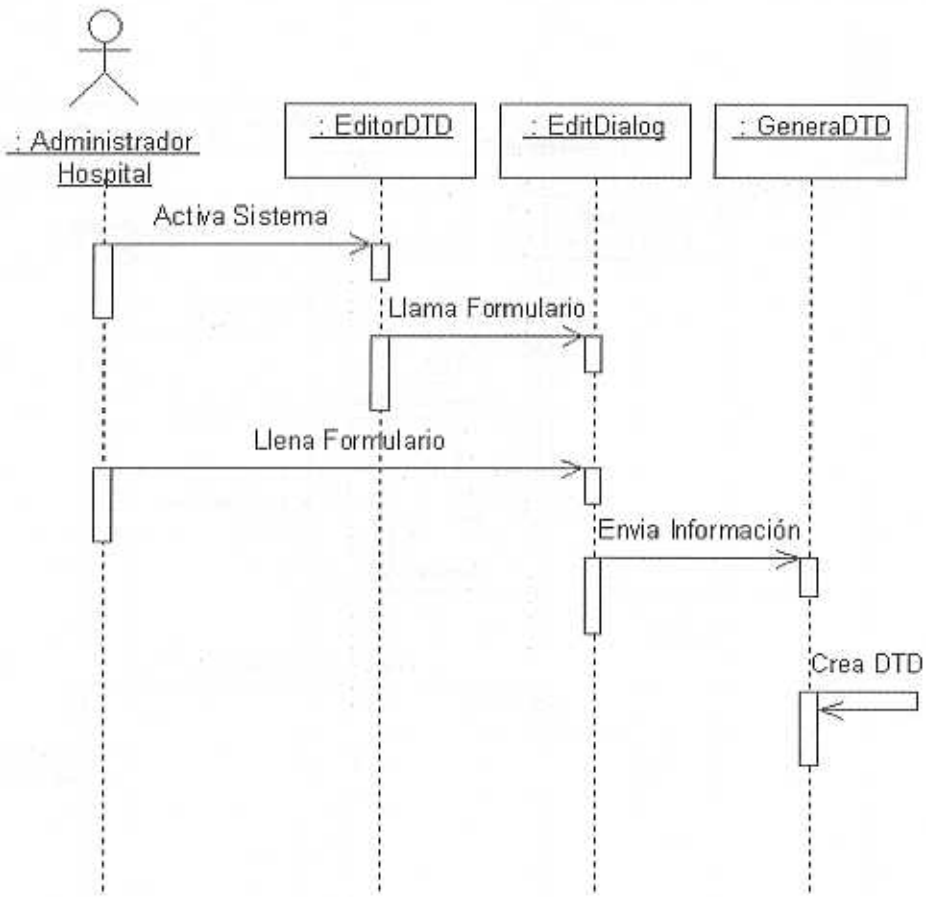

Figura 3-22. Diagrama de secuencia del Caso de Uso CrearDTD.

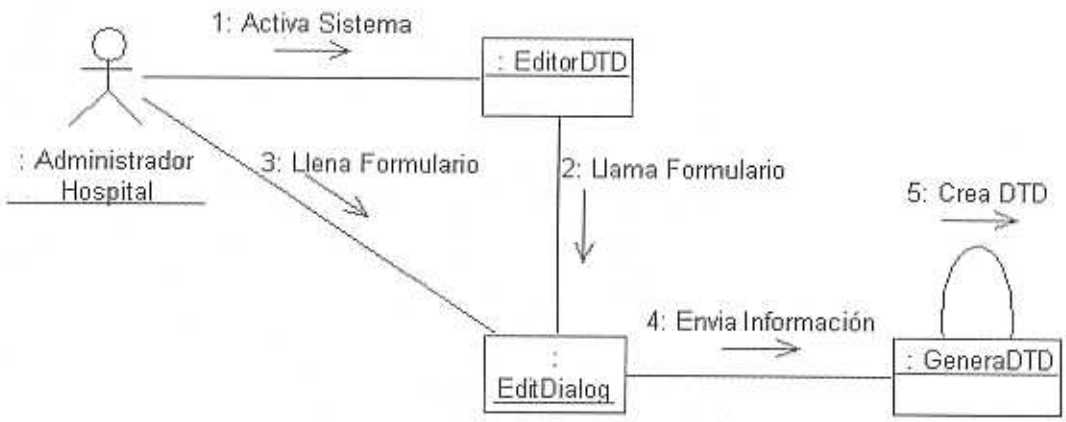

Figura 3-23. Diagrama de secuencia del Caso de Uso CrearDTD.

3.6. CONCLUSIONES

En este capítulo, se obtuvo el modelo de análisis con el cual se podrá realizar el diseño del sistema.

El modelo de análisis se obtuvo identificando los casos de uso, de acuerdo a los requerimientos del sistema, se analizaron sus escenarios por medio de diagramas de secuencia y colaboración para encontrar nuevos requerimientos e identificar objetos y clases de objetos que puedan colaborar para que el sistema tenga el comportamiento deseado. En el siguiente capítulo se realizará el diseño del sistema.

3.7. BIBLIOGRAFÍA

[1] Ingeniería de Software Orientado a Objetos. Bernd Bruegge, Allen H. Dutoit. Ed. Prentice Hall. México, 2002.

Capítulo

4

Diseño

Resumen

En este capítulo, se transformará el modelo de análisis en un modelo de diseño del sistema. Este capítulo, se enfoca en los procesos, estructuras de datos, y componentes de software y hardware necesarios para implementar el sistema.

4.1. INTRODUCCIÓN

En este capítulo, se transformará el modelo de análisis en un modelo de diseño del sistema. Este capítulo, se enfoca en los procesos, estructuras de datos, y componentes de software y hardware necesarios para implementar el sistema. Al finalizar el diseño se tendrá una arquitectura de software que describa la descomposición en subsistemas desde el punto de vista de responsabilidades del subsistema, dependencias entre subsistemas, correspondencia de los subsistemas con el hardware y decisiones de políticas principales, como el flujo de control, control de acceso y almacenamiento de datos.

4.2. INTERFAZ DE USUARIO [1]

Para el diseño de la interfaz de usuario, se deben tener varias cosas en mente. Por ejemplo, WML puede ser interpretado de varias formas por navegadores de distintos fabricantes. Provocando que la misma aplicación se despliegue de diferentes maneras en cada teléfono.

Se tienen algunas limitaciones como los retardos en la red, que se empiezan a hacer evidentes cuando superan los 500 bytes pudiendo pasar de 10 segundos. Este tiempo aunque parece corto, se acentúa debido a que a diferencia de una PC, un teléfono celular es un dispositivo de una sola tarea, por lo que el usuario no puede realizar otra actividad mientras espera a que se descargue una página. Para lograr resultados uniformes, se recomienda mantener el tamaño de las páginas compiladas por debajo de los 500 bytes.

Otras limitaciones son las pantallas, por lo regular resultan visibles hasta cuatro líneas de texto, aunque hay dispositivos que sólo muestran dos líneas y algunos otros hasta ocho. El texto de los botones de función puede visualizarse directamente sobre sus referentes. Si el dispositivo los visualiza, podría limitar su longitud a cinco caracteres. En cada línea, se pueden visualizar aproximadamente 12 caracteres. Esta cifra puede variar ya que la mayoría de los teléfonos soportan fuentes de distinto ancho.

Se recomienda incluir una función de retroceso en las aplicaciones debido a que los usuarios tienden a recurrir a ésta para salir de una aplicación. Si no resulta apropiado volver a una página anterior, se asocia la función de retroceso al siguiente menú en importancia o al más intuitivo.

El sistema contará con 5 cartas:

1. En la primera carta ("Clave", mostrada en la figura 4-1), se le pide al usuario que introduzca su clave de acceso, la cual es numérica de 4 dígitos.
2. Si el usuario esta registrado, el sistema le mostrará la carta "Tipo" (mostrada en la figura 4-2). El usuario elegirá el tipo de hospital al que quiere acceder, (IMSS, ISSTE, Cruz Roja o privado) mediante las flechas de cursor y la tecla *select*.
3. Cuando seleccione el tipo de hospital, se le presentará al usuario la carta "Servicio" (mostrada en la figura 4-3) para elegir el tipo de servicio que requiere (Urgencias, Quemaduras, Traumatismo, etc.).
4. Una vez que selecciona la especialidad, el sistema realiza una búsqueda con el tipo y especialidad enviados. Si el sistema encontró uno o más hospitales que coincidan con los parámetros de búsqueda, se los enviará al usuario. En esta carta (figura 4-4), el usuario podrá elegir un hospital para realizar la reservación.
5. Una vez elegido el hospital, el sistema se comunica con éste para hacer la reservación y obtener el número de identificación (figura 4-5), con el cual se registrará al paciente cuando llegue al hospital.

```
- <card id="Clave" title="Clave:" newcontext="false" ordered="true">
  - <do type="accept" label="Valida" optional="false">
      <go href="ValidaClave.wmls#validaClave($(Clave))" sendreferer="false"
        method="get" />
    </do>
  - <p align="left">
      Introduzca clave de acceso:
      <input type="text" name="Clave" emptyok="false" />
    </p>
  </card>
```

Figura 4-1. Código y vista de la carta "Clave"

```
- <card id="index" title="Seleccione tipo" newcontext="false" ordered="true">
  - <onevent type="onenterforward">
    - <refresh>
        <setvar name="tipo" value="" />
      </refresh>
    </onevent>
  - <do type="options" optional="false">
      <noop />
    </do>
  - <p align="left">
    - <select name="tipo" multiple="false">
        <option onpick="#serv" value="IMSS">IMSS</option>
        <option onpick="#serv" value="ISSTE">ISSTE</option>
        <option onpick="#serv" value="Cruz Roja">Cruz Roja</option>
        <option onpick="#serv" value="Privado">Privado</option>
        <option onpick="#serv" value="Salubridad">Salubridad</option>
      </select>
    </p>
  </card>
```

Figura 4-2. Código y vista de la Carta "Seleccione Tipo"

```
- <card id="serv" title="Seleccione especialidad" newcontext="false" ordered="true">
  - <onevent type="onenterforward">
    - <refresh>
        <setvar name="servicio" value="" />
      </refresh>
    </onevent>
  - <p align="left">
    - <select name="servicio" multiple="false">
        <option onpick="#env" value="Urgencias">Urgencias</option>
        <option onpick="#env" value="Quemaduras">Quemaduras</option>
        <option onpick="#env" value="Traumatismo">Traumatismo</option>
        <option onpick="#env" value="Ginecologia">Ginecologia</option>
        <option onpick="#env" value="General">General</option>
      </select>
    </p>
  </card>
```

Figura 4-3. Código y vista de la Carta "Seleccione Especialidad"

```
<card id="resp" title="Seleccione Hospital">
 <onevent type="onenterforward">
  <refresh>
   <setvar name="Servicio" value="General" />
  <setvar name="Hospital" value="" />
  </refresh>
 </onevent>
 <p>
  <select name="Hospital">
   <option onpick="#sel" value="Clinica Guadalupe Tepeyac S.A. de C.V.">Clinica Guadalupe
                                                                Tepeyac S.A. de C.V.</option>
   <option onpick="#sel" value="Hospital MIG S.A. de C.V.">Hospital MIG S.A. de C.V.</option>
  </select>
 </p>
</card>
```

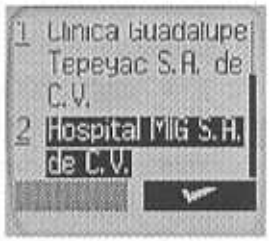

Figura 4-4. Código y vista de la Carta "Seleccione Hospital"

```
- <card id="asigna" title="Identificador" newcontext="false" ordered="true">
  - <onevent type="onenterforward">
    - <refresh>
        <setvar name="Ident" value="1234" />
      </refresh>
    </onevent>
    <p align="left">No. asignado: $(ident)</p>
  - <do type="accept" label="Aceptar" optional="false">
      <go href="http://148.204.211.194:8080/wap/ActualizaContenedor"
        sendreferer="false" method="get" enctype="application/x-www-form-urlencoded" />
    </do>
  </card>
```

Figura 4-5. Código y vista de la Carta "Identificador"

4.3. ADMINISTRACIÓN DE DATOS

Esta sección se enfoca en el diseño de la base de datos del AGR y en el diseño de la DTD, la cual contendrá la información relevante para que se pueda realizar la comunicación con cada hospital. Además se mostrará el diseño de una Base de Datos que se utilizará como prueba en el sistema.

La presentación será de la siguiente manera, primero se mostrará el diseño de las Base de Datos de los Hospitales con los que trabajará inicialmente el sistema. Una vez conociendo estas bases de datos, se

conocerá que información es relevante para el sistema y se podrá obtener la estructura de la DTD. Posteriormente, se realizará el diseño de la base de datos del AGR.

4.3.1 Bases de datos de Prueba

El diseño de las Bases de Datos de los Hospitales puede variar, pero hay cierta información que siempre se debe manejar en un Hospital.

En base a esto, se distinguen las siguientes entidades con sus atributos para una Base de datos de un hospital.

- Paciente
  - No_Filiacion
  - Nombre
  - Apellidos
  - Domicilio
  - Teléfono
  - Pago
- Servicio
  - Id_Serv
  - Nom_Serv
  - Camas
- Ingreso
  - Id_Ingreso
  - No_Cama
  - F_ingreso
  - F_Salida
- Personal
  - Id_Pers
  - Nombre_Personal
  - Apellidos_Personal
  - Especialidad

A continuación se muestra el diagrama de Entidad Relación (Figura 4-6)

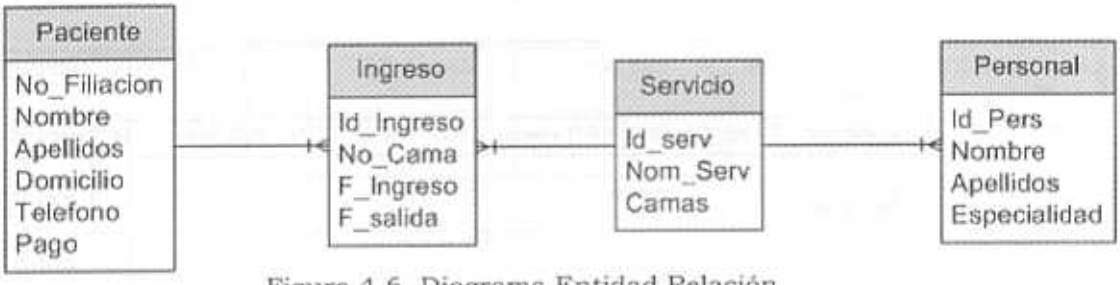

Figura 4-6. Diagrama Entidad Relación.

En el diagrama Entidad Relación (mostrado en la figura 4-6), se puede observar que un Paciente puede ingresar una o más veces a un hospital. Un Servicio puede aparecer una o más veces en un Ingreso y por cada servicio puede estar asignada más de una persona (médico, enfermera, etc.)

Una vez que se tiene el diseño lógico de la base de datos, se puede realizar el diseño de las tablas. Cada entidad se convertirá en una tabla. Las columnas en la base de datos corresponden directamente a los atributos del diseño lógico. A cada columna se le debe asignar un tipo de datos. Las llaves primarias de cada tabla son las siguientes:

- No_Filiacion
- Id_Ingreso
- Id_Serv
- Id_Pers

Mediante estos campos se identifican de manera única cada registro en cada tabla. Como se puede ver en la figura 4-6 todas las tablas están relacionadas, para lograr esto fisicamente se necesitan las llaves foráneas. Para cada relación identificada en el modelo lógico (figura 4-6), se debe agregar una llave foránea en la tabla que forma parte de la relación y ésta es la llave primaria de la otra tabla que participa en la relación. Las tablas de la Base de Datos se muestran en la figura 4-7.

En la tabla 4-1 se muestra el diccionario de datos para esta Base de datos en el que se incluyen los tipos, longitud y a que tabla pertenece cada campo.

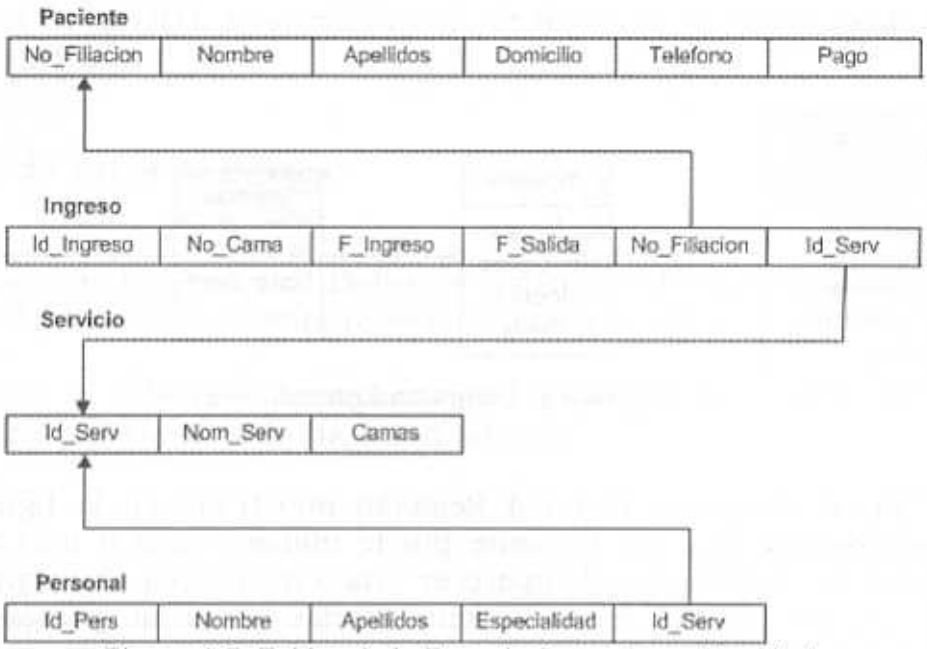

Figura 4-7. Tablas de la Base de Datos de un Hospital.

Tabla 4-1. Diccionario de Datos de la Base de Datos Hospital.

NOMBRE	TIPO	LONGITUD	TABLA	DESCRIPCIÓN
No_Filiación	Texto	14	Paciente e Ingreso	Identificador único de paciente
Nombre	Texto	15	Paciente	Nombre del paciente
Apellidos	Texto	30	Paciente	Apellidos del paciente
Domicilio	Texto	50	Paciente	Domicilio del paciente
Teléfono	Numérico	8	Paciente	Teléfono del paciente
Pago	Texto	15	Paciente	Forma de pago del Paciente
Id_Serv	Texto	30	Servicio, Ingreso y Personal	Identificador único del tipo de servicio
Nom_Serv	Texto	30	Servicio	Nombre del servicio
Camas	Numérico	3	Servicio	Cantidad de camas disponibles por cada tipo de servicio
Costo	Moneda	Automático	Servicio	Costo del servicio por día
ID_Ingreso	Numérico	Automático	Ingreso	Identificador único del ingreso de un paciente al hospital

Tabla 4-1(Continuación). Diccionario de Datos de la Base de Datos Hospital.

NOMBRE	TIPO	LONGITUD	TABLA	DESCRIPCIÓN
No_Cama	Numérico	3	Ingreso	Cama asignada al paciente
F_Ingreso	Fecha	Mediana	Ingreso	Fecha en la que ingreso el paciente al hospital
F_Salida	Fecha	Mediana	Ingreso	Fecha en la que salio el paciente del hospital
Id_Pers	Texto	30	Personal	identificador único del personal
Nombre	Texto	15	Personal	Nombre del empleado
Apellidos	Texto	30	Personal	Apellidos del empleado
Especialidad	Texto	30	Personal	Especialidad del empleado

4.3.2. Diseño del almacén de datos en XML

Es necesario compartir la información contenida en las bases de datos de cada servidor. Pero éstas pueden estar almacenadas en distintos RDBMS. Por esto es necesario crear estructuras XML que permitan lograr la completa interoperabilidad entre bases de datos diferentes.

Para lograrlo, se tiene que crear una DTD XML que represente la estructura de la base de datos con la información que interesa. Teniendo el ejemplo de la Base de Datos anterior, se puede observar que para lograr el propósito del sistema no se necesita saber con que personal cuenta el hospital ni su información. Tampoco es necesario conocer la información del paciente. Básicamente lo que se quiere saber es con que servicios cuenta el hospital y el número de camas disponibles para atender a los pacientes. También es necesario saber cómo administran sus recursos.

Analizando las tablas presentadas en la sección anterior, resultan interesantes dos: Servicio e Ingreso. De Servicio es importante todo su contenido y de Ingreso sólo se necesita conocer el Id_Ingreso y el Id_serv para poder indicarle a ese Hospital que se ha asignado uno de sus recursos. Los campos restantes pueden llenarse siguiendo la lógica de cada Hospital.

Por lo tanto, en las Bases de Datos de los Hospitales se debe buscar una tabla que contenga la información de los servicios que se ofrecen y su disponibilidad, y una tabla en la que se registren los pacientes.

Para lograr esto, es necesario que cada hospital construya una DTD. En ésta se debe indicar el nombre de las tablas y los atributos relevantes. Además, es necesario conocer el tipo de RDBMS y su controlador JDBC específico. Además, se debe conocer el nombre de la Base de Datos, un nombre de usuario y una contraseña con la que se permita el acceso a la Base de Datos. Además también se deben conocer los datos de identificación del Hospital como su nombre, su especialidad, teléfono, domicilio y qué tipo de Hospital es (público, privado o institución a la que pertenece).

Con esto en mente, se puede construir una DTD general siguiendo los siguientes pasos:

1. Crear un elemento raíz para el documento y declarar qué atributos de ese elemento son requeridos para manejar información adicional. Básicamente son 3 Hospital, Servicio e Ingreso. El elemento que los va a contener es Disponibilidad.

2. Modelar las tablas de contenido, es decir, crear un elemento en la DTD para cada tabla de la base de datos que se haya elegido para el modelo. Estos elementos son Servicio e Ingreso los cuales se declaran como EMPTY, y Hospital que contiene el elemento Domicilio.

3. Se crea un atributo para cada columna que se haya decidido incluir en el documento XML. Estos atributos deben aparecer en la declaración !ATTLIST del elemento correspondiente de la tabla en la cual aparecen. Cada uno de estos elementos deben declararse como CDATA, y declararlos como #FIXED para evitar que alguien los cambie y se produzcan inconsistencias en el sistema.

4. Se crean atributos para almacenar la información relacionada con el Hospital y que no está contenida en su Base de Datos. Nombre, especialidad, teléfono, tipo, domicilio, manejador y URL.

Partiendo de la Base de Datos de prueba mostrada en la sección anterior y siguiendo los pasos mencionadas anteriormente, se obtiene la estructura mostrada en la figura (4-8).

Toda la información que se muestra entre comillas en la figura 4-8 es la información que va a cambiar con cada Hospital.

```
<!ELEMENT Disponibilidad (Hospital, Servicio, Ingreso)>
<!ELEMENT Hospital (Domicilio)>
<!ATTLIST Hospital
      nombre CDATA #FIXED "Clinica Guadalupe Tepeyac S.A. de C.V."
      especialidad CDATA  #FIXED "General"
      telefono CDATA  #FIXED "55772722"
      tipo CDATA  #FIXED "Privado"
      JDBCDriver CDATA #FIXED "sun.jdbc.odbc.JdbcOdbcDriver"
      JDBCURL CDATA #FIXED "jdbc:odbc:Hospital">
<!ELEMENT Domicilio EMPTY >
<!ATTLIST Domicilio
    direccion CDATA  #FIXED "Av. Ticoman 128"
      localidad CDATA  #FIXED "Lindavista"
      delegacion CDATA  #FIXED "Gustavo A. Madero">
<!ELEMENT Servicio EMPTY>
<!ATTLIST Servicio
      nombre_tabla CDATA #FIXED "Servicio"
      nombre_id CDATA #FIXED "Id_Serv"
      nombre_serv CDATA #FIXED "Nom_Serv"
      camas_disp CDATA #FIXED "Camas">
<!ELEMENT Ingreso EMPTY>
<!ATTLIST Ingreso
      nombre_tabla CDATA #FIXED "Ingreso"
      nombre_id CDATA #FIXED "Id_Ingreso"
      nombre_serv CDATA #FIXED "Id_Serv"
      Cama_asig CDATA #FIXED "No_Cama">
```

Figura 4-8. DTD que representa la estructura de la información necesaria de la Base de Datos del Hospital ejemplo.

4.3.3. Base de Datos del Administrador Global de Recursos

El AGR necesita una Base de Datos, para almacenar los datos generales de los hospitales y de los servicios que prestan para realizar búsquedas más eficientes. Como ya se tienen almacenados los servicios que presta cada hospital con su disponibilidad[1], no es necesario que se busque en cada hospital, únicamente en aquellos que tengan mayor probabilidad de aceptar a un paciente. Con esto se reduce el tráfico en la red y además se obtienen los beneficios de encapsular el código relacionado con las Bases de datos en un solo lugar (Seguridad, fácil mantenimiento, etc.).

Las entidades con sus atributos se muestran a continuación.

- Hospital
  - URL
  - Nom_Hosp
  - Tipo
  - Especialidad
  - Dirección

[1] La disponibilidad se actualiza cada 5 minutos. Este proceso se explica en la sección 2.2.3.

- Teléfono

- Servicios
  - ID_Serv
  - Nom_Serv
  - Camas

- Base Datos
  - JDBCURL
  - JDBCDriver
  - Tabla_serv
  - Id_serv
  - Nombre_serv
  - Camas_disp
  - Tabla_ing
  - Id_ing

- Usuarios
  - Clave
  - Usuario

En la figura 4-9 se muestra el diagrama entidad relación del Administrador Global de Recursos.

En el diagrama Entidad Relación (mostrado en la figura 4-9), se puede observar que cada Hospital tiene sólo una Base de Datos, y que a su vez cada Hospital puede prestar uno o más Servicios. También se observa que la tabla Usuarios no se relaciona con ninguna tabla, esto es porque esta tabla sirve para tener un control en el acceso de los usuarios. Si alguien quiere utilizar el sistema y no aparece en la tabla Usuarios se le negará el acceso.

Una vez que se tiene el diseño lógico de la base de datos, se puede realizar el diseño de las tablas. Cada entidad se convertirá en una tabla. Las columnas en la base de datos corresponden directamente a los atributos del diseño lógico. A cada columna se le debe asignar un tipo de datos. Las llaves primarias de cada tabla son las siguientes:

- JDBCURL
- URL
- ID_Serv
- Clave

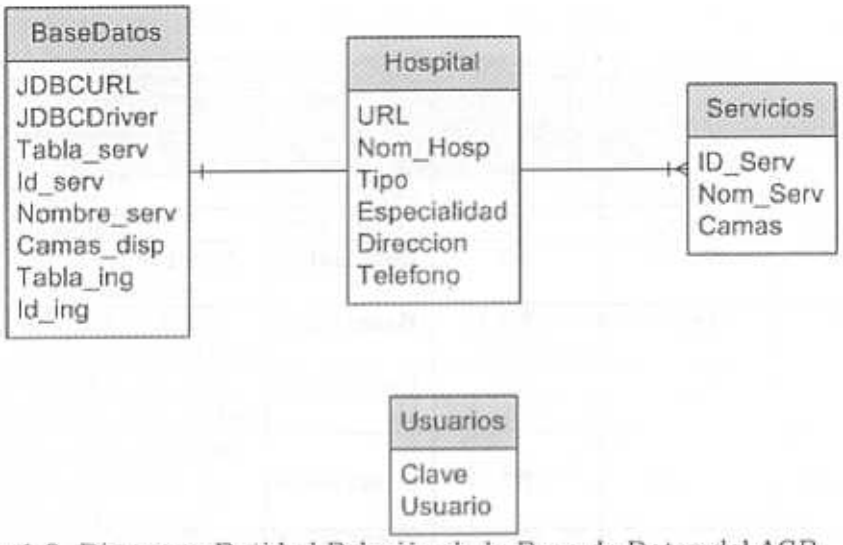

Figura 4-9. Diagrama Entidad Relación de la Base de Datos del AGR.

Para cada relación identificada en el modelo lógico (figura 4-9), se debe agregar una llave foránea en la tabla que forma parte de la relación y ésta es la llave primaria de la otra tabla que participa en la relación. Las tablas de la Base de Datos se muestran en la figura 4-10.

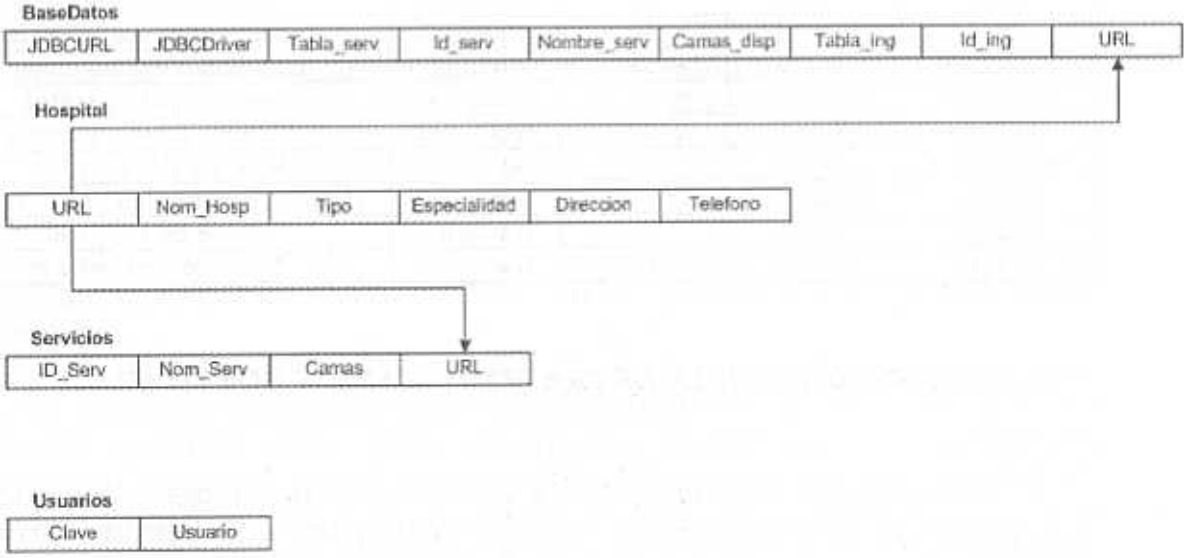

Figura 4-10. Tablas de la Base de Datos del AGR.

En la tabla 4-2 se muestra el diccionario de datos para esta Base de datos en el que se incluyen los tipos, longitud y a que tabla pertenece cada campo.

Tabla 4-2. Diccionario de Datos de la Base de Datos del AGR.

NOMBRE	TIPO	LONGITUD	TABLA	DESCRIPCIÓN
JDBCURL	Texto	50	BaseDatos	URL JDBC
JDBCDriver	Texto	50	BaseDatos	Maejador JDBC
Tabla_serv	Texto	50	BaseDatos	Nombre de la tabla en la que se encuentran los servicios.
Id_serv	Texto	50	BaseDatos	Nombre del campo de Identificador único de Servicio.
Nombre_serv	Texto	50	BaseDatos	Nombre del campo donde se encuentra el nombre del servicio.
Camas_disp	Texto	50	BaseDatos	Nombre del campo en el que se encuentran las camas disponibles.
Tabla_ing	Texto	50	BaseDatos	Nombre de la tabla en la que se registra el ingreso de los pacientes.
Id_ing	Texto	50	BaseDatos	Nombre del identificador único del registro.
URL	Texto	50	Hospital, Servicios y BaseDatos	URL del Hospital.
Nom_Hosp	Texto	50	Hospital y Servicios	Nombre del Hospital.
Tipo	Texto	15	Hospital	Sistema de salud.
Especialidad	Texto	30	Hospital	Especialidad del hospital.
Dirección	Texto	50	Hospital	Ubicación del hospital.
Teléfono	Numérico	8	Hospital	Teléfono del hospital.
ID_Serv	Texto	30	Servicios	Identificador único del servicio.
Nom_Serv	Texto	30	Servicios	Nombre del servicio.
Camas	Numérico	3	Servicios	Camas disponibles por servicio.
Clave	Texto	8	Usuarios	Clave de Usuario
Usuario	Texto	15	Usuarios	Nombre de Usuario

4.4. CORRESPONDENCIA ENTRE HARDWARE Y SOFTWARE

Este sistema requiere que los teléfonos celulares desde los que se va a ingresar al sistema tengan soporte WAP para que puedan utilizar el sistema.

Se requiere un servidor Web en el cual va a ejecutarse el sistema. Este servidor, puede ejecutarse en un Sistema Operativo Windows o Unix, debido a que la implementación se hará en Java. Para efectos de esta tesis, el Servidor Web se instalará en una máquina con Sistema Operativo Windows NT Server.

Prácticamente el sistema se ejecuta de dos formas. Dentro del Servidor Web, se ejecuta el AGR, el cual se encarga de administrar los recursos de cada Hospital. Además hay un servicio que se encarga de manejar las peticiones provenientes de los teléfonos celulares y asignarles un recurso. En la figura 4-11, se muestra un diagrama de organización UML en el que se muestra la asignación de los subsistemas al hardware.

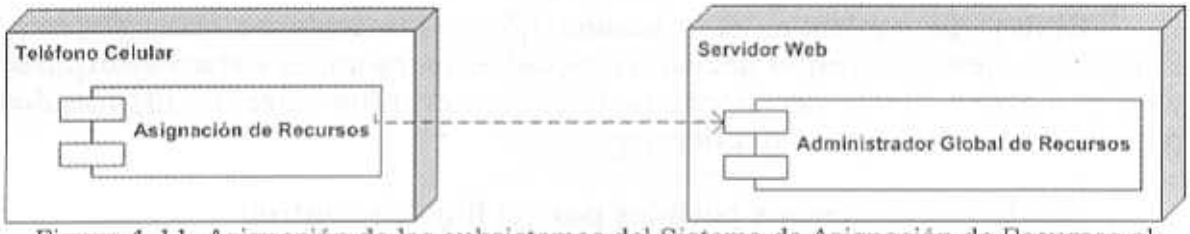

Figura 4-11. Asignación de los subsistemas del Sistema de Asignación de Recursos al hardware.

4.5. CONTROL DE ACCESO

El control de acceso se hace en base a los actores que participan en el sistema, ya que cada actor tiene acceso a funcionalidades y datos diferentes.

Se tienen tres usuarios diferentes. El Administrador del SAR se encarga de agregar y eliminar tanto a los Hospitales como a los usuarios participantes en el sistema. Para que éste le permita realizar las funciones de mantenimiento es necesario que se autentifique como administrador.

El administrador del hospital, tiene acceso únicamente a una interfaz que le permita crear una DTD con la información correspondiente a su hospital.

Por último se tiene al usuario móvil. Éste tiene acceso al SAR, el cual le permite solicitar una cama en algún hospital que cubra ciertos requisitos presentados por él a través del teléfono celular. La forma en la que se controla el acceso a los usuarios móviles es a través de una clave única para cada usuario. Cuando se ingrese al sistema a través de un teléfono celular se le pedirá que ingrese una clave de acceso (4 dígitos). Si es válida, se le permitirá continuar, de lo contrario no podrá ingresar al sistema.

Además el AGR se tiene que autentificar con cada hospital, para que le sea permitido obtener su DTD. Una vez que se lleva a cabo la autentificación, se cifra el contenido de la DTD. Esto se hace para impedir

accesos no autorizados, ya que aunque un intruso intercepte el mensaje, no podrá comprenderlo. Sólo el AGR tiene el conocimiento suficiente para descifrar en forma correcta el mensaje.

4.6. FLUJO DE CONTROL

El flujo de control es el ordenamiento de las acciones en el sistema. Estas acciones incluyen la decisión de cuáles operaciones deben ejecutarse y en qué orden. Estas decisiones se basan en eventos externos provocados por un actor o en el paso del tiempo.

Hay tres mecanismos posibles para el flujo de control:

- Control manejado por procedimientos
- Control manejado por eventos
- Hilos

Para el desarrollo del Sistema, se utilizará el flujo de control manejado por eventos en el que un ciclo principal espera un evento externo. Cada vez que se tiene disponible un evento se le despacha al objeto adecuado con base en la información asociada con el evento. Este tipo de flujo de control tiene la ventaja de conducir hacia una estructura más simple y centralizar toda la entrada en el ciclo principal.

4.7. DIAGRAMAS DE CLASES [2], [3], [4]

Los diagramas de clase que se explicarán en breve, describen los tipos de objetos que hay en el sistema y las diversas clases de relaciones estáticas que existen entre ellos. En estos diagramas también se muestran los atributos y operaciones de una clase y las restricciones a que se ven sujetos, según la forma en la que se conecten los objetos.

La figura 4-12, muestra el diagrama de clases del Administrador Global de recursos.

El servlet Principal se activa cuando el Administrador del SAR activa el AGR por medio de un navegador Web. Este servlet se conecta a la base de datos del AGR por medio de un objeto Connection y busca las URL de los hospitales las cuales almacena en el Contexto y redirige la petición al servlet ViewResource (ver diagrama de secuencia de la figura 4-13).

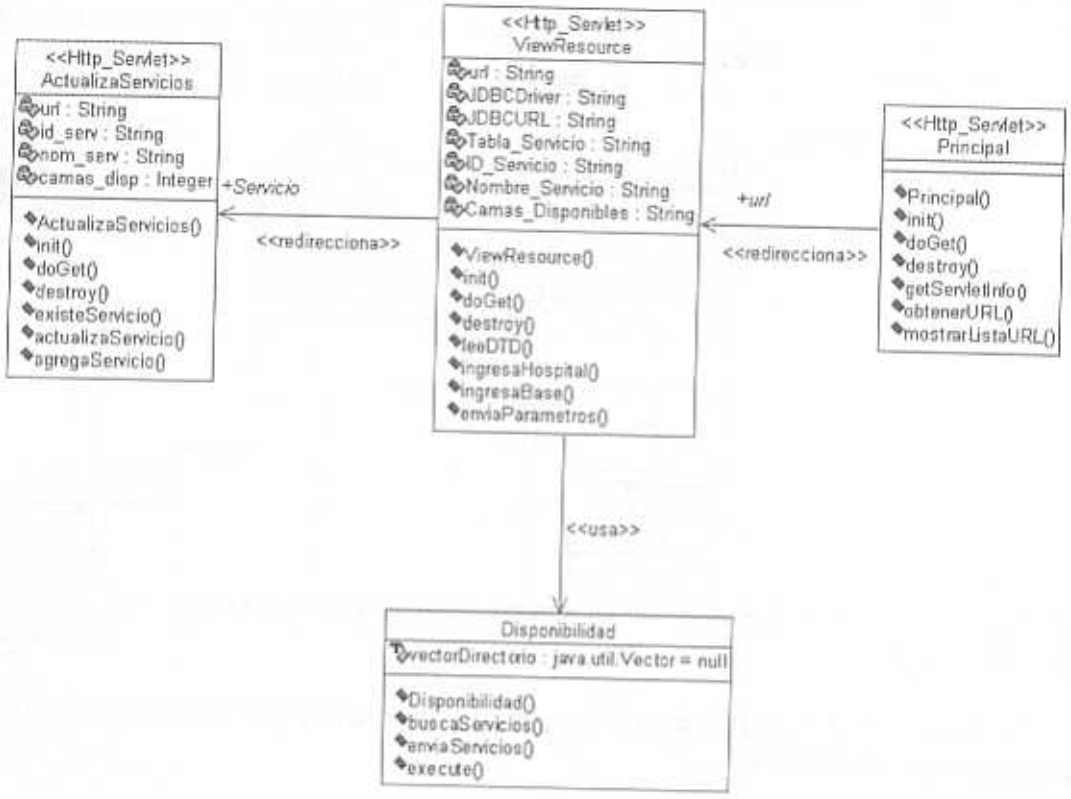

Figura 4-12. Diagrama de clases del Administrador Global de Recursos.

El servlet ViewResource obtiene las direcciones almacenadas en el contexto por el servlet Principal. Con esta información, construye el directorio en el que se encuentran almacenadas las DTD's de los hospitales. Para cada DTD encontrada, la analiza y almacena la información que contienen, en la base de datos del AGR. Ejecuta el método execute() de la clase Disponibilidad con los atributos JDBCDriver, JDBCURL, URL, Tabla_Servicio, ID_Servicio, Nombre_Servicio y Camas_Disponibles obtenidos de la DTD. Para obtener los servicios que presta ese hospital y la cantidad de camas disponibles para cada uno. Almacena la información obtenida en el contexto y redirige la petición al servlet ActualizaServicios (Ver diagrama de secuencia de la figura 4-13).

El servlet ActualizaServicios obtiene el identificador y el nombre del servicio, las camas disponibles para cada servicio y la dirección del hospital que ofrece ese servicio. Con esta información actualiza la base de datos del AGR (Ver diagrama de secuencia de la figura 4-13).

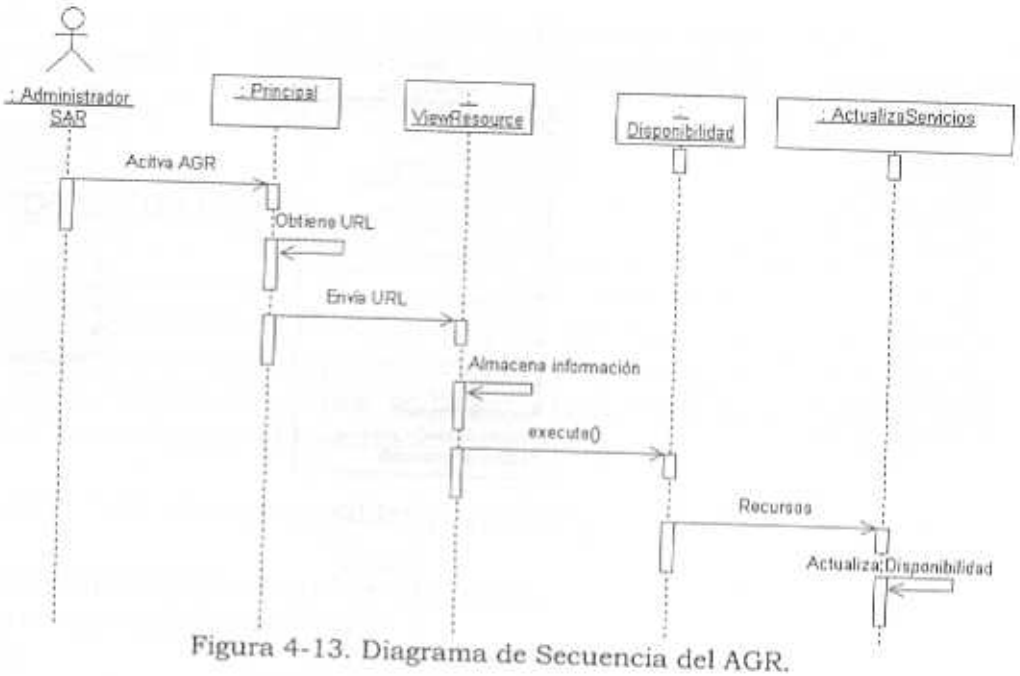

Figura 4-13. Diagrama de Secuencia del AGR.

La figura 4-14, muestra el Diagrama de clases para el subsistema de Asignación de Recursos.

El usuario móvil utilizando el navegador de su teléfono celular, abre la página del Sistema de Asignación de Recursos. Se le presentará una carta wml, en donde se le pide que ingrese su clave de acceso. Una vez que termine de escribir su clave el sistema verifica que el uusario este registrado en el sistema. Si no esta registrado envia un error, de lo contrario le permite ingresar al sistema (Ver diagrama de secuencia de la figura 4-15).

Al ingresar al sistema, el servidor Web carga el servlet ValidaUsuario. Este servlet verifica que el usuario tenga permiso de ingresar al sistema. Si lo tiene, verifica que no tenga una sesión pendiente. Si tiene una sesión pendiente, le envía una advertencia con el estado de su petición y la opción de cancelar o continuar con la misma. Si elige cancelar, o no tenía ninguna sesión pendiente, el sistema le muestra una carta en la que pueda elegir el tipo de hospital que requiere de una lista desplegable. Al seleccionar el tipo deseado, se le presentara otra carta wml, en la que se le pida que elija la especialidad deseada de un menú desplegable. Al seleccionar la especialidad, el servidor Web carga el servlet ProcesaPeticion. Éste recibe del usuario un tipo de hospital y una especialidad. Con esta información busca recursos disponibles en la base de datos del AGR y genera una carta wml. Esta carta contiene el nombre de los hospitales que tienen recursos disponibles con esas caracteristicas

en una lista desplegable para que el usuario elija el que más le convenga (Ver diagrama de secuencia de la figura 4-15).

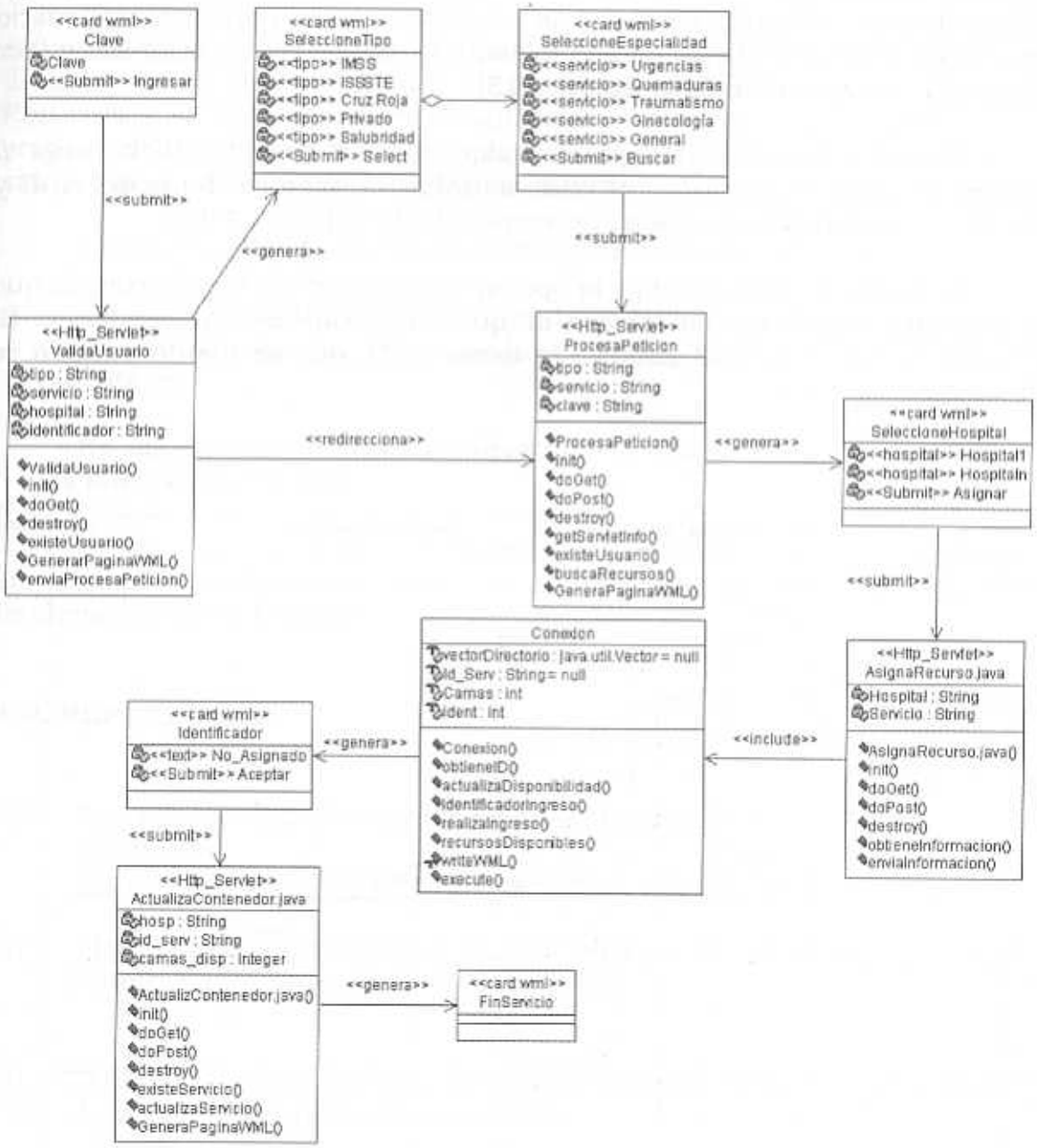

Figura 4-14. Diagrama de clases para el subsistema de Asignación de Recursos.

Cuando el usuario elige un hospital, el servidor Web carga el servlet AsignaRecurso. Éste busca en la base de datos del AGR, la información necesaria para comunicarse con el hospital deseado (JDBCDriver, JDBCURL, Tabla_serv, Id_serv, Nombre_serv, Camas_disp, Tabla_ing,

Id_ing). Entonces, ejecuta el método execute de la clase Conexion, con los atributos mencionados. Este método lo que hace es conectarse con la base de datos del Hospital elegido para asignar el recurso de la especialidad elegida, registrarlo en su base de datos y actualizar su tabla de disponibilidad. Además genera una carta wml para indicarle al usuario móvil que se le asigno ese recurso y le envía un número de asignación (Ver diagrama de secuencia de la figura 4-15).

Cuando el usuario acepta la asignación, el Servidor Web carga el servlet ActualizaContenedor, el cual actualiza la base de datos del AGR y finaliza la sesión (Ver diagrama de secuencia de la figura 4-15).

Cuando un usuario elige la opción "Continuar" de la advertencia que le presenta el sistema en el caso de que tenga una sesión pendiente. El sistema le presentará al usuario la carta en la que se quedo cuando se desconecto de éste.

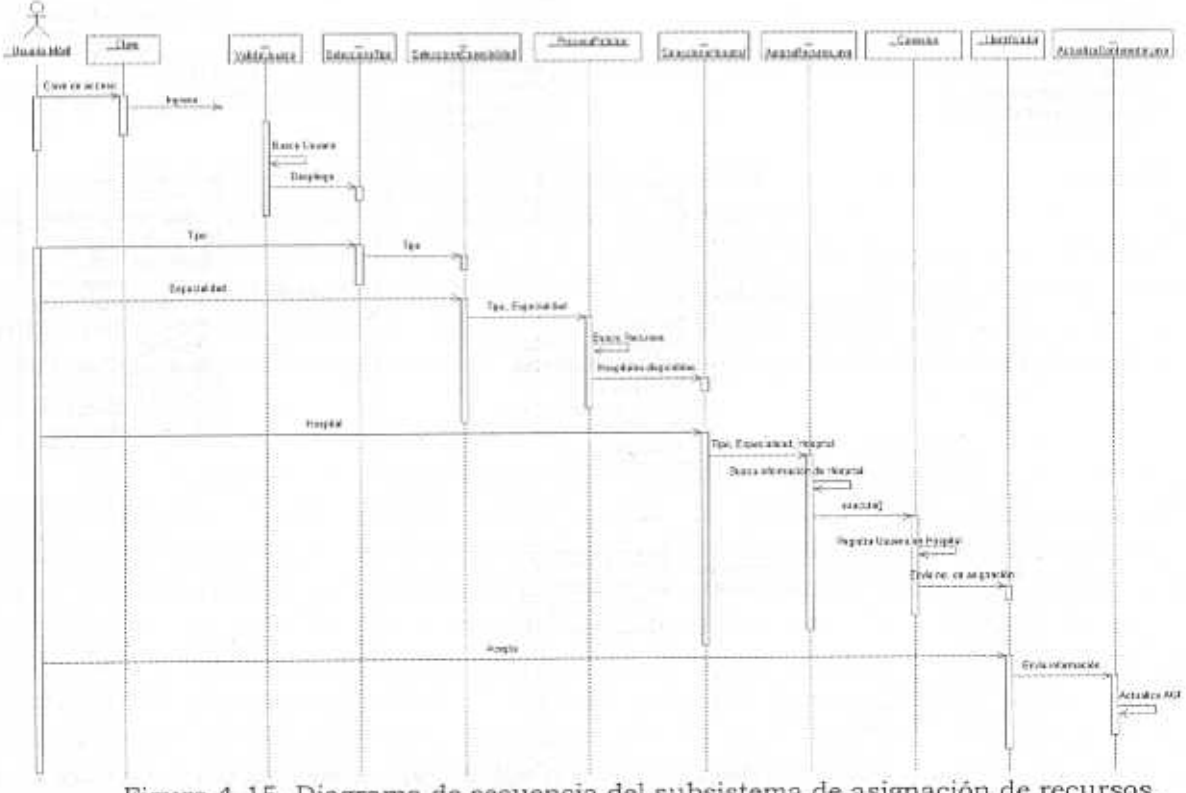

Figura 4-15. Diagrama de secuencia del subsistema de asignación de recursos.

4.8. CONCLUSIONES

Partiendo de los capítulos 2 y 3, Arquitectura y Análisis, respectivamente, se realizó el diseño de la Interfaz de usuario, y se descompuso el sistema en dos subsistemas, el Administrador Global de Recursos y el Subsistema de Asignación de Recursos que comprende el procesamiento de las peticiones realizadas por el usuario móvil. Estos subsistemas, se diseñaron utilizando Diagramas de clase UML.

Se diseñó el AGR, que permite integrar bases de datos heterogéneas.

Se diseñó una DTD en XML que describe la estructura de la base de datos e información relacionada con las bases de datos de los hospitales.

Se diseñó una interfaz intuitiva que se apega a las restricciones de los teléfonos celulares.

En general, se obtuvo un modelo de diseño, el cual puede traducirse en código directamente, esto se realiza en el siguiente capítulo. Implementación y Pruebas.

4.9. REFERENCIAS

[1] Top 10 Usability Guidelines for WAP Applications
Openwave
http://developer.phone.com/resources/uiguide.html

[2] Ingeniería de Software Orientado a Objetos. Bernd Bruegge, Allen H. Dutoit. Ed. Prentice Hall. México, 2002.

[3] Database Design for Smarties Using UML for Data Modeling. Robert J. Muller. Morgan Kaufmann 1999.

[4] Building Web Applications with UML. Jim Conallen. Addison Wesley. 1999.

Capítulo

5

Implementación y Pruebas

Resumen

En los capítulos anteriores se realizaron los casos de uso, se analizó el problema y se definió una arquitectura. En este capitulo, se realizará la implementación del modelo de análisis obtenido en el capitulo anterior.

5.1. INTRODUCCIÓN

En los capítulos anteriores se realizaron los casos de uso, se analizó el problema y se definió una arquitectura. En este capitulo, se realizará la implementación del modelo de análisis obtenido en el capitulo anterior.

En la sección 5.2, se mencionan las herramientas utilizadas y la configuración del entorno de trabajo para el desarrollo del sistema.

En las secciones 5.3, 5.4 y 5.5 se realiza la Implementación del sistema, el cual se divide en 2 módulos principales. El Administrador Global de Recursos (AGR) y el Subsistema de Asignación de Recursos (secciones 5.3 y 5.4 respectivamente). Además de un módulo de administración de usuarios y hospitales que ayudará al Administrador del SAR a agregar y eliminar usuarios y hospitales al sistema (sección 5.5).

En la sección 5.6 Se realizan las pruebas del sistema para verificar que funciona correctamente y que además cumple con los objetivos planteados.

En la sección 5.7 se realizan las conclusiones del capítulo. Por último, en la sección 5.8, se mencionan las referencias del material utilizado para la elaboración de este capítulo.

5.2. ENTORNO DE TRABAJO

La arquitectura del sistema, se implementa con clases Java, en su mayoría Servlets, por lo que es necesario instalar J2SE1.4[1]. Además se utilizarán algunas de las API's que contiene como: JDBC 3.0[2], JSSE[3].

Como se van a utilizar Servlets, es necesario tener un contenedor de Servlets para que se puedan ejecutar, para esto se utiliza el tomcat 4.1[4] que implementa la especificación 2.3 de Java Servlet e incluye varias características para desarrollar aplicaciones y servicios Web.

Además se va a trabajar con documentos XML por lo que es necesario un analizador XML. Para esto se utilizará *XML Parser for Java*[5] de IBM que es una librería para analizar y generar documentos XML.

[1] http://java.sun.com/j2se/1.4.1/download.html
[2] http://java.sun.com/products/jdbc/download.html#corespec30
[3] http://java.sun.com/products/jsse/index.html
[4] http://jakarta.apache.org/tomcat/
[5] http://www.alphaworks.ibm.com/tech/xml4j

Como se mencionó en la justificación de la tesis, además de probar la aplicación con teléfonos celulares se utilizará un simulador para que ésta sea probada en varios equipos. Se utiliza OpenwaveSDK 5.1[6], el cual permite crear aplicaciones WAP, además de simular su funcionamiento.

Se utilizarán dos manejadores de bases de datos, la base de datos del AGR se implementará en Microsoft Access 2000. Para las pruebas del sistema, se utilizarán dos bases de datos, una se implementará en Microsoft SQL Server 2000 y la otra en Microsoft Access 2000

5.3. ADMINISTRADOR GLOBAL DE RECURSOS

Este módulo se encarga de administrar los recursos de todos los hospitales. Se conecta a cada hospital registrado en su base de datos y busca qué servicios ofrece y cuántas camas tiene disponibles para cada uno. Para lograrlo, necesita información sobre la base de datos de cada hospital. En cada hospital se tiene una DTD con la información necesaria para que el AGR pueda realizar consultas sobre los servicios con que cuenta el hospital. El diseño de esta DTD se detalla en la sección 4.3.2 del capítulo 4.

Para que el AGR, y sólo él tenga acceso a las DTD's de los hospitales, se implementa una comunicación segura entre un Cliente y un Servidor para transferir esta DTD por Internet. Es decir, se implementan, un servidor, en el cual se encuentra la DTD con la información del hospital, y un cliente que convive con el AGR y es el que se encarga de pedirle la DTD al Servidor.

La implementación se hace utilizando JSSE [1] que proporciona un marco de trabajo, así como una implementación en Java de los protocolos de Capa de Socket Seguro (SSL) y de Seguridad en la Capa de Transporte (TLS). Además incluye funcionalidad para la encriptación de datos, autentificación del servidor, integridad de los mensajes y autentificación opcional del cliente.

En la figura 5-1 y 5-2 se muestra el código resumido del Servidor y el Cliente respectivamente. En la flecha 1 de la figura 5-1 se indica que se va a usar el formato X509 para certificados digitales. En la instrucción de la flecha 2 se obtiene el nombre de la DTD que se va a transmitir, esto se hace en la instrucción de la flecha 3. En la línea de la flecha 4 se indica que es necesario que se autentifique el cliente. En la sección del *try* que

[6] http://developer.openwave.com/omdt/download_toolkit.html

muestra la flecha 5 se establece el administrador de llaves para la autentificación del servidor.

En la figura 5-2, se muestra el código del cliente. En la sección del *try* que indica la flecha 1 se establece el administrador de llaves para realizar la autentificación del cliente. Utiliza la implementación por defecto de *TrustStore* (Administrador de Confianza) y de las rutinas *SecureRandom*. En la instrucción que índica la flecha 2, se inicia el protocolo de transferencia de archivos.

Para que esto funcione, se necesita crear un *keystore* (Administradores de llaves) y un certificado autofirmado. En este caso, con sus llaves pública y privada. Este procedimiento se explica en el apéndice 1 de esta tesis.

Una vez que se recibió la DTD, el AGR la analiza (a través de la clase ViewResource) para almacenar esta información en su base de datos. El método que analiza la DTD se muestra en la figura 5-3. Dentro del try que índica la flecha 1, se carga la DTD que va a ser analizada. Para cargar una DTD debe crearse primero un objeto Parser y después invocar el método readDTDStream. El valor de retorno de este método es un objeto DTD.

Un objeto DTD contiene información acerca de una DTD, como el modelo de contenido de cada elemento, las declaraciones de atributos y las declaraciones de entidades. A partir de la línea que se índica con la flecha 2; se muestra la forma en la que se recupera la información contenida en el elemento Hospital. En este caso, el método getATTributeDeclarations() toma como argumento el nombre del elemento Hospital y devuelve la relación de todas las definiciones de atributo en forma de una clase especial AttDef. Un objeto AttDef representa una definición de atributo cuyo método getName() devuelve el nombre de dicho atributo, y el método getDeafultStringValue() devuelve el valor del mismo atributo.

Ya que se guardó la información contenida en la DTD, es necesario consultar la Base de Datos de cada Hospital para conocer que servicios presta y cuanta disponibilidad tiene para cada uno. Dentro de la clase ViewResource se encuentra el método enviaParametros(). Este método manda a la clase Disponibilidad la información necesaria para crear la conexión a la Base de Datos de determinado hospital, así como la información del manejador de base de datos que usa y la estructura de su base de datos, para que esta clase construya la sentencia SQL necesaria para conocer la disponibilidad y los servicios de un hospital en particular. En la figura 5-4, se muestra el fragmento de código del método enviaParametros().

```
. . .
import javax.security.cert.X509Certificate;      ----------------------------->(1)
class threadServer extends Thread {
. . .
   public threadServer(Socket s) throws IOException {
      . . .
   }public void run() {
      try {
         String s;
         String filename = in.readLine();      ----------------------------> (2)
         infile = new BufferedReader(new FileReader(filename));
         while((s = infile.readLine())!= null)      ------------------------> (3)
            out.println(s);
         System.out.println("Termina hilo de servicio...");
      } catch (IOException e) {
         System.out.println("Error en el hilo de servicio: " + e);
      }finally {
       . . .
      }
      public class SecureFileServer {
         . . .
        try {
            ServerSocketFactory ssf = getSecureServerSocketFactory();
            ss = ssf.createServerSocket(PORT);
            ((SSLServerSocket)ss).setNeedClientAuth(true);      ------------->(4)
            System.out.println("Arranca Servidor");
            . . .
      })
      private static ServerSocketFactory getSecureServerSocketFactory() {
         SSLServerSocketFactory ssf = null;
         try {     ------------------------------------------------------------> (5)
            SSLContext ctx;
            KeyManagerFactory kmf;
            KeyStore ks;
            char[] passphrase = "skeystore".toCharArray();
            ctx = SSLContext.getInstance("TLS");
            kmf = KeyManagerFactory.getInstance("SunX509");
            ks = KeyStore.getInstance("JKS");
            ks.load(new FileInputStream("skeystore"), passphrase);
            kmf.init(ks, passphrase);
            ctx.init(kmf.getKeyManagers(), null, null);
            ssf = ctx.getServerSocketFactory();
            return ssf;
         }
         . . .
})
```

Figura 5-1. Código resumido del Servidor SSL.

El método buscaServicios() de la clase Disponibilidad, construye una consulta SQL para obtener la información de los servicios que ofrece el hospital. En el fragmento de código de la figura 5-5, la línea que indica la flecha 1, ejecuta la sentencia SQL construida en la línea superior. En la línea que indica la flecha 2, se recuperan los valores resultantes de la consulta.

```
. . .
public class SecureFileClient {
   public static void main(String[] args) throws Exception {
   . . .
   try {     ------------------------------------------------------------------> (1)
      SSLSocketFactory factory = null;
      try {
         SSLContext ctx;
         KeyManagerFactory kmf;
         KeyStore ks;
         char[] passphrase = "ckeystore".toCharArray();
         ctx = SSLContext.getInstance("TLS");
         kmf = KeyManagerFactory.getInstance("SunX509");
         ks = KeyStore.getInstance("JKS");
         ks.load(new FileInputStream("ckeystore"), passphrase);
         kmf.init(ks, passphrase);
         ctx.init(kmf.getKeyManagers(), null, null);
         factory = ctx.getSocketFactory();
      }
      . . .
      SSLSocket socket = (SSLSocket)factory.createSocket(host, port);
      socket.startHandshake();     ----------------------------------------> (2)
      . . .
   }
   . . .
}}}
```

Figura 5-2. Código resumido del Cliente SSL.

```
import com.ibm.xml.parser.*;
. . .
public int leeDTD(String dir) {
   try {     ------------------------------------------------------------------> (1)
      String dtdfile = dir.concat("\\Hosp.dtd");
      FileInputStream is = new FileInputStream(dtdfile);
      Parser parser = new Parser(dtdfile);
      DTD dtd = parser.readDTDStream(is);
      . . .
      )
      Enumeration enum;     ----------------------------------------------------> (2)
      enum = dtd.getAttributeDeclarations("Hospital");
      while(enum.hasMoreElements()) {
         AttDef attrs = (AttDef)enum.nextElement();
         String attName = attrs.getName();
         String attValue = attrs.getDefaultStringValue();
         if (attName.compareTo("nombre") == 0)
            Nombre = attValue;
         . . .
      }
      . . .
   }
   . . .
}
```

Figura 5-3. Código resumido del método leeDTD().

```
public void enviaParametros(HttpServletRequest req, HttpServletResponse resp) {
   Disponibilidad disp = new Disponibilidad();
   Hashtable serv = new Hashtable();
   serv = disp.execute(JDBCDriver, JDBCURL, url, Tabla_Servicio, ID_Servicio,
          Nombre_Servicio, Camas_Disponibles);
   . . .
}
```

Figura 5-4. Fragmento de código de la clase enviaPatametros.

```
public int buscaServicios(String id_serv, String nom_serv, String camas, String tabla) {
    . . .
    try {
       stmt = conn.createStatement();
       String atributos ="SELECT " +id_serv+ ", "  +nom_serv+ ", " +camas+ " FROM " +tabla;
       rs=stmt.executeQuery(atributos);      ----------------------------------------> (1)
       vectorDirectorio=new Vector();
       while (rs.next()) {      ------------------------------------------------------> (2)
          Directorio temp = new Directorio();
          temp.setID_Serv(rs.getString(id_serv));
          temp.setNom_Serv(rs.getString(nom_serv));
          temp.setCamas(rs.getInt(camas));
          vectorDirectorio.addElement(temp);
       }
       if(vectorDirectorio.size()==0)
          return -3;
       return 0;
    } . . .
}
```

Figura 5-5. Fragmento de código de la clase buscaServicios.

5.4. SUBSISTEMA DE ASIGNACIÓN DE RECURSOS

Para que los usuarios móviles puedan ingresar al sistema desde un teléfono celular, es necesario configurar el servidor, para que soporte el MIME type "text/vnd.wap.wml". Êste soporta código wml que es el lenguaje que se utiliza para realizar páginas para cualquier elemento que utilice la tecnologia WAP, como los teléfonos celulares.

Cuando el usuario ingresa la dirección del servidor (en la que reside el SAR) desde el navegador de su teléfono celular. Se carga en éste la página index.wml. Esta página se compone de una carta. En la figura 5-6, se muestra el código de la página index.wml. La línea que índica la flecha 1 y la línea que le sigue, definen la versión del lenguaje wml que se va a usar. Entre el tag <wml> (flecha2) y el tag </wml> se encuentra todo el contenido de la página wml. Al cargar la página se muestra la carta (flecha 3). Êsta le pide al usuario que ingrese su clave de acceso y se le da la opción de ingresar al sistema. En este momento se manda llamar al servlet ValidaUsuario.

```
<?xml version="1.0"?>      ------------------------------------------------------------> (1)
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.3//EN" "http://www.phone.com/dtd/wml13.dtd">
<wml>      ------------------------------------------------------------------------------> (2)
   <card id="Clave" title="Clave">      ---------------------------------------------> (3)
      <p>
         Introduzca clave de acceso: <input type="text" name="Clave"/>
      </p>

      <do type="accept" label="Ingresar">
         <go
            href="http://148.204.211.194:8080/wap/servlet/ValidaUsuario?Clave=$(Clave)"
         />
      </do>
   </card>
</wml>
```

Figura 5-6. Código de la página index.wml

El servlet ValidaUsuario (figura 5-7) recibe la clave que ingresó el usuario (flecha 1) y verifica que el usuario esté registrado en su base de datos. Esto lo hace por medio del método existeUsuario() (flecha 2). Si el usuario tiene permiso de ingresar al sistema, entonces verifica que no tenga una sesión pendiente (flecha 3). Si el usuario dejó una petición sin concluir (flecha 5), el sistema le presentará una advertencia en la que le da la información del estado de su petición y le pregunta si desea cancelar o continuar con la misma. Si desea continuar, el sistema le mostrará la carta en la que quedo pendiente su sesión. Si cancela la sesión, o si no tenía ninguna sesión pendiente (flecha 4), le envía la carta de inicio del sistema (método GenerarPaginaWML() mostrado en la figura 5-8).

```
. . .
public class ValidaUsuario extends HttpServlet {
   . . .
   public void doGet (HttpServletRequest req, HttpServletResponse resp) throws
      ServletException, IOException {

      session = req.getSession();
      clave = req.getParameter("Clave");      ----------------------------------------> (1)
      . . .
      if ( existeUsuario(clave) == 0) {       ----------------------------------------> (2)

         if (!session.isNew()) {      ---------------------------------------------------> (3)
            java.util.Date horaAntes = new java.util.Date(System.currentTimeMillis() -
                                                60*60*1000);
            java.util.Date cincomAntes = new java.util.Date(System.currentTimeMillis() -
                                                5*60*1000);
            java.util.Date creada = new java.util.Date(session.getCreationTime());
            java.util.Date ingreso = new java.util.Date(session.getLastAccessedTime());

            if (creada.before(horaAntes) || ingreso.before(cincomAntes) ||
                (clave.compareTo(session.getAttribute("clave")) != 0)) {
               session.invalidate();
               session = req.getSession();
            }
         }
         . . .
         if (tipo == null && servicio == null) {      ---------------------------------> (4)

            session.setAttribute("clave", clave);
            GenerarPaginaWML(resp);
         }
         else if ( hospital == null)
            enviaProcesaPeticion(resp);      ----------------------------------------> (5)

      }
      . . .
} // fin del método doGet()
```

Figura 5-7. Fragmento del servlet ValidaUsuario.

Este método crea la carta "index" (flecha 1), la cual le pide al usuario que elija un tipo de hospital. También crea la carta "serv" (flecha 2), la cual se muestra al usuario después de elegir un tipo de hospital. Se le pide al usuario que elija una especialidad. Después de que el usuario hizo su elección, se le presenta la tercera carta "env" (flecha 3), esta carta funciona como confirmación, se le muestra al usuario el tipo de hospital y

especialidad que eligió y se le da la oportunidad de regresar a los menús anteriores en caso de que su elección no haya sido la adecuada. En caso contrario, al oprimir la tecla *acept* se manda llamar al servlet ProcesaPeticion.

El servlet ProcesaPetición recibe el tipo de hospital y la especialidad que Eligió el usuario. Entonces busca recursos disponibles (método buscaRecursos()) en su base de datos y le envía una carta con el nombre de los hospitales que pueden atender al paciente para que el usuario elija uno.

```
public void GenerarPaginaWML (HttpServletResponse resp) {
   resp.setContentType("text/vnd.wap.wml");
   . . .
   // Se genera el contenido de la página WML
   out.println("<?xml version=\"1.0\"?>");
   out.println("<!DOCTYPE wml PUBLIC \"-//OPENWAVE.COM//DTD WML 1.3//EN\"
                \"http://www.openwave.com/dtd/wml13.dtd\"> ");
   out.println("<wml> <template>");
   out.println("<do type=\"options\" label=\"Back\">");
   out.println("<prev /> </do> </template>");
   out.println("<card id=\"index\" title=\"Seleccione tipo\">");      ----------------> (1)
   out.println("<onevent type=\"onenterforward\">");
   out.println("<refresh>");
   out.println("<setvar name=\"Tipo\" value=\"\" />");
   out.println("</refresh> </onevent>");
   out.println("<do type=\"options\">");
   out.println("<noop /> </do> <p>");
   out.println("<select name=\"tipo\">");
   out.println("<option onpick=\"#serv\" value=\"IMSS\">IMSS</option>");
   out.println("<option onpick=\"#serv\" value=\"ISSTE\">ISSTE</option>");
   out.println("<option onpick=\"#serv\" value=\"Cruz Roja\">Cruz Roja</option>");
   out.println("<option onpick=\"#serv\" value=\"Privado\">Privado</option>");
   out.println("<option onpick=\"#serv\" value=\"Salubridad\">Salubridad</option>");
   out.println("</select> </p> </card>");
   out.println("<card id=\"serv\" title=\"Seleccione especialidad\">");      ----------> (2)
   out.println("<onevent type=\"onenterforward\">");
   out.println("<refresh>");
   out.println("<setvar name=\"servicio\" value=\"\" />");
   out.println("</refresh> </onevent> <p>");
   out.println("<select name=\"servicio\">");
   out.println("<option onpick=\"#env\" value=\"Urgencias\">Urgencias</option>");
   out.println("<option onpick=\"#env\" value=\"Quemaduras\">Quemaduras</option>");
   out.println("<option onpick=\"#env\" value=\"Traumatismo\">Traumatismo</option>");
   out.println("<option onpick=\"#env\" value=\"Ginecologia\">Ginecologia</option>");
   out.println("<option onpick=\"#env\" value=\"General\">General</option>");
   out.println("</select> </p> </card>");
   out.println("<card id=\"env\" title=\"Buscar Disponibilidad\">");      ------------> (3)
   out.println("<p>");
   out.println("Tipo de Hospital: $(tipo)");
   out.println("<br />");
   out.println("Especialidad: $(servicio)");
   out.println("<br /> </p>");
   out.println("<do type=\"accept\">");
   out.println("<go");
   out.println("href=\"http://148.204.211.194:8080/wap/servlet/ProcesaPeticion?Tipo=$(tipo)
               &amp;Servicio=$(servicio)\"");
   out.println("/> </do> </card> </wml>");
   out.flush();
   out.close();
} // fin de GenerarPaginaWML ()
```

Figura 5-8. Código del método GeneraPaginaWML() del servlet ValidaUsuario.

Cuando el usuario elige un hospital de la lista mostrada, se manda llamar al servlet AsignaRecurso. Este servlet obtiene de su base de datos la información necesaria para comunicarse con el hospital elegido y realizar la asignación. Además, este servlet implementa el método enviaInformacion() que es muy parecido al método enviaParametros() de la clase ViewResource. La clase conexion se encarga de comunicarse con el hospital y realizar la asignación, es similar a la clase Disponibilidad. En el apéndice B encontrará todos los códigos del sistema.

5.5. ADMINISTRACIÓN DE USUARIOS Y HOSPITALES

El sistema cuenta con una interfaz que puede utilizar el administrador del SAR para administrar a los Usuarios y a los Hospitales que van a formar parte del sistema.

En la figura 5-9 Se presenta la pantalla de administración de usuarios. Del lado izquierdo se encuentra el menú de operaciones que se pueden hacer sobre un usuario (Agregar, Eliminar y Buscar). Sólo en la opción de Agregar, es requerido tanto la clave de usuario como su nombre. En las opciones Eliminar y Buscar, sólo se pide la clave.

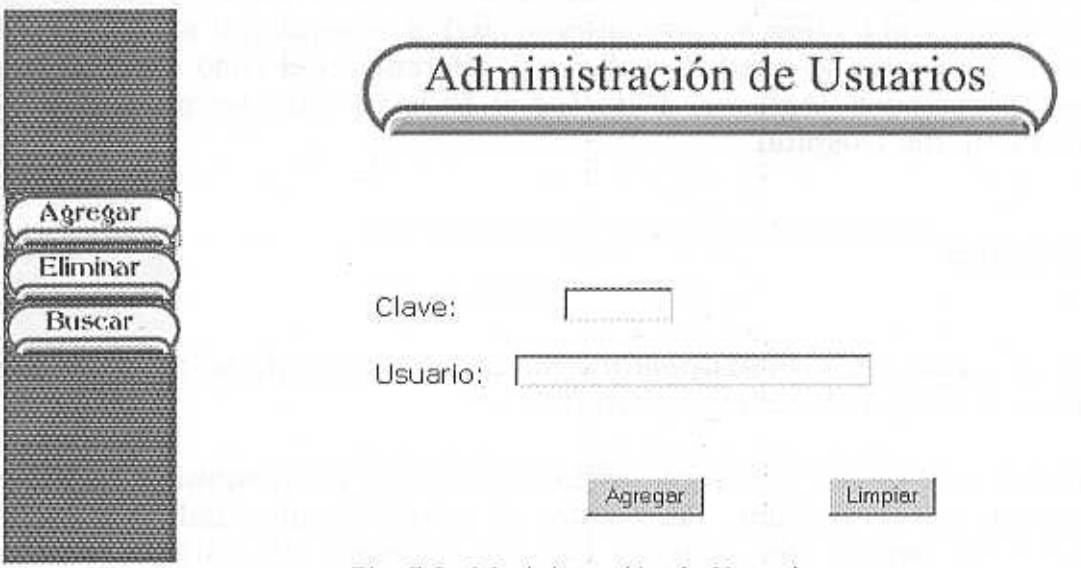

Fig. 5.9. Administración de Usuarios.

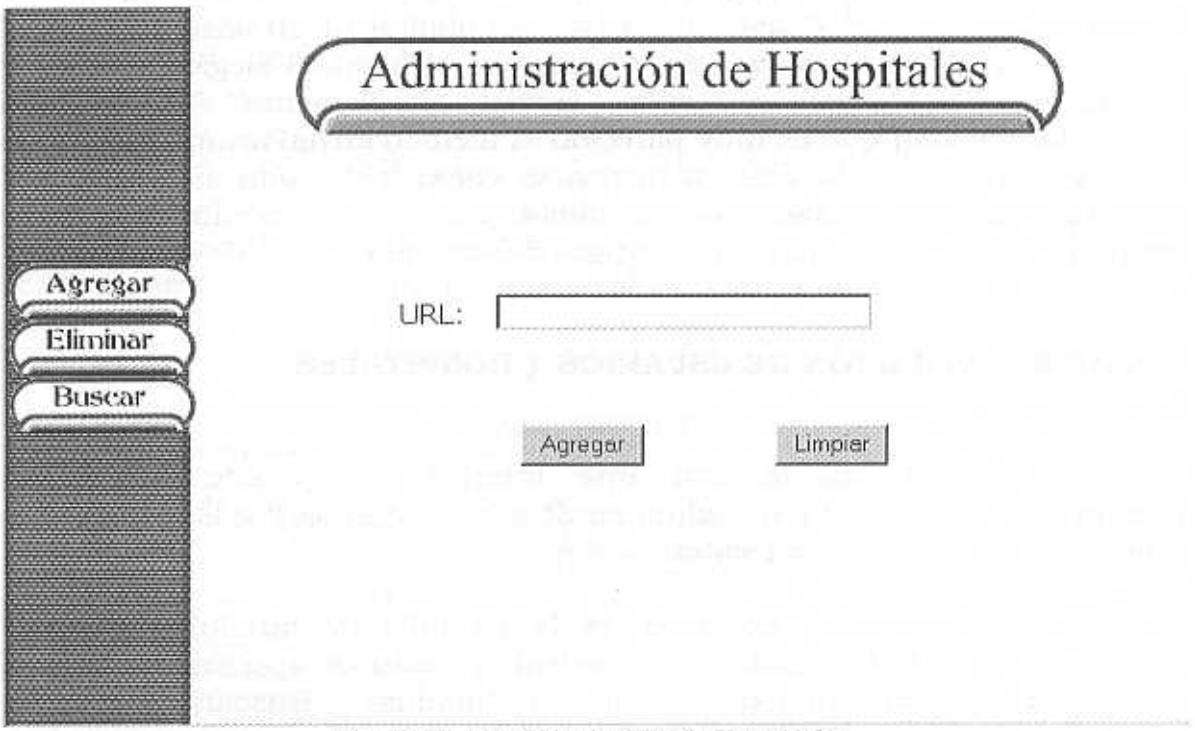

Fig. 5.10. Administración de Hospitales.

En la figura 5-10. Se presenta la pantalla de administración de hospitales. Al igual que en el caso anterior, del lado izquierdo se presentan las operaciones que se pueden realizar. A diferencia del caso anterior, en las tres operaciones (Agregar, Eliminar y Buscar) sólo se pide que se ingrese la URL del Hospital.

5.6. PRUEBAS

Para verificar que el sistema funcione correctamente se seguirán los diagramas de secuencia de la sección 3.5.

Primero se tienen que registrar los usuarios y los hospitales que van a participar en el sistema. Para esto, el administrador del SAR debe ingresar a la página del sistema con un nombre de usuario y una contraseña validos (figura 5-11).

Cuando el sistema verifica que la contraseña es correcta, le permite al usuario elegir entre ir al administrador de usuarios o de hospitales. Si se elige Usuarios y Agregar que aparece en el menú del lado izquierdo, se presenta la pantalla mostrada en la figura 5-9. Al ingresar la clave y el

usuario y oprimir el botón Agregar, aparecerá una pantalla de confirmación (figura 5-12).

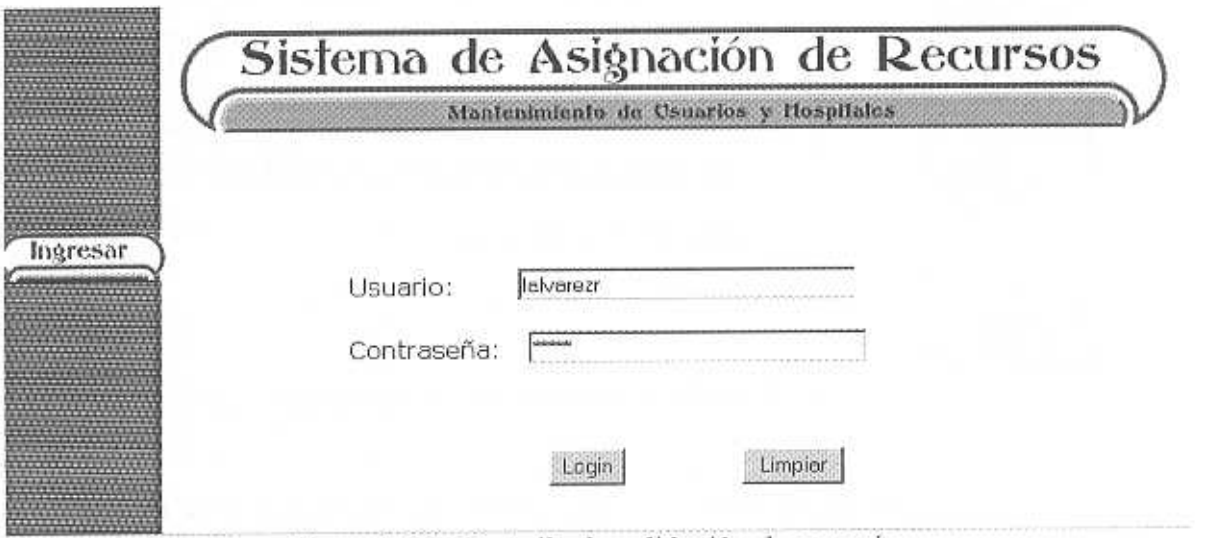

Figura 5-11. Pantalla de validación de usuario.

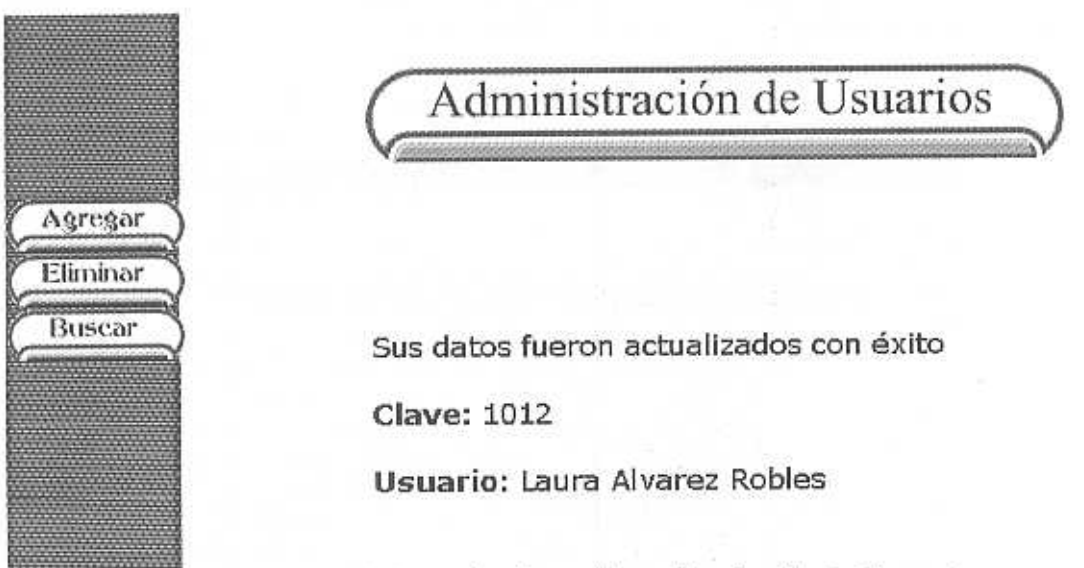

Figura 5-12. Pantalla de confirmación de alta de Usuario.

Si el administrador del SAR elige Hospitales y Agregar que aparece en el menú del lado izquierdo, se presenta la pantalla mostrada en la figura 5-10. Al ingresar la URL del hospital y oprimir el botón Agregar, aparecerá una pantalla de confirmación (figura 5-13).

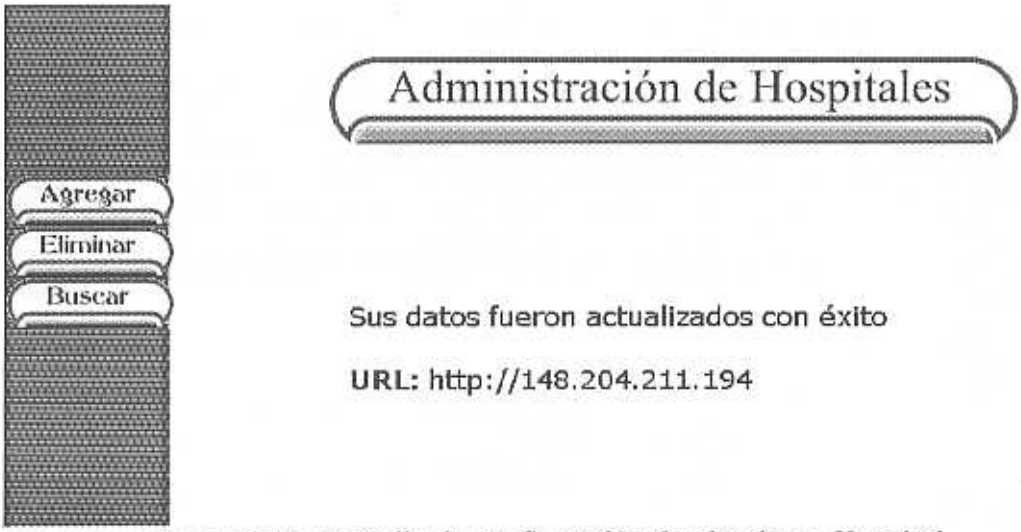

Figura 5-13. Pantalla de confirmación de alta de un Hospital.

Ahora, es necesario que cada hospital que se agregue a la base de datos tenga una DTD con la información sobre dicho hospital y su base de datos. El administrador del hospital ingresa al editor DTD (figura 5-14) donde se le muestra un formulario. Al ingresar la información solicitada y presionar el botón OK, se genera la DTD para ese hospital. La DTD generada se muestra en la figura 5-15.

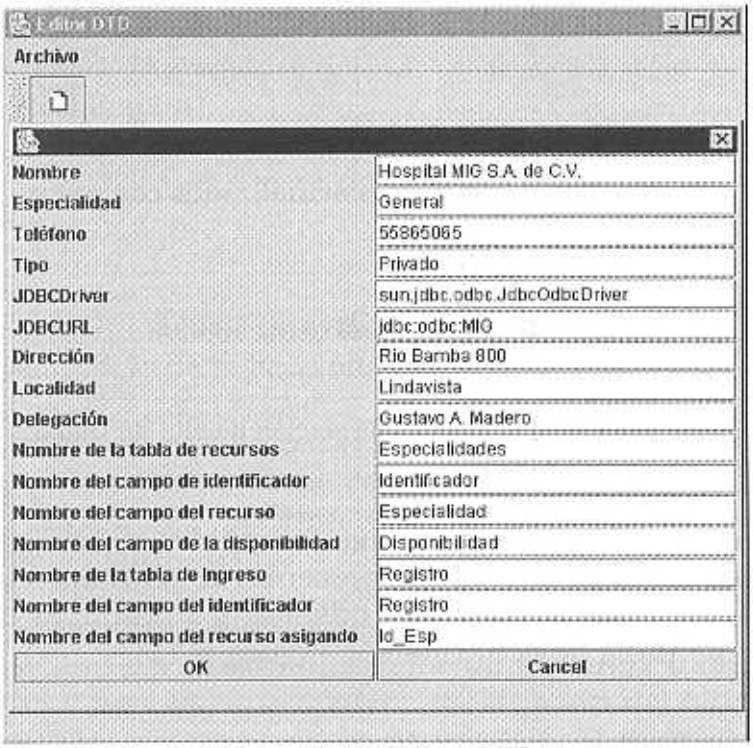

Figura 5-14. Editor DTD

```
<!ELEMENT Disponibilidad (Hospital, Servicio, Ingreso)>
<!ELEMENT Hospital (Domicilio)>
<!ATTLIST Hospital
nombre CDATA #FIXED "Hospital MIG S.A. de C.V."
especialidad CDATA #FIXED "General"
telefono CDATA #FIXED "55865065"
tipo CDATA #FIXED "Privado"
JDBCDriver CDATA #FIXED "sun.jdbc.odbc.JdbcOdbcDriver"
JDBCURL CDATA #FIXED "jdbc:odbc:MIG"
<!ELEMENT Domicilio EMPTY>
<!ATTLIST Domicilio
direccion CDATA #FIXED "Rio Bamba 800"
localidad CDATA #FIXED "Lindavista"
delegacion CDATA #FIXED "Gustavo A. Madero"
<!ELEMENT Servicio EMPTY>
<!ATTLIST Servicio
nombre_tabla CDATA #FIXED "Especialidades"
nombre_id CDATA #FIXED "Identificador"
nombre_serv CDATA #FIXED "Especialidad"
camas_disp CDATA #FIXED "Disponibilidad"
<!ELEMENT Ingreso EMPTY>
<!ATTLIST Ingreso
nombre_tabla CDATA #FIXED "Registro"
nombre_id CDATA #FIXED "Registro"
nombre_serv CDATA #FIXED "Identificador"
```

Figura 5-15. DTD del hospital MIG

Después se inicializa el AGR. El administrador del SAR se conecta mediante un cliente SSL al servidor SSL de un hospital para descargar su DTD, ejecutando el siguiente comando en la interfaz de comandos.

```
C:\java -Djavax.net.ssl.trustStore=ckeystore –
    Djavax.net.ssl.trustStorePassword=ckeystore
    SecureFileClient 148.204.211.230 Hosp.dtd
```

Después activa al AGR introduciendo la siguiente dirección en el navegador web.

http://148.204.211.194:8080/contenedor/servlet/Principal

Ahora se puede probar el Subsistema de asignación de Recursos. Cuando el usuario ingrese al sistema, éste le presentará una carta (figura 5-16 (1)) en la que se le pide ingresar su clave de acceso. Al terminar, se elige la opción "Ingresar".

Si el usuario se encuentra registrado en el sistema. Se le presenta un menú (figura 5-16(2)) con los tipos de Hospitales con que cuenta el sistema. Éstos pueden pertenecer al IMSS, al ISSSTE, a la Cruz Roja, a Salubridad, o ser Privados. Se dan estas opciones, por si el paciente cuenta con algún tipo de Seguro Médico, pueda ser canalizado al hospital adecuado. En caso de no conocer nada sobre el paciente, puede ser enviado a algún hospital de la Cruz Roja o de Salubridad.

En este caso se eligió el tipo de hospital "Privado". El sistema presenta otro menú (figura 5-16(3)) para que el usuario indique la especialidad requerida para atender al paciente y así asegurar que el hospital que se le asigne, realmente pueda atenderlo. De las especialidades que se puede elegir son las siguientes, Urgencias, Quemaduras, Traumatismo, Ginecología, y General. Al oprimir el botón debajo de la palabra "Opción", el sistema realiza una búsqueda con los parámetros enviados. En caso de que no encuentre disponibilidad con esos parámetros, le envía al usuario la advertencia de "Recursos no disponibles". En ese caso el usuario puede regresar a los menús anteriores para probar con otros parámetros. Si hay disponibilidad, envía él o los nombres de los hospitales en los que se puede atender al paciente de acuerdo con los parámetros de búsqueda. En el caso del ejemplo se encontraron dos hospitales (Figura 5-16(4)). El usuario selecciona uno de los hospitales y oprime el botón debajo de la palabra "Elegir".

El sistema se comunica con ese hospital y le envía al usuario un número de asignación con el cual se puede registrar al paciente en el hospital cuando llegue (figura 5-16(5)). Una vez que el sistema le envía el número de asignación y el usuario lo acepta finaliza aplicación (figura 5-16(6)).

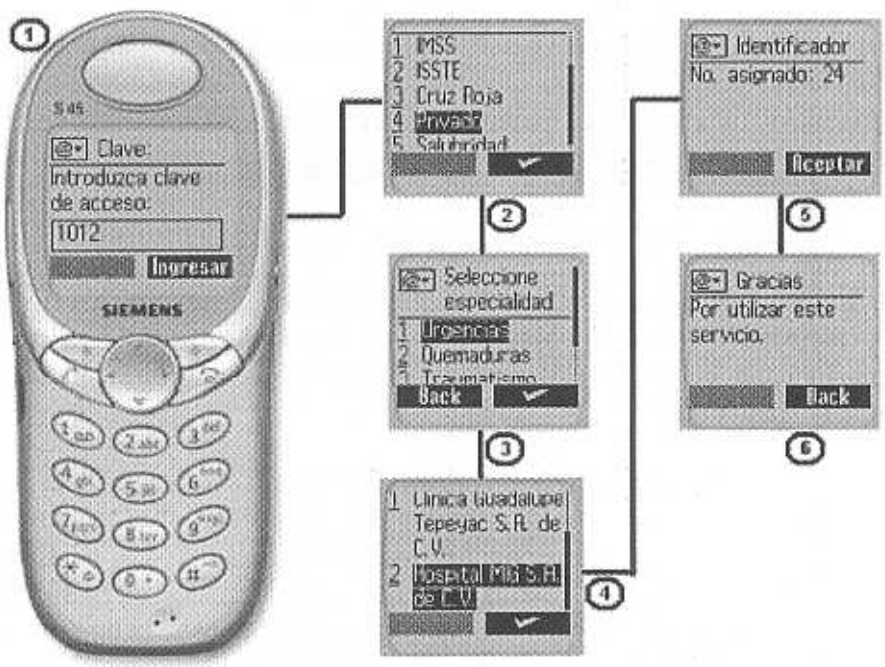

Figura 5-16. Pantallas del Sistema de asignación de Recursos.

5.7. CONCLUISONES

En este capitulo, se lograron integrar las tecnologías de cómputo móvil y las tecnologías para la interoperabilidad de bases de datos. Se implementó un mecanismo que permite un acceso seguro a los sistemas de información de los hospitales.

Se implemento un mecanismo que logra que el sistema sea capaz de recuperarse de posibles desconexiones.

También se detalló cómo se implementan el Subsistema de asignación de Recursos, el Administrador Global de Recursos, la función de mantenimiento a usuarios y hospitales, y la función para crear DTD's y descargarlas de una manera segura. Todos los códigos se encuentran en el Apéndice B.

Además, se han descrito las pruebas que se han realizado a todos los módulos del sistema. De esta manera, se comprueba su correcto funcionamiento con dos hospitales, ya que se han obtenido resultados exitosos.

5.8. REFERENCIAS

[1] JSSE Reference Guide for the J2SDK, v 1.4
http://java.sun.com/j2se/1.4/docs/guide/security/jsse/JSSERefGuide .html

Capítulo

6

Conclusiones y trabajos a futuro

Resumen

En este capítulo se describen los logros alcanzados de acuerdo a los objetivos planteados. También se describen algunos trabajos futuros que pudieran realizarse tomando como base este trabajo.

6.1. INTRODUCCIÓN

En este capítulo se describen los logros alcanzados en base a los objetivos planteados al inicio de la tesis y las aportaciones que no se habían contemplado en los objetivos. También se describen algunos trabajos futuros que pudieran realizarse tomando como base este trabajo. Por último, se mencionan las conclusiones generales relacionadas con el planteamiento del problema y la solución presentada.

6.2. LOGROS ALCANZADOS

Se logro implementar un sistema de asignación de recursos para entidades móviles basado en una infraestructura de telefonía celular. La arquitectura del sistema se basa en el modelo de funcionamiento WAP (capítulo 2), ésta integra tecnologías de telefonía celular como GSM. Logrando así, que ésta sea flexible y se puede trasladar a cualquier área en la que sea necesaria una asignación de recursos en un ambiente móvil. En los capítulos 3, 4 y 5 se presenta el análisis, diseño e Implementación de un caso de prueba (Asignación de recursos médicos en el D. F.) basado en la arquitectura propuesta.

Se realizó una Definición de Tipo de Documento en XML que define la estructura de la base de datos con la información relevante de cada Proveedor de Recursos (sección 4.3.2). De esta forma y con la ayuda de una base de datos, se implementó un mecanismo (AGR) que permite integrar bases de datos heterogéneas. Éste permite proporcionar información oportuna sobre los recursos disponibles siempre y cuando tengan un manejador JDBC de preferencia de tipo 4. Esto es porque al usar un manejador de tipo 4 no es necesario instalar librerías, o algún software intermedio en el AGR para tener acceso a bases de datos con este tipo de manejador. El sistema también funciona con los demás tipos de manejadores, sólo que el funcionamiento no es tan automático, ya que es necesario instalar un software intermedio. En el caso de los manejadores de tipo 1 (ODBC) es necesario instalar una fuente de datos (DSN, *Data Source Name*).

Utilizando J2SSE se implementó un mecanismo Cliente – Servidor seguro para los Sistemas de Información de los proveedores de Recursos.

Se lograron integrar en esta tesis las tecnologías de cómputo móvil, las tecnologías para la interoperabilidad de bases de datos y la tecnología de criptografía de llave pública.

Se implementó también, un mecanismo para el manejo de desconexiones del teléfono celular. Esto es importante debido a que los dispositivos móviles se caracterizan por el hecho de no tener conexiones permanentes, a diferencia de los sistemas de escritorio. Este mecanismo se implementó utilizando sesiones para cada usuario que ingrese al sistema.

En cada uno de los capítulos de esta tesis se dan una serie de recomendaciones relacionadas con la tecnología utilizada y el desarrollo de la aplicación.

6.3. TRABAJOS A FUTURO

6.3.1. Cómputo ubicuo

El cómputo ubicuo[1] es el uso de un nuevo tipo de dispositivos inteligentes y portales para acceder a información crítica que sea necesaria en ese momento y que faciliten la toma de decisión sin importar el lugar o el instante de tiempo en que sea necesario.

Mark Weiser[2] acuñó el termino "cómputo ubicuo" en 1988 para describir el incremento en el uso de sistemas de cómputo a través del ambiente físico, haciéndolos disponibles y a la vez invisibles al usuario. Esta propuesta se ha posicionado como la tercera generación o paradigma en la computación y cuenta ya con múltiples aplicaciones y equipos de investigación que procuran su desarrollo. El concepto de ubicuidad se refiere en general a la presencia de una entidad en todas partes; pero en la computación adquiere la característica de ser, además, invisible. Este paradigma pretende brindar sistemas de cómputo inteligentes que se adapten al usuario, y cuyas interfaces permitan que éste realice un uso intuitivo del sistema. De allí que la meta, de la computación ubicua, de integrar varias computadoras (dispositivos) al entorno físico busca habilitar los beneficios de éstas y de la información digitalizada en todo momento y en todas partes.

Una de las áreas que están en investigación es el área de salud ubicua (Pervasive Healthcare). En la actualidad los médicos y enfermeras están en movimiento constantemente, en la mayoría de los hospitales no hay lugar para poner computadoras. Además, las interrupciones son frecuentes a lo largo del día. Debido a esto, las tareas y actividades en el área de la salud se prestan para una solución con cómputo móvil y ubicuo.

Para lograr la invisibilidad en el cómputo propuesta por Mark Weiser, los desarrolladores de aplicaciones tanto móviles como ubicuas requerirán de software intermedio[3] (middleware) confiable que proporcione un modelo de programación fuerte en el cual basar sus productos.

En el diseño de aplicaciones para dispositivos móviles, serán también importantes otras características técnicas como por ejemplo el ancho de banda de la red y sus posibilidades multimedia, etc. Este conjunto de opciones técnicas que permiten la ubicuidad de un sistema de cómputo y las posibilidades conceptuales de diseñar la información, pueden considerarse diseño ubicuo. El diseño ubicuo brinda opciones para mejorar el desempeño de Internet y la Web como medios de comunicación.

La computación ubicua, esta enfocada a todo lo referente a innovación, diseño, uso y evaluación de las nuevas generaciones de dispositivos portátiles pequeños. Los dispositivos portátiles inalámbricos trabajan en conjunto mediante una red, para proporcionar servicios y utilizar los recursos disponibles en la red. Para esto, es necesario contar con protocolos de comunicación, hardware y software que permitan interactuar con los demás dispositivos disponibles en el ambiente donde nos encontremos.

Es necesario desarrollar aplicaciones que funcionen de manera confiable en situaciones cambiantes. Las funciones que interactúan con los usuarios y la infraestructura deben de mantenerse a niveles tolerables (con el mínimo margen de error), incluso si los dispositivos cambian de ubicación, interactúan con nuevos dispositivos o se encuentran en una red que provee servicios limitados.

6.3.2 Localización

Algo que no se contempló en la realización de esta tesis es la respuesta del sistema en base a la localización del usuario móvil.

La adaptabilidad es uno de los requerimientos fundamentales en el cómputo "nómada" El principio básico de la adaptabilidad es simple. Cuando las circunstancias cambian, el comportamiento de una aplicación cambia de acuerdo a los deseos del usuario (de acuerdo a los principios atribuidos a él).

El monitoreo del contexto, es una de las áreas fundamentales que permiten construir aplicaciones adaptables. Las tres áreas principales son:

descubrimiento (equipo disponible), servicio de localización (servicios disponibles), y capacidades disponibles (poder de cómputo, capacidad de almacenamiento, capacidad de comunicación).

Por ejemplo, en este Centro, se desarrolló una aplicación basada en el monitoreo del contexto. La tesis Servicio de localización sobre una infraestructura de telefonía celular [4], ofrece un servicio que proporciona la ruta de acceso más corta a un punto de interés respondiendo al contexto del usuario. Es decir, la ruta se traza desde el lugar en el que se encuentra el usuario.

6.3.3. Esquemas XML

Como se mencionó en la sección de logros alcanzados, se diseñó una DTD con la estructura de las Bases de Datos de los CR. Cuando se inició este trabajo la especificación ampliamente usada para validar documentos XML era por medio de una DTD. Actualmente el uso de éstas ha quedado en un uso más restringido, y se están empezando a utilizar los esquemas XML como estándar; por ello, se recomienda, como un trabajo a futuro transformar el sistema para que utilice esquemas XML en lugar de DTDs.

Al igual que las DTDs, los esquemas XML describen la estructura de la información. El motivo de la creación de este nuevo estándar es, básicamente, la utilidad. Durante un tiempo, y a falta de otra solución más ajustada, se emplearon los mecanismos que proporcionaba SGML para modelar la información en XML. Pero el descubrimiento de nuevas aplicaciones XML al margen de la estructuración de documentos forzó la creación de otras soluciones que ayudaran a eliminar algunas de las desventajas de las DTDs, una de las cuales es que posee un lenguaje propio de escritura. Por el contrario, los esquemas XML están escritos en éste lenguaje.

6.3.4. Escalabilidad

La escalabilidad de un sistema es la capacidad para responder a cargas de trabajo crecientes.

Un principio básico en el diseño de sistemas escalables es que la carga de trabajo de cualquier componente del sistema debe estar acotada por una constante independiente del tamaño del sistema. Si no es así, entonces el crecimiento del sistema estará limitado.

El AGR del sistema es un mecanismo centralizado, por lo tanto tiene un límite en cuanto a las peticiones que puede atender. Este mecanismo no es escalable, en consecuencia el sistema tampoco lo es.

Queda como un trabajo futuro implementar un mecanismo distribuido que administre los recursos y que además sea tolerante a fallas. La replicación es el método comúnmente usado para lograrlo, pero es necesario adaptarlo para que responda a las características del cómputo móvil.

6.3.5. Interacción Humano - Computadora

La Interacción Humano–Computadora (HCI, *Human-Computer Interaction*) es una disciplina relacionada con el diseño, evaluación e implementación de sistemas de cómputo interactivo para uso humano.

La HCI se relaciona con el desempeño de las tareas realizadas por humanos y máquinas, la estructura de comunicación entre ellos, las habilidades de los humanos para utilizar las máquinas (interfaces intuitivas) y la programación de las interfaces.

Las interfaces de los módulos de administración de usuarios y hospitales del Sistema desarrollado en esta tesis se implementaron en un ambiente Web. Esto es debido a que la mayoría de las personas que utilizan PC están familiarizadas con las páginas HTML y su "navegación". Una interfaz de este tipo no requiere que haya una capacitación en los usuarios obteniendo un mejor desempeño del sistema. Queda para el futuro adaptarla a las nuevas tecnologías para obtener una interfaz más intuitiva.

Por otro lado, los dispositivos móviles actuales como PDAs, teléfonos celulares, etc. Comparten un problema común: los usuarios acceden a través de pequeñas interfaces a servicios y recursos de cómputo poderosos. Esto introduce nuevos retos, como el diseño de interfaces que se adapten a un acceso de red intermitente. Además que sean sensibles al contexto y a la ubicación.

Algunos de estos retos son:

- El ambiente de la aplicación debe adaptarse a la ubicación del usuario.
- Los usuarios normalmente no tienen un entrenamiento formal en este tipo de tecnologías.
- Manejar adecuadamente las limitaciones de entrada y salida de información. Las pantallas están mejorando en términos de resolución, color y píxeles/cm, pero seguirán siendo pequeñas. La calidad en el sonido es muy pobre y tiene muchas restricciones en cuanto a reconocer la voz para ingresar información. Los teclados están limitados en cuanto al tamaño y número de teclas.
- La multitarea y el soporte para interrumpir las tareas es una de las claves de los sistemas de escritorio qué aún no se encuentran en los dispositivos móviles.

En este sentido, para la realización de la interfaz móvil del Sistema de Asignación de Recursos, se trato de que fuera "amigable" con el usuario. La interacción con el usuario se hace por medio de listas desplegables con la información que el sistema requiere del usuario. De esta forma, se reducen los errores y el tiempo en la introducción de información vía teclado. Excepto en el registro del usuario, ya que es necesario que éste teclee su clave de acceso. Queda como trabajo futuro implementar esta parte utilizando reconocimiento de voz.

6.4. COMENTARIOS FINALES

Las ventajas que brinda un sistema de este tipo son infinitas, el acceso a la información es de forma inmediata porque se puede acceder desde cualquier lugar. Debido a que su uso es simple e intuitivo, puede llegar a un amplio espectro de personas, además de que el acceso a aplicaciones de este tipo desde teléfonos celulares es de bajo costo si lo comparamos con una computadora.

La ventaja principal del caso de prueba desarrollado, es que, indirectamente mejora la calidad en la atención a los pacientes. Esto debido a que se reduce el tiempo en el que se canaliza un paciente a un hospital.

6.5 REFERENCIAS

[1] Weiser, M. The Computer for the 21st Century. Scientific American, Vol. 265, No. 3, September 1991.

[2] Davies, N. and Gellersen, H. Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. IEEE Pervasive Computing, Vol. 1, No. 1, January-March 2002.

[3] Kimmo Raatikainen et.al., "Application Requirements for Middleware for Mobile and Pervasive Systems", Mobile Computing and Communications Review, Volume 6, Number 4, pages 16-24.

[4] Paredes A. Daniel (2003), "Servicio de localización sobre una infraestructura de telefonía celular", Tesis de Maestría, Centro de Investigación en Computación del Instituto Politécnico Nacional.

Apéndice

Administrador de llaves

Un administrador de llaves (KeyStore) proporciona llaves públicas a otros cuando estas son necesitadas.

1. Para crear un nuevo almacén de llaves y un certificado autofirmado con sus correspondientes llave pública y privada. Se introduce el siguiente comando.

   keytool –genkey –alias server –keyalg RSA – validity 31 –keystore keystore

   Se inicia el siguiente dialogo.

Enter keystore password: **password**
What is your first and last name?
[Unknown]: **Laura Alvarez**
What is the name of your organizational unit?
[Unknown]: **CIC**
What is the name of your organization?
[Unknown]: **IPN**
What is the name of your City or Locality?
[Unknown]: **DF**
What is the name of your State or Province?
[Unknown]: **Mexico**
What is the two-letter country code for this unit?
[Unknown]: **MX**
Is CN=Laura Alvarez, OU=CIC, O=IPN, L=DF, ST=Mexico, C=MX correct?
[no]: **yes**
Enter key password for <Laura Alvarez>
(RETURN if same as keystore password): **<CR>**

Éste es el almacén de llaves que utilizará el servidor.

2. Examinar el contenido del almacén de llaves

   keytool -list -v -keystore keystore
   Enter keystore password: **password**

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: server
Creation date: 31-dic-2002
Entry type: trustedCertEntry

Owner: CN=Laura Alvarez, OU=CIC, O=IPN, L=DF, ST=Mexico, C=MX
Issuer: CN=Laura Alvarez, OU=CIC, O=IPN, L=DF, ST=Mexico, C=MX
Serial Number: 3e122819

Valid from: Tue Dec 31 17:28:25 CST 2002 to: Wed Dec 31 17:28:25 CST 2003
Certificate fingerprints:
MD5: C0:49:F0:8B:5D:77:85:4B:97:EB:1C:B4:B2:05:4C:E7
SHA1:
EA:A5:2E:5A:CF:02:79:79:C8:87:1E:9B:57:E2:54:F3:35:4D:C1:45

3. Exportar y examinar un certificado autofirmado.

keytool -export -alias server -keystore keystore -rfc -file server.cer

Enter keystore password: **password**
Certificate stored in file <server.cer>

El certificado creado se muestra en la figura A-1.

Figura A-1. Certificado del servidor.

Con estos pasos, se ha creado el certificado del servidor. El certificado del cliente se crea de la misma forma.

Es necesario que estos certificados estén instalados en las máquinas del servidor y cliente respectivamente.

Apéndice

B

Código Fuente

Principal.java

```
/* Principal.java */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

public class Principal extends HttpServlet {

        Connection conn = null;
        Vector vectorDirectorio=null;

        public void init (ServletConfig config) throws ServletException {

                super.init(config);

                String url=new String("jdbc:odbc:Contenedor");

                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                } catch(ClassNotFoundException ex) {
                        System.out.println("Error al cargar el driver");
                        System.out.println(ex.getMessage());
                }

                try {
                        conn=DriverManager.getConnection(url,"","");
                } catch (SQLException sqlEx) {
                        System.out.println("Se ha producido un error al establecer "+"la
conexión con: "+url);
                        System.out.println(sqlEx.getMessage());
                }
                System.out.println("Iniciando URL");
        } // fin del método init()

        public void destroy () {
                super.destroy();
                System.out.println("Cerrando conexion...");
                try {
                        conn.close();
                } catch(SQLException ex){
                        System.out.println("No se pudo cerrar la conexion");
                        System.out.println(ex.getMessage());
                }
        } // fin del método destroy()

        public void doGet (HttpServletRequest req, HttpServletResponse resp)throws
ServletException, IOException {

                resp.setHeader("Refresh", "300");
                // Consulta a la base de datos para obtener las URL de los hospitales
                if(obtenerURL()==0) {
                        mostrarListaURL();
                        //Llama al servlet ViewResource para que se conecte al sitio web
del Hospital
                        RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/servlet/ViewResource");
                        dispatcher.forward(req,resp);
                }
                else
                        if(obtenerURL()==-3)
                                resp.sendError(500, "No se ha encontrado URL");
                        else
                                resp.sendError(500, "Se ha producido un error en el acceso
" +"a la base de datos");

        } // fin del método doGet()
```



```
        public int obtenerURL() {
                Statement stmt = null;
                ResultSet rs = null;

                try {
                        stmt=conn.createStatement();
                        String buscar = "SELECT  URL *-
                                                "FROM Hospital ";
                        rs=stmt.executeQuery(buscar);
                        vectorDirectorio=new Vector();

                        while (rs.next()) {
                                Directorio temp = new Directorio();
                                temp.setURL(rs.getString("URL"));
                                vectorDirectorio.addElement(temp);
                        }

                        if(vectorDirectorio.size()==0)
                                return -3;
                        return 0;
                } catch (SQLException sql) {
                        System.out.println("Se produjo un error al crear la declaración");
                        System.out.println(sql.getMessage());
                        return -1;
                } finally {
                        if(stmt!=null) {
                                try {
                                        stmt.close();
                                } catch(SQLException e) {
                                        System.out.println("Error al cerrar la
declaración");
                                        System.out.println(e.getMessage());
                                        return -2;
                                }
                        }
                }
        } // fin del método obtenerURL()

        public void mostrarListaURL() {
                Directorio dir = null;
                String URL = null;
                String clave = null;

                //Almacena la información obtenida en el contexto.
                ServletContext contexto = getServletContext();

                for (int i=0; i<vectorDirectorio.size();i++) {
                        dir=(Directorio)vectorDirectorio.elementAt(i);

                        URL = dir.getURL();
                        Integer j = new Integer(i);
                        clave = j.toString();
                        contexto.setAttribute(clave, URL);
                }
                System.out.println("Se ha grabado el contexto");

        } // fin del método mostrarListaURL()

        public String getServletInfo() {
                return "Obtiene la URL de un hospital";
        }
} // fin de la clase Principal
```

ViewResource.java

```
/* ViewResource.java */

import java.io.*;
import java.net.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.xml.parser.*;

public class ViewResource extends HttpServlet {

        Connection conn = null;
        private String Nombre = null;
        private String Tipo = null;
        private String Especialidad = null;
        private String Direccion = null;
        private String Localidad = null;
        private String Delegacion = null;
        private String Telefono = null;
        private String url = null;
        private String path = null;
        private String JDBCDriver = null;
        private String JDBCURL = null;
        private String Tabla_Servicio = null;
        private String Tabla_Ingreso = null;
        private String ID_Servicio = null;
        private String ID_Ingreso = null;
        private String Nombre_Servicio = null;
        private String Camas_Disponibles = null;

        public void init (ServletConfig config) throws ServletException {

                super.init(config);

                String url=new String("jdbc:odbc:Contenedor");

                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                } catch(ClassNotFoundException ex) {
                        System.out.println("Error al cargar el driver");
                        System.out.println(ex.getMessage());
                }

                try {
                        conn=DriverManager.getConnection(url,"","");
                } catch (SQLException sqlEx) {
                        System.out.println("Se ha producido un error al establecer la
conexión con: "+url);
                        System.out.println(sqlEx.getMessage());
                }
                System.out.println("Iniciando Actualización");
        } // fin del método init()

        public void destroy () {
                super.destroy();
                System.out.println("Cerrando conexion...");
                try {
                        conn.close();
                } catch(SQLException ex){
                        System.out.println("No se pudo cerrar la conexion");
                        System.out.println(ex.getMessage());
                }
        } // fin de destroy()

        public void doGet(HttpServletRequest req, HttpServletResponse res)
                                                        throws ServletException,
IOException {

                int k, l;

                ServletContext contexto = getServletContext();

                String clave = null;
```



```
                for (int i=0; i<2; i++) {
                        Integer j = new Integer(i);
                        clave = j.toString();
                        url = (String)contexto.getAttribute(clave);
                        path="D: \\Laura\\Proyecto\\DTD\\"+url.substring(6);

                        leeDTD(path);
                        k = ingresaHospital();
                        l = ingresaBase();
                        if(k==0 && l==0)
                                enviaParametros(req, res);
                        else
                                if(k==-1 && l== 1)
                                        res.sendError(500, "No se ha encontrado URL");
                        else
                                        res.sendError(500, "Se ha producido un error en el
acceso a la base de datos");

                }
        }

        public int leeDTD(String dir) {

                try {
                        String dtdfile = dir.concat("\\Hosp.dtd");
                        InputStream is = new FileInputStream(dtdfile);
                        Parser parser = new Parser(dtdfile);
                DTD dtd = parser.readDTDStream(is);
                if (parser.getNumberOfErrors() > 0) {
                  System.exit(1);
                        }

                Enumeration enum;

                        enum = dtd.getAttributeDeclarations("Hospital");
                        while(enum.hasMoreElements()) {
                                AttDef attra = (AttDef)enum.nextElement();
                                String attName = attra.getName();
                                String attValue = attra.getDefaultStringValue();
                                if (attName.compareTo("nombre") == 0)
                                        Nombre = attValue;
                                else if (attName.compareTo("tipo") == 0)
                                        Tipo = attValue;
                                else if (attName.compareTo("especialidad") == 0)
                                        Especialidad = attValue;
                                else if (attName.compareTo("telefono") == 0)
                                        Telefono = attValue;
                                else if (attName.compareTo("JDBCDriver") == 0)
                                        JDBCDriver = attValue;

                                else if (attName.compareTo("JDBCURL") == 0)
                                        JDBCURL = attValue;
                        }
                        enum = dtd.getAttributeDeclarations("Domicilio");
                        while(enum.hasMoreElements()) {
                                AttDef attra = (AttDef)enum.nextElement();
                                String attName = attra.getName();
                                String attValue = attra.getDefaultStringValue();
                                if (attName.compareTo("direccion") == 0)
                                        Direccion = attValue;
                                else if (attName.compareTo("localidad") == 0)
                                        Localidad = attValue;
                                else if (attName.compareTo("delegacion") == 0)
                                        Delegacion = attValue;
                        }
                        Localidad = Localidad.concat(" "+Delegacion);
                        Direccion = Direccion.concat(" "+Localidad);

                        enum = dtd.getAttributeDeclarations("Servicio");
                        while(enum.hasMoreElements()) {
                                AttDef attra = (AttDef)enum.nextElement();
                                String attName = attra.getName();
                                String attValue = attra.getDefaultStringValue();
                                if (attName.compareTo("nombre_tabla") == 0)
                                        Tabla_Servicio = attValue;
                                else if (attName.compareTo("nombre_id") == 0)
```

```
                    else if (attName.compareTo("nombre_serv") == 0)
                        Nombre_Servicio = attValue;

                    else if (attName.compareTo("camas_disp") == 0)
                        Camas_Disponibles = attValue;
                }
                enum = dtd.getAttributeDeclarations("Ingreso");
                while(enum.hasMoreElements()) {
                    AttDef attra = (AttDef)enum.nextElement();
                    String attName = attra.getName();
                    String attValue = attra.getDefaultStringValue();
                    if (attName.compareTo("nombre_tabla") == 0)
                        Tabla_Ingreso = attValue;
                    else if (attName.compareTo("nombre_id") == 0)
                        ID_Ingreso = attValue;
                }
            }
            catch (Exception e) {
                e.printStackTrace();
                return -2;
            }
            return 0;
        }
        public int ingresaHospital() {

            Statement stmt=null;
            String actualizar;
            int numeroFilasActualizadas=0;

            try {
                stmt = conn.createStatement();
                actualizar = "UPDATE Hospital SET Nom_Hosp='"+Nombre+"',
                             Tipo='"+Tipo+"', Especialidad='',
                              Especialidad='', Direccion='"+Direccion+"',
                              Telefono='"+Telefono+"' " +
                                        "WHERE URL='"+url+"'";
                numeroFilasActualizadas = stmt.executeUpdate(actualizar);
                if(numeroFilasActualizadas!=1) return -1;
            } catch (SQLException sql) {
                System.out.println("Se produjo un error creando sentencia SQL");
                System.out.println(sql.getMessage());
                return -2;
            } finally {
                if(stmt!=null) {
                    try {
                        stmt.close();
                    } catch(SQLException e){
                        System.out.println("Error cerrando sentencia SQL");
                        System.out.println(e.getMessage());
                        return -3;
                    }
                }
                return 0;
            } // fin finally
        } // fin método ingresaHospital()

        public int ingresaBase() {

            Statement stmt=null;
            String actualizar;
            int numeroFilasActualizadas=0;

            try {
                stmt = conn.createStatement();
                actualizar = "UPDATE BaseDatos SET JDBCDriver='"+JDBCDriver+"',
                             JDBCURL='"+JDBCURL+"', Tabla_serv='"+
                             Tabla_Servicio+"', id_serv='"+ID_Servicio+"',
                             Nombre_serv='"+Nombre_Servicio+"', Camas_disp='"+
                             Camas_Disponibles+"',
                             Tabla_ing='"+Tabla_Ingreso+"',
                             Id_ing='"+ID_Ingreso+"' " +
                             "WHERE URL='"+url+"'";
                numeroFilasActualizadas = stmt.executeUpdate(actualizar);
                if(numeroFilasActualizadas!=1) return -1;
            } catch (SQLException sql) {
                System.out.println("Se produjo un error creando sentencia SQL");
```



```
            } finally {
                if(stmt!=null) {
                    try {
                        stmt.close();
                    } catch(SQLException e){
                        System.out.println("Error cerrando sentencia SQL");
                        System.out.println(e.getMessage());
                        return -3;
                    }
                }
                return 0;
            }
        } // fin método ingresaBase()

        public void enviaParametros(HttpServletRequest req, HttpServletResponse resp) {

            Disponibilidad disp = new Disponibilidad();
            Hashtable serv = new Hashtable();

            serv = disp.execute(JDBCDriver, JDBCURL, url, Tabla_Servicio,
             ID_Servicio, Nombre_Servicio, Camas_Disponibles);

            String clave = null;
            int controlador = 0;

            Integer j = new Integer(0);
            clave = j.toString();
            url = (String)serv.get("url"+clave);
            String id_serv = (String)serv.get("id_serv"+clave);
            String nom_serv = (String)serv.get("nom_serv"+clave);
            String cam = (String)serv.get("camas"+clave);

            ServletContext contexto = getServletContext();

            contexto.setAttribute("url"+clave, url);
            contexto.setAttribute("id_serv"+clave, id_serv);
            contexto.setAttribute("nom_serv"+clave, nom_serv);
            contexto.setAttribute("camas"+clave, cam);

            for (int i=1; controlador == 0; i++) {

                j = new Integer(i);
                clave = j.toString();
                url = (String)serv.get("url"+clave);
                id_serv = (String)serv.get("id_serv"+clave);
                nom_serv = (String)serv.get("nom_serv"+clave);
                cam = (String)serv.get("camas"+clave);

                if(url == null)
                    controlador = 1;
                else {
                    contexto.setAttribute("url"+clave, url);
                    contexto.setAttribute("id_serv"+clave, id_serv);
                    contexto.setAttribute("nom_serv"+clave, nom_serv);
                    contexto.setAttribute("camas"+clave, cam);
                }
            }

            RequestDispatcher dispatcher =
             getServletContext().getRequestDispatcher("/servlet/ActualizaServicios");
            try {
                dispatcher.forward(req,resp);
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }

        } // fin del método enviaParametros()

    } // fin de la clase ViewResource
```

ActualizaServicios.java

```
/* ActualizaServicios.java */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

public class ActualizaServicios extends HttpServlet {

        Connection conn = null;
        Vector VectorDirectorio=null;
        String url = null;
        String id_serv = null;
        String nom_serv = null;
        String can = null;
        Integer canas_disp = null;
        Integer canas_obt = null;

        public void init (ServletConfig config) throws ServletException {

                super.init(config);

                String url=new String("jdbc:odbc:Contenedor");

                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                } catch(ClassNotFoundException ex) {
                        System.out.println("Error al cargar el driver");
                        System.out.println(ex.getMessage());
                }

                try {
                        conn=DriverManager.getConnection(url,"","");
                } catch (SQLException sqlEx) {
                        System.out.println("Se ha producido un error al establecer "+"la
conexión con: "+url);
                        System.out.println(sqlEx.getMessage());
                }
                System.out.println("Iniciando ActualizaServicios");
        } // fin del método init()

        public void destroy () {
                super.destroy();
                System.out.println("Cerrando conexion...");
                try {
                        conn.close();
                } catch(SQLException ex){
                        System.out.println("No se pudo cerrar la conexion");
                        System.out.println(ex.getMessage());
                }
        } // fin de destroy()

        public void doGet (HttpServletRequest req, HttpServletResponse resp)throws
ServletException, IOException {

                ServletContext contexto = getServletContext();
                String clave = null;
                int controlador = 0;

                Integer j = new Integer(0);
                clave = j.toString();
                url = (String)contexto.getAttribute("url"+clave);
                id_serv = (String)contexto.getAttribute("id_serv"+clave);
                nom_serv = (String)contexto.getAttribute("nom_serv"+clave);
                can = (String)contexto.getAttribute("canas"+clave);

                canas_disp = new Integer(can);

                for (int i=1; controlador == 0; i++) {

                        int resultado = existeServicio();

                        if(resultado >= 0 && canas_disp.compareTo(canas_obt) >= 0)
```



```
                        actualizaServicio();
                else {
                        if(resultado == -3) {
                                int resul = agregaServicio();
                                if(resul == 0) {
                                System.out.println ("Base De datos actualizada");
                                }
                                else
                                        if(resul ==-1) {
                                resp.sendError(500, "No hay recursos disponibles");
                                }
                                else
resp.sendError(501, "Se ha producido un error en el acceso " +"a la base de datos");
                        }
                }

                j = new Integer(i);
                clave = j.toString();
                url = (String)contexto.getAttribute("url"+clave);
                id_serv = (String)contexto.getAttribute("id_serv"+clave);
                nom_serv = (String)contexto.getAttribute("nom_serv"+clave);
                can = (String)contexto.getAttribute("canas"+clave);

                if(url == null)
                        controlador = 1;
                else
                        canas_disp = new Integer(can);
        }
  } // fin del método doGet()

  public int existeServicio() {

        Statement stmt = null;
        ResultSet rs = null;
        Directorio dir=null;

        try {
                stmt=conn.createStatement();
                String buscar = "SELECT Canas"+
                                        " FROM Servicios "+
                                        "WHERE ((Id_Serv)='"+id_serv+"') "+
                                        "AND ((URL)='"+url+"')";

                rs=stmt.executeQuery(buscar);
                vectorDirectorio=new Vector();

                while (rs.next()) {
                        Directorio temp = new Directorio();
                        temp.setCanas(rs.getInt("Canas"));
                        vectorDirectorio.addElement(temp);
                }

                if(vectorDirectorio.size()==0)
                        return -3;

                for (int i=0; i<vectorDirectorio.size();i++) {
                        dir=(Directorio)vectorDirectorio.elementAt(i);
                        canas_obt = new Integer(dir.getCanas());
                }
                return 0;

        } catch (SQLException sql) {
                System.out.println("Se produjo un error al crear sentencia SQL");
                System.out.println(sql.getMessage());
                return -1;
        } finally {
                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e) {
                        System.out.println("Error al cerrar sentencia SQL");
                                System.out.println(e.getMessage());
                                return -2;
                        }
                }
        }
```

```
}// fin del método existeServicio
public int actualizaServicio() {

        Statement stmt=null;
        ResultSet rs = null;
        String actualizar;
        int numeroFilasActualizadas=0;

        try {
                stmt = conn.createStatement();
                actualizar = "UPDATE Servicios SET Camas='"+camas_disp+"'" +
                                "WHERE ((ID_Serv)='"+id_serv+"') " +
                                "AND ((URL)='"+url+"') ";

                numeroFilasActualizadas = stmt.executeUpdate(actualizar);
                if(numeroFilasActualizadas!=1) return -1;
        } catch (SQLException sql) {
                System.out.println("Se produjo un error creando sentencia SQL");
                System.out.println(sql.getMessage());
                return -2;
        } finally {
                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e){
                                System.out.println("Error cerrando sentencia SQL");
                                System.out.println(e.getMessage());
                                return -3;
                        }
                }
                return 0;
        }
} // fin método actualizaServicio()

public int agregaServicio() {
        Statement stmt=null;
        int numeroFilasActualizadas=0;

        try {
                stmt = conn.createStatement();
                numeroFilasActualizadas = stmt.executeUpdate("INSERT INTO"+"
Servicios VALUES"+"('"+id_serv+"','"+nom_serv+"','"+camas_disp+"',0,0,0,'"+url+"')");
                if(numeroFilasActualizadas!=1) return -1;
        } catch (SQLException sql) {
                System.out.println("Se produjo un error creando Statement");
                System.out.println(sql.getMessage());
                return -2;
        } finally {
                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e){
                                System.out.println("Error cerrando Statement");
                                System.out.println(e.getMessage());
                                return -3;
                        }
                }
                return 0;
        }
} // fin método agregaServicio()

}// fin de la clase ActualizaServicios
```


Disponibilidad.java

```
/* Disponibilidad.java */

import java.sql.*;
import java.util.*;

public class Disponibilidad {

        Connection conn = null;
        Vector vectorDirectorio=null;

        public Disponibilidad() {
                super();
        }

        public int buscaServicios(String id_serv, String nom_serv, String camas, String
tabla) {

                Statement stmt=null;
                ResultSet rs = null;
                int numeroFilasActualizadas=0;

                try {
                        stmt = conn.createStatement();

                        String atributos ="SELECT " +id_serv+ ", "  +nom_serv+ ", "
                                        +camas+ " FROM " +tabla;

                        rs=stmt.executeQuery(atributos);
                        vectorDirectorio=new Vector();

                        while (rs.next()) {
                                Directorio temp = new Directorio();
                                temp.setID_Serv(rs.getString(id_serv));
                                temp.setNom_Serv(rs.getString(nom_serv));
                                temp.setCamas(rs.getInt(camas));
                                vectorDirectorio.addElement(temp);
                        }

                        if(vectorDirectorio.size()==0)
                                return -3;
                        return 0;

                } catch (SQLException sql) {
                        System.out.println("Se produjo un error creando sentencia SQL");
                        System.out.println(sql.getMessage());
                        return -1;
                } finally {
                        if(stmt!=null) {
                                try {
                                        stmt.close();
                                } catch(SQLException e){
                                        System.out.println("Error cerrando sentencia SQL");
                                        System.out.println(e.getMessage());
                                        return -2;
                                }
                        }
                }
        } // fin método buscaServicios()

        public Hashtable enviaServicios(String url) {

                Directorio dir = null;
                String clave = null;
                String id_serv = null;
                String nom_serv = null;
                int camas = 0;
                Hashtable serv = new Hashtable();

                for (int i=0; i<vectorDirectorio.size();i++) {

                        dir=(Directorio)vectorDirectorio.elementAt(i);
                        id_serv = dir.getID_Serv();
                        nom_serv = dir.getNom_Serv();
```

```
                    canas = dir.getCanas();

                    Integer can = new Integer(canas);

                    Integer j = new Integer(i);
                    clave = j.toString();

                    // Se guardan las variables
                    serv.put("url"+clave, url);
                    serv.put("id_serv"+clave, id_serv);
                    serv.put("nom_serv"+clave, nom_serv);
                    serv.put("canas"+clave, can.toString());
                }
                return serv;
        } // fin del método enviaServicios()

        public Hashtable ejecuta(String JDBCDriver, String JDBCURL, String url, String
tabla, String id_serv, String nom_serv, String canas) {
                Hashtable servicio = new Hashtable();
                try {
                        Class.forName(JDBCDriver);
                        conn = DriverManager.getConnection(JDBCURL," "," ");

                        int resultado =  buscaServicios(id_serv, nom_serv, canas, tabla);
                        if(resultado == 0) {
                                for (int i=0; i<vectorDirectorio.size();i++)
                                        servicio = enviaServicios(url);

                        }
                        else if(resultado == -1) {
                                System.out.println("No hay recursos disponibles");
                        }
                        else
                                System.out.println("Se produjo un error en el acceso a la
base de datos");

                        conn.close();
                }
                catch(Exception e) {
                        System.out.println("Se ha producido un error al establecer la
conexión con: "+JDBCURL);
                }
                return servicio;
        }

} // fin de la clase Disponibilidad
```

Directorio.java

```
/*Programa: Directorio.java*/

public class Directorio {

        private String url;
        private String hospital;
        private String tipo;
        private String especialidad;
        private String direccion;
        private int telefono;
        private String servicio;
        private String nombre;
        private int camas;
        private int cunas;
        private int incubadoras;
        private int quirofanos;

        // Métodos para establecer los datos

        public void setURL(String u) { url=u; }
        public void setNom_Hosp(String hosp) { hospital=hosp; }
        public void setTipo(String tip) { tipo=tip; }
        public void setEspecialidad(String esp) { especialidad=esp; }
        public void setDireccion(String dir) { direccion=dir; }
        public void setTelefono(int tel) { telefono=tel; }
        public void setID_Serv(String serv) { servicio=serv; }
```



```
        public void setNom_Serv(String nom) { nombre=nom; }
        public void setCamas(int cam) { camas=cam; }
        public void setCunas(int cun) { cunas=cun; }
        public void setIncubadoras(int inc) { incubadoras=inc; }
        public void setQuirofanos(int quir) { quirofanos=quir; }

        // Métodos de recuperación de datos

        public String getURL() { return url; }
        public String getNom_Hosp() { return hospital; }
        public String getTipo() { return tipo; }
        public String getEspecialidad() { return especialidad; }
        public String getDireccion() { return direccion; }
        public int getTelefono() { return telefono; }
        public String getID_Serv() { return servicio; }
        public String getNom_Serv() { return nombre; }
        public int getCamas() { return camas; }
        public int getCunas() { return cunas; }
        public int getIncubadoras() { return incubadoras; }
        public int getQuirofanos() {return quirofanos; }

} //fin de la clase Directorio
```

B.2. Subsistema de Asignación de Recursos

ValidaUsuario.java

```
//ValidaUsuario.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

public class ValidaUsuario extends HttpServlet {

        Connection conn = null;
        Vector vectorDirectorio=null;
        private String clave = null;
        String Error = null;
        HttpSession session;

        String tipo = null;
        String servicio = null;
        String hospital = null;
        String identificador = null;


        public void init(ServletConfig config) throws ServletException {
                super.init(config);
                String url=new String("jdbc:odbc:Contenedor");

                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                } catch(ClassNotFoundException ex) {
                        System.out.println("Error al cargar el driver");
                        System.out.println(ex.getMessage());
                }

                try {
                        conn=DriverManager.getConnection(url,"","");
                } catch (SQLException sqlEx) {
                        System.out.println("Se ha producido un error al establecer "+"la
conexión con: "+url);
                        System.out.println(sqlEx.getMessage());
                }
                System.out.println("Iniciando ProcesaPeticion");
        } // fin del método init()

        public void destroy () {
                super.destroy();
```

```
            try {
                  conn.close();
            } catch(SQLException ex){
                  System.out.println("No se pudo cerrar la conexion");
                  System.out.println(ex.getMessage());
            }
      } // fin del método destroy()

      public void doGet (HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
      sesion = req.getSession();
      clave = req.getParameter("Clave");

      if(clave==null) {
            Error = "Se ha producido un error en la lectura de la solicitud";
                  enviaError(resp, Error);
                  return;
            }

      if ( existeUsuario(clave) == 0) {
            if (!sesion.isNew()) {
                  java.util.Date horaAntes = new
                  java.util.Date(System.currentTimeMillis() - 60*60*1000);
                  java.util.Date cincoAntes = new
                  java.util.Date(System.currentTimeMillis() - 5*60*1000);
                  java.util.Date creada = new
                  java.util.Date(sesion.getCreationTime());
                  java.util.Date ingreso = new
                  java.util.Date(sesion.getLastAccessedTime());

                  if (creada.before(horaAntes) || ingreso.before(cincoAntes) ||
(clave.compareTo(sesion.getAttribute("clave")) != 0)) {
                        sesion.invalidate();
                        sesion = req.getSession();
                  }
            }
            System.out.println("Session id creada: " +sesion.getId());

            tipo = (String)sesion.getAttribute("tipo");
            servicio = (String)sesion.getAttribute("servicio");
            hospital = (String)sesion.getAttribute("hospital");

            if (tipo == null && servicio == null) {
                  sesion.setAttribute("clave", clave);
                  GenerarPaginaWML(resp);
            }
            else if ( hospital == null)
                        enviaProximaPeticion(resp);

            }
            else {
                  Error = ("Clave de acceso incorrecta.");
                  enviaError(resp, Error);
            }

      } // fin del método doGet()

public int existeUsuario(String clave) {

            Statement stmt=null;
            ResultSet rs = null;
            String buscar;
            int numeroFilasActualizadas=0;

            try {
                  stmt = conn.createStatement();

                  buscar = "SELECT  Clave "+
                        "FROM Usuarios "+
                        "WHERE Clave = '"+clave+"' ";

                  rs=stmt.executeQuery(buscar);
                  vectorDirectorio=new Vector();

                  while (rs.next()) {
                        Directorio temp = new Directorio();
```



```
                        vectorDirectorio.addElement(temp);
                  }

                  if(vectorDirectorio.size()==0)
                        return -3;
                  return 0;

            } catch (SQLException sql) {
                  System.out.println("Se produjo un error creando sentencia SQL");
                  System.out.println(sql.getMessage());
                  return -1;
            } finally {

                  // Se cierra el Statement
                  if(stmt!=null) {
                        try {
                              stmt.close();
                        } catch(SQLException e){
                              System.out.println("Error cerrando sentencia SQL");
                              System.out.println(e.getMessage());
                              return -2;
                        }
                  }
            }
} // fin método existeUsuario()

public void GenerarPaginaWML (HttpServletResponse resp) {
      resp.setContentType("text/vnd.wap.wml");
PrintWriter out = null;
try {
      out=resp.getWriter();
      }
catch (IOException io) {
            ;
      }

      // Se genera el contenido de la página WML
      out.println("<?xml version=\"1.0\"?>");
      out.println("<!DOCTYPE wml PUBLIC \"-//OPENWAVE.COM//DTD WML 1.3//EN\"
                  \"http://www.openwave.com/dtd/wml13.dtd\"> ");
      out.println("<wml>");
      out.println("<template>");
      out.println("<do type=\"options\" label=\"Back\">");
      out.println("<prev />");
      out.println("</do>");
      out.println("</template>");
      out.println("<card id=\"index\" title=\"Seleccione tipo\">");
      out.println("<onevent type=\"onenterforward\">");
      out.println("<refresh>");
      out.println("<setvar name=\"tipo\" value=\"\" />");
      out.println("</refresh>");
      out.println("</onevent>");
      out.println("<do type=\"options\">");
      out.println("<noop /> </do> <p>");
      out.println("<select name=\"tipo\">");
      out.println("<option optick=\"#serv\" value=\"IMSS\">IMSS</option>");
      out.println("<option optick=\"#serv\" value=\"ISSTE\">ISSTE</option>");
      out.println("<option optick=\"#serv\" value=\"Cruz Roja\">Cruz
                  Roja</option>");
      out.println("<option optick=\"#serv\"
                  value=\"Privado\">Privado</option>");
      out.println("<option optick=\"#serv\"
                  value=\"Salubridad\">Salubridad</option>");
      out.println("</select> </p> </card>");
      out.println("<card id=\"serv\" title=\"Seleccione especialidad\">");
      out.println("<onevent type=\"onenterforward\">");
      out.println("<refresh>");
      out.println("<setvar name=\"servicio\" value=\"\" />");
      out.println("</refresh> </onevent> <p>");
      out.println("<select name=\"servicio\">");
      out.println("<option optick=\"#emv\"
                  value=\"Urgencias\">Urgencias</option>");
      out.println("<option optick=\"#emv\"
                  value=\"Quemaduras\">Quemaduras</option>");
      out.println("<option optick=\"#emv\"
                  value=\"Traumatismo\">Traumatismo</option>");
```

```
            out.println("<option onpick=\"$env\"
                        value=\"Ginecologia\">Ginecologia</option>");
            out.println("<option onpick=\"$env\"
                        value=\"General\">General</option>");
            out.println("</select> </p> </card>");
            out.println("<card id=\"env\" title=\"Buscar Disponibilidad\">");
            out.println("<p>");
            out.println("Tipo de Hospital: $(tipo)");
            out.println("<br />");
            out.println("Especialidad: $(servicio)");
            out.println("<br /> </p>");
            out.println("<do type=\"accept\">");
            out.println("<go ");

out.println("href=\"http://148.204.211.134:8080/wap/servlet/ProcesaPeticion?Tipo=
            $(tipo)&amp;Servicio=$(servicio)\"");
            out.println("/> </do> </card> </wml>");
            out.flush();
            out.close();

} // fin de GenerarPaginaWML ()


public void enviaProcesaPeticion (HttpServletResponse resp) {
        resp.setContentType("text/vnd.wap.wml");
PrintWriter out = null;
try {
        out=resp.getWriter();
        }
catch (IOException io) {
                ;
        }

        // Se genera el contenido de la página WML
        out.println("<?xml version=\"1.0\"?>");
        out.println("<!DOCTYPE wml PUBLIC \"-//OPENWAVE.COM//DTD WML 1.3//EN\"
                    \"http://www.openwave.com/dtd/wml13.dtd\">");
        out.println("<wml>");
        out.println("<card id=\"Advertencia\" title=\"Advertencia\">");
        out.println("<onevent type=\"onenterforward\">");
        out.println("<refresh>");
        out.println("<setvar name=\"tipo\" value=\""+tipo+"\" />");
        out.println("<setvar name=\"servicio\" value=\""+servicio+"\" />");
        out.println("</refresh>");
        out.println("</onevent>");
        out.println("<p>");
        out.println("Tipo de Hospital: $(tipo)");
        out.println("<br />");
        out.println("Especialidad: $(servicio)");
        out.println("<br />");
        out.println("Continuar  peticion OK de lo contrario Cancelar");
        out.println("</p>");
        out.println("<do type=\"accept\">");
        out.println("<go ");
out.println("href=\"http://148.204.211.134:8080/wap/servlet/ProcesaPeticion?Tipo=
            $(tipo)&amp;Servicio=$(servicio)\"");
        out.println("/> </do>");
        out.println("<do type=\"options\" label=\"Cancelar\">");
        out.println("<go ");
        out.println("href=\"#seleccionar\"");
        out.println("/> </do> </card>");
        out.println("<card id=\"seleccionar\" title=\"Seleccione tipo\">");
        out.println("<onevent type=\"onenterforward\">");
        out.println("<refresh>");
        out.println("<setvar name=\"Tipo\" value=\"\" />");
        out.println("</refresh>");
        out.println("</onevent>");
        out.println("<do type=\"options\">");
        out.println("<noop /> </do> <p>");
        out.println("<select name=\"tipo\">");
        out.println("<option onpick=\"#servi\" value=\"IMSS\">IMSS</option>");
        out.println("<option onpick=\"#servi\" value=\"ISSTE\">ISSTE</option>");
        out.println("<option onpick=\"#servi\" value=\"Cruz Roja\">Cruz
                    Roja</option>");
        out.println("<option onpick=\"#servi\"
                    value=\"Privado\">Privado</option>");
        out.println("<option onpick=\"#servi\"
```



```
                    value=\"Salubridad\">Salubridad</option>");
        out.println("</select> </p> </card>");
        out.println("<card id=\"servB\" title=\"Selecciona especialidad\">");
        out.println("<onevent type=\"onenterforward\">");
        out.println("<refresh>");
        out.println("<setvar name=\"servicio\" value=\"\" />");
        out.println("</refresh> </onevent> <p>");
        out.println("<select name=\"servicio\">");
        out.println("<option onpick=\"#envi\"
                    value=\"Urgencias\">Urgencias</option>");
        out.println("<option onpick=\"#envi\"
                    value=\"Quemaduras\">Quemaduras</option>");
        out.println("<option onpick=\"#envi\"
                    value=\"Traumatismo\">Traumatismo</option>");
        out.println("<option onpick=\"#envi\"
                    value=\"Ginecologia\">Ginecologia</option>");
        out.println("<option onpick=\"#envi\"
                    value=\"General\">General</option>");
        out.println("</select> </p> </card>");
        out.println("<card id=\"envi\" title=\"Buscar Disponibilidad\">");
        out.println("<p>");
        out.println("Tipo de Hospital: $(tipo)");
        out.println("<br />");
        out.println("Especialidad: $(servicio)");
        out.println("<br /> </p>");
        out.println("<do type=\"accept\">");
        out.println("<go ");

out.println("href=\"http://148.204.211.134:8080/wap/servlet/ProcesaPeticion?Tipo=
            $(tipo)&amp;Servicio=$(servicio)\"");
        out.println("/> </do> </card> </wml>");
        out.flush();
        out.close();

} // fin de enviaProcesaPeticion ()

public void enviaError(HttpServletResponse resp, String error) {
        resp.setContentType("text/vnd.wap.wml");
PrintWriter out = null;
try {
        out=resp.getWriter();
        }
catch (IOException io) {
                ;
        }

        // Se genera el contenido de la página WML
        out.println("<?xml version=\"1.0\"?>");
        out.println("<!DOCTYPE wml PUBLIC \"-//OPENWAVE.COM//DTD WML 1.3//EN\"
                    \"http://www.openwave.com//dtd/wml13.dtd\"> ");
        out.println("<wml>");
        out.println("<card id=\"resp\" title=\"Error\">");
        out.println("<p>");
        out.println(error);
        out.println("</p></card> </wml>");
        out.flush();
        out.close();

} // fin de enviaError()

public String getServletInfo() {
        return "Genera WML";
} // fin del método getServletInfo()

}
```

ProcesaPeticion.java

```
//ProcesaPeticion.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

public class ProcesaPeticion extends HttpServlet {

        Connection conn = null;
        Vector vectorDirectorio=null;
        private String tipo = null;
        private String servicio = null;
        String Error = null;
        HttpSession session;

        public void init(ServletConfig config) throws ServletException {
                super.init(config);
                String url=new String("jdbc:odbc:Contenedor");

                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                } catch(ClassNotFoundException ex) {
                        System.out.println("Error al cargar el driver");
                        System.out.println(ex.getMessage());
                }

                try {
                        conn=DriverManager.getConnection(url,"","");
                } catch (SQLException sqlEx) {
                        System.out.println("Se ha producido un error al establecer "+"la
                                        conexión con: "+url);
                        System.out.println(sqlEx.getMessage());
                }
                System.out.println("Iniciando ProcesaPeticion");
        } // fin del método init()

        public void destroy () {
                super.destroy();
                System.out.println("Cerrando conexion...");
                try {
                        conn.close();
                } catch(SQLException ex){
                        System.out.println("No se pudo cerrar la conexion");
                        System.out.println(ex.getMessage());
                }
        } // fin del método destroy()

        public void doPost (HttpServletRequest req, HttpServletResponse resp) throws
                ServletException, IOException {

        tipo = req.getParameter("Tipo");
        servicio = req.getParameter("Servicio");

        session.setAttribute("tipo", tipo);
        session.setAttribute("Servicio", servicio);

        if(tipo==null && servicio==null) {
                Error = "Se ha producido un error en la lectura de la solicitud";
                        enviaError(resp, Error);
                        return;
                }

                int resultado;
                resultado =  buscaRecursos(tipo, servicio);
                if(resultado == 0)
                GenerarPaginaWML(resp);
        else if(resultado == -3) {
                Error = ("No hay recursos disponibles");
                        enviaError(resp, Error);
                }
                else {
                        Error = ("Se ha producido un error en el acceso a la base de
                                datos");
                        enviaError(resp, Error);
                }

        } // fin del método doPost()

        public void doGet (HttpServletRequest req, HttpServletResponse resp) throws
                ServletException, IOException {
        session = req.getSession();
        doPost(req,resp);
        } // fin del método doPost()

        public int buscaRecursos(String tipo, String servicio) {

                Statement stmt=null;
                ResultSet rs = null;
                String buscar;
                int numeroFilasActualizadas=0;

                try {
                        stmt = conn.createStatement();

                        buscar = "SELECT  DISTINCT Nom_Hosp "+
                                                        "FROM Hospital "+
                                                        "WHERE Tipo = '"+tipo+"' "+
                                                        "AND URL "+
                                                        "IN (SELECT URL "+
                                                        "FROM Servicios "+
                                                        "WHERE
                                                         Nom_Serv='"+servicio+"' "+
                                                        "AND Camas > 0) ";

                        rs=stmt.executeQuery(buscar);
                        vectorDirectorio=new Vector();

                        while (rs.next()) {
                                Directorio temp = new Directorio();
                                temp.setNom_Hosp(rs.getString("Nom_Hosp"));
                                vectorDirectorio.addElement(temp);
                        }

                        if(vectorDirectorio.size()==0)
                                return -3;
                        return 0;

                } catch (SQLException sql) {
                        System.out.println("Se produjo un error creando sentencia SQL");
                        System.out.println(sql.getMessage());
                        return -1;
                } finally {

                        // Se cierra el Statement
                        if(stmt!=null) {
                                try {
                                        stmt.close();
                                } catch(SQLException e){
                                        System.out.println("Error cerrando sentencia SQL");
                                        System.out.println(e.getMessage());
                                        return -2;
                                }
                        }
                }
        } // fin método buscaServicios()

        public void GenerarPaginaWML (HttpServletResponse resp) {
                resp.setContentType("text/vnd.wap.wml");
        PrintWriter out = null;
        try {
                out=resp.getWriter();
                }
        catch (IOException io) {
                        ;
                }

                // Se genera el contenido de la página WML
                out.println("<?xml version=\"1.0\"?>");
                out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"
```

```
                        \"http://www.wapforum.org/DTD/wml_1.1.xml\"> ");
            out.println("<wml>");
            out.println("<card id=\"resp\" title=\"Seleccione Hospital\">");
            out.println("<onevent type=\"oncenterforward\">");
            out.println("<refresh>");
            out.println("<setvar name=\"Servicio\" value=\""+servicio+"\" />");
            out.println("<setvar name=\"Hospital\" value=\"\" />");
            out.println("</refresh>");
            out.println("</onevent>");
            out.println("<p>");
            out.println("<select name=\"Hospital\">");

            Directorio dir=null;
            String hosp;

            for (int i=0; i<vectorDirectorio.size();i++) {
                  dir=(Directorio)vectorDirectorio.elementAt(i);
                  hosp=dir.getNom_Hosp();
                  //hosp = hosp.replace(' ','+');

                  out.println("<option onpick=\"$wml\"
                              value=\""+hosp+"\">"+hosp+"</option>");
            }

            out.println("</select></p></card>");
            out.println("<card id=\"wml\" title=\"Asignar\">");
            out.println("<p>");
            out.println("Hospital: $(Hospital)");
            out.println("<br />");
            out.println("Especialidad: $(Servicio)");
            out.println("<br />    </p>");
            out.println("<do type=\"accept\">");
            out.println("<go");

      out.println("href=\"http://148.204.211.194:8080/wap/servlet/AsignaRecurso?hosp=$(
                  Hospital)&amp;serv=$(Servicio)\"");
            out.println("/>");
            out.println("</do> </card> </wml>");
            out.flush();
            out.close();

      } // fin de GenerarPaginaWML ()

      public void enviaError(HttpServletResponse resp, String error) {
            resp.setContentType("text/vnd.wap.wml");
      PrintWriter out = null;
      try {
            out=resp.getWriter();
            }
      catch (IOException io) {
                  ;
            }

            // Se genera el contenido de la página WML
            out.println("<?xml version=\"1.0\"?>");
            out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"
                        \"http://www.wapforum.org/DTD/wml_1.1.xml\"> ");
            out.println("<wml>");
            out.println("<card id=\"resp\" title=\"Error\">");
            out.println("<p>");
            out.println(error);
            out.println("</p></card> </wml>");
            out.flush();
            out.close();

      } // fin de enviaError()

      public String getServletInfo() {
            return "Genera WML";
      } // fin del método getServletInfo()
}
```


AsignaRecurso.java

```
//AsignaRecurso.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

public class AsignaRecurso extends HttpServlet {

      Connection conn = null;
      Vector vectorDirectorio=null;
      private String hospital = null;
      private String servicio = null;
      String jdbcdriver = null;
      String jdbcurl = null;
      String tserv = null;
      String ting = null;
      String inserv = null;
      String iing = null;
      String nserv = null;
      String odisp = null;
      String uid = "";
      String pwd = "";
      HttpSession session;

      public void init(ServletConfig config) throws ServletException {
            super.init(config);
            String url=new String("jdbc:odbc:Contenedor");

            try {
                  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            } catch(ClassNotFoundException ex) {
                  System.out.println("Error al cargar el driver");
                  System.out.println(ex.getMessage());
            }

            try {
                  conn=DriverManager.getConnection(url,"","");
            } catch (SQLException sqlEx) {
                  System.out.println("Se ha producido un error al establecer la
                  conexión con: "+url);
                  System.out.println(sqlEx.getMessage());
            }
            System.out.println("Iniciando AsignaRecurso");
      } // fin del método init()

      public void destroy () {
            super.destroy();
            System.out.println("Cerrando conexion...");
            try {
                  conn.close();
            } catch(SQLException ex){
                  System.out.println("No se pudo cerrar la conexion");
                  System.out.println(ex.getMessage());
            }
      } // fin del método destroy()

      public void doPost (HttpServletRequest req, HttpServletResponse resp) throws
            ServletException, IOException {

      hospital = req.getParameter("hosp");
      servicio = req.getParameter("serv");

      session.setAttribute("hospital", hospital);
      session.setAttribute("servicio", servicio);

      if(hospital==null && servicio==null) {
            System.out.println("Se ha producido un error en la lectura de la
                              solicitud");
            return;
      }

      int resultado;
```

```
            enviaInformacion(req, resp);
else if(resultado == -3)
        System.out.println("No hay recursos disponibles");
        else
        System.out.println("Se ha producido un error en el acceso a la base de
                            datos");

} // fin del método doPost()

public void doGet (HttpServletRequest req, HttpServletResponse resp) throws
        ServletException, IOException {
sesion = req.getSession();
doPost(req,resp);
} // fin del método doGet()


public int obtieneInformacion(String hosp) {

        Statement stmt=null;
        ResultSet rs = null;
        String buscar;
        int numeroFilasActualizadas=0;


        try {
                stmt = conn.createStatement();

                buscar = "SELECT  DISTINCT JDBCDriver, JDBCURL, Tabla_serv,
                 Id_serv, Nombre_serv, Camas_disp, Tabla_ing, Id_ing "+
                                                        "FROM BaseDatos "+
                                                        "WHERE URL "+
                                                        "IN (SELECT URL "+
                                                        "FROM Hospital "+
                                                "WHERE Nom_Hosp='"+hosp+"') ";

                rs=stmt.executeQuery(buscar);

                while (rs.next()) {
                        Directorio temp = new Directorio();
                        jdbcdriver = rs.getString("JDBCDriver");
                        jdbcurl = rs.getString("JDBCURL");
                        tserv = rs.getString("Tabla_serv");
                        iserv = rs.getString("Id_serv");
                        nserv = rs.getString("Nombre_serv");
                        cdisp = rs.getString("Camas_disp");
                        ting = rs.getString("Tabla_ing");
                        iing = rs.getString("Id_ing");
                }

                if(jdbcdriver==null && jdbcurl==null && iserv==null && iserv==null
                    && nserv==null && cdisp==null && ting==null && ting==null)
                        return -3;
                return 0;

        } catch (SQLException sql) {
                System.out.println("Se produjo un error creando sentencia SQL");
                System.out.println(sql.getMessage());
                return -1;
        } finally {

                // Se cierra el Statement
                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e){
                                System.out.println("Error cerrando sentencia SQL");
                                System.out.println(e.getMessage());
                                return -2;
                        }
                }
        }
} // fin método obtieneInformacion()

public void enviaInformacion(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
```



```
        PrintWriter out = resp.getWriter();

        Conexion info = new Conexion();

        out.println(info.ejecuta(jdbcdriver, jdbcurl, uid, pwd, hospital,
                servicio, tserv, ting, nserv, cdisp, iing, iserv));

        } // fin del método enviaInformacion()
}
```

Conexión.java

```
/* Conexion.java */

import java.sql.*;
import java.util.*;

public class Conexion {

        Connection conn = null;
        Vector vectorDirectorio=null;
        String Id_Serv = null;
        int Camas;
        int identificador;

        public Conexion() {
                super();
        }

        public int obtieneID(String ID, String iserv) {

                Statement stmt = null;
                ResultSet rs = null;
                Directorio dir = null;

                try {
                        stmt=conn.createStatement();
                        rs=stmt.executeQuery(ID);
                        vectorDirectorio=new Vector();

                        while (rs.next()) {
                                Directorio temp = new Directorio();
                                temp.setID_Serv(rs.getString(iserv));
                                vectorDirectorio.addElement(temp);
                        }

                        if(vectorDirectorio.size()==0)
                                return -3;

                        for (int i=0; i<vectorDirectorio.size();i++) {
                                dir=(Directorio)vectorDirectorio.elementAt(i);
                                Id_Serv = new String(dir.getID_Serv());
                        }
                        return 0;

                } catch (SQLException sql) {
                        System.out.println("Se produjo un error al crear sentencia SQL");
                        System.out.println(sql.getMessage());
                        return -1;
                } finally {
                        if(stmt!=null) {
                                try {
                                        stmt.close();
                                } catch(SQLException e) {
                                System.out.println("Error al cerrar sentencia SQL");
                                        System.out.println(e.getMessage());
                                        return -2;
                                }
                        }
                }
        }// fin del método obtieneID()

        public int actualizaDisponibilidad(String Disponibilidad) {
```

```
        Statement stmt=null;
        ResultSet rs = null;
        String actualizar;
        int numeroFilasActualizadas=0;

        try {
                stmt = conn.createStatement();
                numeroFilasActualizadas = stmt.executeUpdate(Disponibilidad);
                if(numeroFilasActualizadas!=1) return -1;
        } catch (SQLException sql) {
                System.out.println("Se produjo un error creando sentencia SQL");
                System.out.println(sql.getMessage());
                return -2;
        } finally {
                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e){
                                System.out.println("Error cerrando sentencia SQL");
                                System.out.println(e.getMessage());
                                return -3;
                        }
                }
                return 0;
        }
} // fin método actualizaDisponibilidad()

public int identificadorIngreso(String ID_ing, String ing){
        Statement stmt = null;
        ResultSet rs = null;
        Directorio dir = null;

        try {
                stmt=conn.createStatement();
                rs=stmt.executeQuery(ID_ing);
                vectorDirectorio = new Vector();

                while (rs.next()) {
                        Directorio temp = new Directorio();
                        temp.setIngreso(rs.getInt(ing));
                        vectorDirectorio.addElement(temp);
                }

                if(vectorDirectorio.size()==0)
                        return -3;

                for (int i=0; i<vectorDirectorio.size();i++) {
                        dir=(Directorio)vectorDirectorio.elementAt(i);
                        identificador=dir.getIngreso();
                }

                identificador++;
                return 0;

        } catch (SQLException sql) {
                System.out.println("Se produjo un error al crear el Statement");
                System.out.println(sql.getMessage());
                return -1;
        } finally {

                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e) {
                                System.out.println("Error al cerrar el Statement");
                                System.out.println(e.getMessage());
                                return -2;
                        }
                }
        }
} // fin del método identificadorIngreso()

public int realizaIngreso(String Ingreso) {
        Statement stmt=null;
        int numeroFilasActualizadas=0;
```



```
        try {
                stmt = conn.createStatement();
                numeroFilasActualizadas = stmt.executeUpdate(Ingreso);
                if(numeroFilasActualizadas!=1) return -1;
        } catch (SQLException sql) {
                System.out.println("Se produjo un error creando Statement");
                System.out.println(sql.getMessage());
                return -2;
        } finally {
                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e){
                                System.out.println("Error cerrando Statement");
                                System.out.println(e.getMessage());
                                return -3;
                        }
                }
                return 0;
        }
} // fin método realizaIngreso()

public int recursosDisponibles(String Recurso, String cdisp) {

        Statement stmt=null;
        ResultSet rs = null;
        int numeroFilasActualizadas=0;

        try {
                stmt = conn.createStatement();
                rs=stmt.executeQuery(Recurso);

                while (rs.next()) {
                        Directorio temp = new Directorio();
                        Camas = rs.getInt(cdisp);
                }

                if(vectorDirectorio.size()==0)
                        return -3;

                return 0;

        } catch (SQLException sql) {
                System.out.println("Se produjo un error creando sentencia SQL");
                System.out.println(sql.getMessage());
                return -1;
        } finally {

                // Se cierra el Statement
                if(stmt!=null) {
                        try {
                                stmt.close();
                        } catch(SQLException e){
                                System.out.println("Error cerrando sentencia SQL");
                                System.out.println(e.getMessage());
                                return -2;
                        }
                }
        }
} // fin método recursosDisponibles()


String writeWML(String hospital) {
        StringBuffer strResultado = new StringBuffer("<?xml version=\"1.0\"
                                        encoding=\"ISO-8859-1\"?>\r\n");
        strResultado.append("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"
                        \"http://www.wapforum.org/DTD/wml_1.1.xml\">\r\n ");
        strResultado.append("<wml>\r\n");
        strResultado.append("<card id=\"asigna\" title=\"Identificador\">\r\n");
        strResultado.append("<onevent type=\"onenterforward\">\r\n");
        strResultado.append("<refresh>\r\n");
        strResultado.append("<setvar name=\"Hospital\" value=\""+hospital+"\"
                        />\r\n");
        strResultado.append("<setvar name=\"ID_Serv\" value=\""+Id_Serv+"\"
                        />\r\n");
        strResultado.append("<setvar name=\"camas\" value=\""+Camas+"\" />\r\n");
```

```
                strResults.append("</refresh>\r\n");
                strResults.append("</onevent>\r\n");
                strResults.append("<p>\r\n");
                strResults.append("No. asignado: " +identificador);
                strResults.append("\r\n</p>\r\n");
                strResults.append("<do type=\"accept\" label=\"Aceptar\">\r\n");
                strResults.append("<go ");

        strResults.append("href=\"http://148.204.211.154:8080/wap/ActualizaContenedor?Hos
pital=${Hospital}&amp;ID_Serv=${ID_Serv}&amp;camas=${camas}&amp;id=${id}\"/>\r\n");
                strResults.append("</do>\r\n </card>\r\n </wml>");
                return strResults.toString();
        }

        public String execute(String driver, String url, String uid, String pwd, String
hospital, String servicio, String nserv, String ting, String nserv, String odisp, String
iing, String iserv) {
                String output = new String();
                try {
                        Class.forName(driver);
                        conn = DriverManager.getConnection(url, uid, pwd);

                        String ID =  "SELECT " +iserv+ " FROM " +tserv+ " WHERE " +nserv+
                                     " = '"+servicio+"'" +
                                     " AND " +odisp+ " > 0";

                        String Disponibilidad = "UPDATE " +tserv+ " SET " +odisp+ " = "+odisp+" -
                                        1" +
                                      " WHERE " +iserv+ " = '"+servicio+"'" + " AND "
                                        +odisp+ " > 0";


                        String ID_ing = "SELECT " +iing+ " FROM " +ting+ " ORDER BY " +iing;

                        obtenerID(ID, iserv);
                        actualizaDisponibilidad(Disponibilidad);
                        identificadorIngreso(ID_ing, iing);

                        String Ingresa = "INSERT INTO " +ting+ "("+iing+ ", " +iserv+")" +
                                         " VALUES ('"+identificador+"',
                                          '"+id_Serv+"')";

                        String Recurso = "SELECT " +odisp+ " FROM " +tserv+ " WHERE "
                                        +nserv+ " = '"+servicio+"'";

                        realizaIngreso(Ingresa);
                        recursosDisponibles(Recurso, odisp);
                        output = writeWML(hospital);
                        conn.close();
                }
                catch(Exception e) {
                        System.out.println("Se ha producido un error al establecer "+"la
                                conexión con: "+url);
                }
                return output;
        }

}
```



```
/* ActualizaContenedor.java */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

public class ActualizaContenedor extends HttpServlet {

        Connection conn = null;
        Vector vectorDirectorio=null;
        String hosp = null;
        String id_serv = null;
        Integer camas_disp = null;
        String cam = null;
        Integer camas_obt = null;
        HttpSession session;

        public void init (ServletConfig config) throws ServletException {

                super.init(config);

                String url=new String("jdbc:odbc:Contenedor");

                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                } catch(ClassNotFoundException ex) {
                        System.out.println("Error al cargar el driver");
                        System.out.println(ex.getMessage());
                }

                try {
                        conn=DriverManager.getConnection(url,"","");
                } catch (SQLException sqlEx) {
                        System.out.println("Se ha producido un error al establecer "+"la
                                        conexión con: "+url);
                        System.out.println(sqlEx.getMessage());
                }
                System.out.println("Iniciando ActualizaServicios");
        } // fin del método init()

        public void destroy () {
                super.destroy();
                System.out.println("Cerrando conexion...");
                try {
                        conn.close();
                } catch(SQLException ex){
                        System.out.println("No se pudo cerrar la conexion");
                        System.out.println(ex.getMessage());
                }
        } // fin de destroy()

        public void doPost (HttpServletRequest req, HttpServletResponse resp)throws
                ServletException, IOException {

                hosp = req.getParameter("Hospital");
                id_serv = req.getParameter("ID_Serv");
                CAM = req.getParameter("camas");

                session.setAttribute("id", req.getParameter("id"));

                camas_disp = new Integer(cam);

                int resultado = existeServicio();

                if(resultado == 0 && camas_disp.compareTo(camas_obt) != 0) {
                        ActualizaServicio();
                        System.out.println ("Base De datos actualizada");
                        session.invalidate();
                        GenerarPaginaWML(resp);
                }
                else {
                        resp.sendError(500, "Se ha producido un error en el acceso " +"a
```

```
                la base de datos");
        }
    } // fin del método doPost()

    public void doGet (HttpServletRequest req, HttpServletResponse resp) throws
        ServletException, IOException {
    sesion = req.getSession();
    doPost(req,resp);
    } // fin del método doGet()

    public int existeServicio() {

        Statement stmt = null;
        ResultSet rs = null;
        Directorio dir=null;

        try {
            stmt=conn.createStatement();
            String buscar = "SELECT Canal"+
                                    " FROM Servicios "+
                                    "WHERE ((ID_Serv)='"+id_serv+"') "+
                                    "AND (URL " +
                                        "IN (SELECT URL "+
                                        "FROM Hospital "+
                                    "WHERE Non_Hosp='"+hosp+"')) ";

            rs=stmt.executeQuery(buscar);
            vectorDirectorio=new Vector();

            while (rs.next()) {
                Directorio temp = new Directorio();
                temp.setCanal(rs.getInt("Canal"));
                vectorDirectorio.addElement(temp);
            }

            if(vectorDirectorio.size()==0)
                return -3;

            for (int i=0; i<vectorDirectorio.size();i++) {
                dir=(Directorio)vectorDirectorio.elementAt(i);
                canal_obt = new Integer(dir.getCanal());
            }
            return 0;

        } catch (SQLException sql) {
            System.out.println("Se produjo un error al crear sentencia SQL");
            System.out.println(sql.getMessage());
            return -1;
        } finally {
            if(stmt!=null) {
                try {
                    stmt.close();
                } catch(SQLException e) {
                System.out.println("Error al cerrar sentencia SQL");
                    System.out.println(e.getMessage());
                    return -2;
                }
            }
        }
    }// fin del método existeServicio


    public int actualizaServicio() {

        Statement stmt=null;
        ResultSet rs = null;
        String actualizar;
        int numeroFilasActualizadas=0;

        try {
            stmt = conn.createStatement();
            actualizar = "UPDATE Servicios SET Canal='"+canal_disp+"' "+
                                "WHERE ((ID_Serv)='"+id_serv+"') "+
                                "AND (URL " +
                                    "IN (SELECT URL "+
                                    "FROM Hospital "+
                                                    "WHERE Non_Hosp='"+hosp+"')) ";

            numeroFilasActualizadas = stmt.executeUpdate(actualizar);
            if(numeroFilasActualizadas!=1) return -1;
        } catch (SQLException sql) {
            System.out.println("Se produjo un error creando sentencia SQL");
            System.out.println(sql.getMessage());
            return -2;
        } finally {
            if(stmt!=null) {
                try {
                    stmt.close();
                } catch(SQLException e){
                    System.out.println("Error cerrando sentencia SQL");
                    System.out.println(e.getMessage());
                    return -3;
                }
            }
            return 0;
        }
    } // fin método actualizaServicio()

    public void ConcretarPaginaWML (HttpServletResponse resp) {
        resp.setContentType("text/vnd.wap.wml");
    PrintWriter out = null;
    try {
        out=resp.getWriter();
        }
    catch (IOException ioe) {
            ;
        }

        // Se genera el contenido de la página WML
        out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
        out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"
                            \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
        out.println("<wml>");
        out.println("<card id=\"fin\" title=\"Gracias\">");
        out.println("<p> Por utilizar este servicio. </p>");
        out.println("</card> </wml>");
        out.flush();
        out.close();

    } // fin de ConcretarPaginaWML ()

}// fin de la clase ActualizaContenedor()
```

Glosario

AGR Administrador Global de Recursos. Mecanismo que integra para su administración recursos provenientes de bases de datos heterogéneas.

API *Application Program Interface*, interfaz para la programación de aplicaciones. Es una interfaz que determina la forma en que un programador accede al comportamiento y estado de clases y objetos cuando desarrolla una aplicación.

Card Carta. Son los controladores de una aplicación wml. Definen el aspecto de una página y los pasos que se han de dar cuando se navega por ella.

CR Contenedor de Recursos. Es un proveedor de recursos.

Deck Es un documento o página wml. Se compone de los mismos elementos básicos que se pueden encontrar en un documento HTML. Cada página contiene una o más cartas (card).

DOM *Document Object Model*, Modelo de Objeto Documento. Es una API que proporciona una estructura de árbol de objetos. Permite manipular la jerarquía de esos objetos. DOM es ideal para aplicaciones interactivas porque el modelo de objetos esta presente en memoria, donde se puede acceder y manipular por el usuario

DTD *Document Type Definition*, Definición de Tipo de Documento. Es el conjunto de normas XML para la representación de documentos de un determinado tipo. En ella se describe la composición de cada elemento que puede aparecer en un documento.

EM Entidad Móvil. Es un teléfono celular.

Gateway WAP Controla todos los datos que se envían desde un dispositivo móvil a un servidor HTTP y viceversa. Filtra la conversación para que al dispositivo únicamente le lleguen datos WAP y no HTTP.

GPRS *General Packet Access Service*. Servicio General de Acceso de Paquetes.

GSM *Global System for Mobile communication*, Sistema Global para comunicaciones Móviles.

HTML HyperText Markup Language, Lenguaje de Marcado de Hipertexto. Es un lenguaje muy sencillo que permite describir hipertexto, es decir, texto presentado de forma estructurada y agradable, con enlaces (hyperlinks) que conducen a otros documentos o fuentes de información relacionadas, y con inserciones multimedia (gráficos, sonido, etc.).

HTTP *Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto. Permite que los sistemas sean construidos independientemente de los datos transferidos, y es el más usado para navegar en Internet. La transferencia de hipertexto es simplemente la transferencia de archivos de hipertexto de una computadora a otra. El protocolo de transferencia de hipertexto es el conjunto de reglas utilizadas por las computadoras para transferir archivos de hipertexto, páginas web por Internet.

RDBMS *Relational Database Management System*, Sistema Manejador de Bases de Datos Relacionales. Es un programa que permite crear, actualizar y administrar bases de datos relacionales, a través de instrucciones SQL suministradas por el usuario o contenidas en un programa. Algunos de los más conocidos son Oracle, Microsoft SQL Server, IBM DB2, etc.

SAR Sistema de Asignación de Recursos.

SAX Es un API para XML, es una interfaz estándar para analizar eventos basados en XML, fue creada con la colaboración de miembros del XML-DEV y evaluada por OASIS. Proporciona un mecanismo orientado a eventos y de acceso serial para XML, realizando un procesamiento elemento por elemento. SAX lee y escribe XML a un repositorio de datos o a Internet.

Script Es una lista de comandos que se pueden ejecutar sin la interacción del usuario. Un lenguaje script es un lenguaje de programación simple con el cual se pueden escribir secuencias de comandos.

Servlet Son piezas de código escrito en JAVA que se ejecutan en un servidor web y le añaden funcionalidad del mismo modo que un applet le añade funcionalidad a un navegador. Están diseñados para soportar un modelo de petición/respuesta que es el más usado en el mundo web.

SGML *Standard Generalized Markup Language*, Lenguaje Estándar de Marcado Generalizado. Es un lenguaje para describir lenguajes de marcado, particularmente aquellos utilizados en el intercambio, administración y publicación de documentos electrónicos.

SQL *Structured Query Language*, Lenguaje Estructurado de Consulta. Se utiliza para pedir a un sistema administrador de bases de datos que realice operaciones de consulta, modificación y control sobre las tablas de una base de datos.

TCP/IP *Transmission Control Protocol/Internet Protocol*, Protocolo de Control de Transmisión / Protocolo de Internet. Es un conjunto de protocolos que generalmente se le hace referencia como un sólo protocolo que se usa en Internet. Su diseño posee una arquitectura en capas. Dichas capas permiten a los diseñadores dividir en módulos las tareas y servicios que se llevaran a cabo.

UML *Unified Model Language*, Lenguaje Unificado de Modelado. Es un lenguaje de modelado que permite hacer análisis y diseño de sistemas, valiéndose de diagramas que muestran la estructura y comportamiento de los objetos que los componen.

W3C *WWW Consortium*, Consorcio WWW. Organización apadrinada por el MIT y el CERN, entre otros, cuya misión es el establecimiento de los estándares relacionados con WWW. Fue promovida por el creador del WWW, Tim Berners-Lee.

WAP *Wireles Application Protocol*, Protocolo de Aplicación Inalámbrica. Es el estándar del cómputo móvil que dirige un grupo de distribuidores llamado WAP Forum. Es un mecanismo para el transporte de datos.

WML *Wireles Markup Language,* Lenguaje de Marcado Inalámbrico. Lenguaje de programación basado en etiquetas que se usa para describir la estructura de los documentos que se distribuyen a través de dispositivos móviles.

WML script Lenguaje de procedimiento orientado a objetos para la creación de script. Complementa y mejora las capacidades relativamente limitadas de WML.

XML *eXtensible Markup Language,* Lenguaje de Marcado Extensible. Lenguaje desarrollado por el W3C para permitir la descripción de información contenida en el WWW a través de estándares y formatos comunes, de manera que tanto los usuarios de Internet como programas específicos (agentes) puedan buscar, comparar y compartir información en la red. El formato de XML es muy parecido al del HTML aunque no es una extensión ni un componente de éste.