UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"Manipulación de información entre diversas tecnologías utilizando XML"

T E S I S

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A N:

GÓMEZ NAVARRO MANUEL

HERNÁNDEZ MATÍAS CARLOS ALBERTO

DIRECTOR DE TESIS: ING. JUAN JOSÉ CARREÓN GRANADOS

CIUDAD UNIVERSITARIA **MÉXICO D.F. 2007**

Dedicatoria

En memoria de mi padre Guillermo Gómez Balanzario,

A mi madre Aurora Navarro y a mis hermanos.

Manuel

A mis padres: Carlos y Audelia, por comprenderme y darme todo su apoyo,

A mis hermanos: Ricardo, Francisco y Fabiola, por estar siempre a mi lado,

Y a todos aquellos, que de una forma u otra me han ayudado

aportándome sus conocimientos.

Carlos Alberto

Agradecimientos

Agradezco a la Dirección General de Servicios Escolares (DGSCA).

Manuel

Agradezco el esfuerzo de mis padres, el apoyo de mis hermanos y familiares, la gentileza y enseñanza de mis profesores, la oportunidad y el espacio proporcionado por nuestra universidad, la UNAM.

Carlos Alberto

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Descripción del problema

Actualmente existen una variedad de tecnologías que permiten desarrollar sistemas de información de gran complejidad que facilitan notablemente el trabajo cotidiano de cualquier oficina o de alguna organización. No obstante, las necesidades se incrementan y una de ellas es la de compartir información entre diversos sistemas de información de una o distintas organizaciones. Si lo anterior sucediera con sistemas que están desarrollados con la misma tecnología, evidentemente no habría tantos problemas como en sistemas que no son del todo compatibles, por lo que se ha buscado la manera de hacer que estas organizaciones puedan interactuar siendo parte de una red de computadoras. Otro problema que se presenta, es la persistencia de datos, ya que de alguna manera la información que se maneja dentro de estos sistemas, tiene que ser almacenada en algún momento. Sabemos que existen una diversidad de sistemas de administración de bases de datos, aunque siempre escogemos el más adecuado a nuestras necesidades, casi nunca nos prevenimos de una posible migración hacia otro sistema. Es por eso que surge un serio problema en cuanto al mantenimiento del software desarrollado.

Partiendo de un ejemplo práctico: Una organización del sector privado ha detectado un problema común en los habitantes del país, dicho problema corresponde al manejo de los recursos monetarios tanto de los usuarios consumidores como de los comerciantes. Por ello, planea desarrollar un proyecto para realizar la afiliación de los comerciantes con algún determinado banco, de esta manera podrán contar con un crédito bancario que les permita mantener seguro los ingresos de su negocio mediante el manejo de terminales de venta electrónicos, y de la misma forma proporcionaran seguridad de las transacciones vía electrónica para los consumidores quienes harán uso de una tarjeta de monedero electrónico.

Dicha organización controlará la información de las distintas instituciones bancarias y de los comerciantes interesados. Para llevar a cabo el proyecto, se requiere del desarrollo de un sistema que controle toda la información involucrada para el proceso de Afiliación. Además, el sistema deberá proporcionar la visualización y administración de la información registrada. La información será proporcionada por personal de la misma organización, o por otras organizaciones partícipes, con las cuales se mantendrá una comunicación para la realización conjunta de la afiliación. Cada una de las organizaciones participantes, tendrá su propio sistema para la recepción, procesamiento, almacenamiento y transmisión de la información de cada una de las etapas del proceso de afiliación. Un punto importante del problema, es que el sistema de cada organización no necesariamente está desarrollado con la misma tecnología ni opera sobre la misma plataforma, por lo que se tendrá que buscar una manera de establecer la comunicación entre los mismos.

1.2 Objetivo

El principal objetivo es demostrar la importancia que tiene el Lenguaje de Marcado Extensible en el manejo de datos, además de solucionar el problema de compatibilidad entre sistemas informáticos desarrollados con distinta tecnología de software.

1.3 Método

Para el desarrollo del sistema informático nos basaremos en el lenguaje de programación Java y tecnologías J2EE. A continuación se mencionará el método a seguir para obtener una solución al problema planteado:

i) Se diseñará un estándar basado en XML que permita representar toda la información requerida por la organización para completar la afiliación, de tal manera que los documentos XML obtenidos con dicho estándar pueda ser entendido por todos los sistemas que interactúan en el proceso de afiliación.

ii) Se definirán las distintas etapas del proceso de afiliación y, a partir del estándar, se obtendrá el documento XML completo además de otros que contendrán sólo

parte de la información, estos últimos serán clasificados de acuerdo a cada etapa.

iii) El sistema proporcionará a las demás organizaciones involucradas, el documento XML con la información requerida en cada una de las etapas de la afiliación, por lo que se definirán los servicios Web que se utilizarán para establecer la comunicación con los otros sistemas.

iv) Al establecer la comunicación con alguno de los otros sistemas, nuestro sistema identificará la etapa en la que se encuentra la afiliación, y manipulará el documento XML proporcionado por el otro sistema, agregando o eliminando datos, y poder proporcionarlo al mismo sistema o a otro.

v) Se realizará el diseño de la base de datos local para almacenar los datos necesarios para el control de las distintas afiliaciones. Se definirá el mapeo de cada uno de los objetos Java, que contendrán los datos obtenidos del documento XML, con respecto a las tablas de la base de datos.

vi) Se desarrollara la interfaz para que los distintos usuarios puedan acceder a la información registrada, de forma limitada o detallada de acuerdo a su perfil.

Capítulo 2

Información

2.1 ¿Qué es dato?

Un dato es una representación simbólica (numérica, alfabética, etc.), atributo o característica de una entidad. El dato no tiene valor semántico, es decir, sentido en sí mismo. Al ser utilizado convenientemente junto con otros datos puede ser muy común en el ámbito informático.

2.2 ¿Qué es información?

Información es un conjunto de datos organizados o procesados con cierto significado sobre un ente. Al juntar y organizar diferente información de una o varias fuentes se construye el conocimiento, que permitirá resolver problemas o ayudar a la toma de decisiones.

De acuerdo con otro punto de vista, información es un fenómeno que proporciona significado a las cosas, indicando, a través de un conjunto de códigos y/o datos, los modelos del pensamiento humano.

En programación, un dato es la expresión que describe la característica de una entidad. Un dato por si mismo no constituye información, es el procesamiento de los datos lo que nos proporciona información. Por lo que, los datos son sujetos a una representación y proceso, son utilizados como diversos métodos para comprimir la información a fin de permitir una transmisión o almacenamientos más eficaces.

2.3 Representación de la información

La representación de la información es la ciencia, filosofía o arte que estudia las distintas formas en que se puede comunicar y acceder a la información.

Para representar la información se busca realizarlo mediante un lenguaje (gráfico o escrito) que sea entendible para el ser humano, para ello es necesario de cierta abstracción, es decir, de un proceso mental a través del cual se pueda extraer los rasgos esenciales de algo.

2.3.1 Representación en una computadora

En el mundo de la computación, la información manejada por las computadoras electrónicas es representada a través de un código binario (0 y 1); siendo la mínima unidad de información el **BIT** (contracción de *binary digit*).

El sistema binario permite representar cualquier número natural y, con mayor dificultad, números enteros, números reales, caracteres y valores lógicos. El principal problema radica en la utilización de un número limitado de bits para llevar a cabo estas representaciones. Por ello, los números deben mantenerse dentro de un rango y una precisión limitados y sólo es posible representar un número finito de caracteres.

2.3.1.1 Representación de texto

Para representar texto, es necesario establecer un código que asocie a cada carácter un valor binario, de tal manera que tenga una representación universal para que distintos individuos puedan hacer uso de ellos y poder intercambiar información.

Cada país tiene sus símbolos estructurados, conocido como página de código de caracteres, estos símbolos difieren por el idioma y su representación es distinta dependiendo del software o plataforma que se emplea.

2.3.1.1.1 Métodos de representación de caracteres

Existen métodos de representación de caracteres, representados internamente como enteros sin signo, los más conocidos son el **ASCII** (American Standard Code for Information

Interchange) y el **EBCDIC** (Extended Binary Code Decimal Interchange Code). Ambos códigos son representados con un byte (8 bits), lo que limita la representación a 256 símbolos, los cuales pueden ser suficientes para un tipo de alfabeto, por ejemplo el latino, pero no para otros donde se tiene lenguajes ideográficos con decenas de miles de caracteres.

Toda computadora (especialmente los servidores) necesita ser compatible con muchos sistemas de codificación distintos; sin embargo, cada vez que los datos se traspasan entre distintos sistemas de codificación o plataformas, dichos datos corren el riesgo de sufrir daños.

Para poder hacer global la representación de caracteres, se inventó un estándar de codificación de caracteres llamado **UNICODE**.

UNICODE es un estándar que proporciona un número único (e inequívoco) para cada carácter independientemente de la plataforma, el software y el idioma. Es un estándar que se está imponiendo a pasos gigantescos, es un requisito para las tecnologías modernas tales como XML, Java, JavaScript, entre otros; pero presenta un inconveniente, su problema es el tamaño. Por defecto, UNICODE es un sistema de codificación de 16 bits, con el objetivo de representar aproximadamente 65,000 caracteres, lo que significa que a menudo se desperdicia espacio porque la mayoría de texto es representada con caracteres de 8 bits.

En la actualidad, UNICODE soporta tres formatos de representación con un repertorio de caracteres comunes pero capaces de representar millones de caracteres. Los tres formatos permiten transmitir el texto empleando 8, 16 o 32 bits por código:

1. UTF-8: permite transformar todos los códigos UNICODE a una secuencia de bytes de longitud variable. Los caracteres que forman parte de ASCII tienen en UNICODE asignado los mismos valores.

2. UTF-16: permite que todos los caracteres comúnmente más utilizados se codifiquen en un sólo código empleando el resto parejas.
3. UTF-32: permite codificar directamente todos los caracteres UNICODE.

2.3.1.2 Estructuras de datos

Una ***estructura de datos*** es cualquier colección o grupo de datos organizados de una cierta forma. La estructura dependerá de la implementación que se vaya a dar. Si la estructura tiene como finalidad representar una colección de datos para su presentación o transmisión, sólo cumplirán con un formato de organización; pero si además, la estructura será sujeta a una manipulación de su contenido, los datos deberán estar organizados de tal forma que tengan asociados un conjunto de operaciones, los cuales, generalmente, son implementados en lenguajes de programación.

Por ejemplo, cualquier lenguaje de programación de alto nivel provee típicamente de tipos de datos estructurados o estructuras de datos predefinidas, como los arreglos o los registros. Un *arreglo* es un conjunto de datos, todos del mismo tipo, con una organización lineal y con métodos claros de acceso a través de subíndices. Las operaciones tradicionales sobre los arreglos incluyen la comparación, la asignación, la escritura, etc. En un nivel más bajo, podría verse a los números enteros como estructuras de datos: se componen de un grupo de dígitos y tienen asociadas operaciones como sumar, restar, multiplicar, entre otras.

2.3.1.2.1 Abstracción de datos

En el desarrollo de sistemas de cómputo, para modelar una realidad requiere necesariamente de hacer abstracciones.

La ***abstracción de datos*** es una metodología o técnica que permite diseñar estructuras de datos. Consiste, básicamente, en representar bajo ciertos lineamientos de formato las características esenciales de una estructura de datos.

La técnica de abstracción de datos establece que al diseñar una estructura de datos, ésta pasa a ser un Tipo de Dato Abstracto (TDA), que podrá implementarse en cualquier lenguaje. Por lo tanto, al usar la metodología de abstracción de datos se diseñaran TDA, siguiendo los lineamientos para hacer una especificación lógica o abstracta.

La especificación lógica de un TDA es un documento en el que se plasma la abstracción realizada al diseñar una estructura de datos. Dicho documento pasará a ser el mapa mediante el cual se construirá la estructura de datos y en el que se definen claramente las reglas en las que podrá usarse.

El documento de la especificación lógica de un TDA consiste de los siguientes cuatro puntos:

1. *Elementos que conformarán la estructura de datos*. Se describe el tipo de los datos individuales que guardará la estructura. Por ejemplo, números enteros, caracteres, fechas, registros con datos de un empleado, etc.
2. *Tipo de organización en el que se guardarán los elementos*. Existen solamente cuatro tipos de organización para los datos en la estructura, la cual deberá tener alguna de las siguientes organizaciones:
    - *Lineal*. Si hay una relación de uno a uno entre los elementos.
    - *Jerárquica*. Si hay una relación de uno a muchos entre los elementos.
    - *Red*. Si hay una relación de muchos a muchos entre los elementos.
    - *Sin relación*. Si no hay relación entre los elementos.
3. *Dominio de la estructura*. Se describe la capacidad de la estructura en cuanto al rango posible de datos por guardar (es opcional).
4. *Descripción de las operaciones de la estructura*. La operación debe describirse con un nombre, explicación de su utilidad, datos de entrada de la operación, datos que genera como salida.

La abstracción de datos es uno de los principios que fundamentan la programación orientada a objetos, y se describirá en el capítulo 4.

2.4 Manejo de la información

El manejo de la información comprende distintas acciones como por ejemplo: organizar, agregar, eliminar, modificar, recortar, copiar, pegar, formatear, sintetizar, representar, verificar, procesar, compartir, etc. Con el uso de la computadora esto se puede realizar de una forma más sencilla y rápida, contando con el software necesario para la manipulación de la información.

2.4.1 Compartir información

El proceso de compartir información trae consigo la obtención, almacenamiento y transmisión de la misma, efectuar esto en una computadora puede realizarse de distintas formas, como las descritas a continuación.

Para almacenar información en la computadora ya sea como una estructura de datos contenida en una base de datos o en un archivo de texto, como un archivo de audio o de video (creado de forma lógica a través de alguna técnica de representación), o como una imagen, puede realizarse en una memoria volátil o no volátil, es decir, en una memoria donde su contenido puede o no borrarse si se interrumpe el flujo de la corriente que la alimenta. Por ejemplo, en la memoria secundaria o disco duro (HD), la memoria principal (RAM), memoria Flash, memoria SD, u algún otro.

Para compartir información, puede realizarse desde su almacenamiento en algún dispositivo de entrada-salida como: disquete, CD, DVD, memoria Flash, memoria SD o un disco duro. Siempre y cuando, las computadoras cuenten con lectores y/o puertos, correspondientes para cada dispositivo, para la transmisión de los datos.

Actualmente, ha aumentado la manera de compartir información, de forma global, por medio de la red de computadoras más grande a nivel mundial, conocida como *Internet*. A través de este medio, la información está expuesta a cualquier usuario, independientemente de su situación geográfica e idioma, que tenga interés sobre la materia, siempre y cuando se cuente con la autorización para acceder a esa información.

2.4.2 Presentación de información por Internet

La forma más habitual de presentar la información por Internet es de forma textual a partir del formateo del texto (fuentes, tamaños, estilos, etc.). Generalmente, el contenido y la presentación del mismo quedan vinculados en un mismo documento, este documento es conocido como página Web.

Una página Web es creada a través de algún lenguaje de programación. Actualmente, la presentación de la información se realiza con un lenguaje de marcas/etiquetas. Dichas marcas delimitan porciones de texto para modificar su presentación.

Al principio, surgió el lenguaje *GML (Generalized Markup Language)*, y su sucesor *SGML (Standard Generalized Markup Language)*. Estos fueron desarrollados entre 1969 y 1986 por Charles Goldfard y otros desarrolladores en IBM, como una forma de agregar información a documentos médicos para que las computadoras pudieran procesarlos.

SGML fue una gran invención pero tenía un problema: su tamaño. A mediados de los 80s, la especificación completa de SGML fue mayor de 500 páginas, así que desarrollar software que siguiera todas esas reglas fue dificultoso.

En 1989, cuando Tim Berners-Lee necesito una forma para describir títulos, texto subrayado y referencias cruzadas, él dio el nombre de World Wide Web y lo dibujó inspirado de SGML. Su formato, llamado *HTML (HyperText Markup Language)*, tiene sólo un pequeño conjunto

de etiquetas. Algunos de estos fueron semánticos mientras que otros tenían un rol puramente visual.

El lenguaje HTML en sus primeras versiones no permitía separar el contenido de la presentación; aún hoy no es totalmente posible.

```
<html>
    <head>
        <title> Titulo del documento </title>
    </head>
    <body bgcolor="white">
        <p>
            Un párrafo con texto en <b> negrita </b>
        </p>
    </body>
</html>
```

Listado 2.1 Pagina HTML básica

Casi tan temprano como HTML apareció, para solucionar algunos de los problemas que éste tenía, algunos programadores comenzaron a agregar sus propias extensiones. Esto, pronto, dio como resultado el XML, del cual abordaremos con mas detalle en el capitulo 3.

Capítulo 3

Lenguaje de Marcado Extensible

3.1 ¿Qué es XML?

XML (eXtensible Markup Language o Lenguaje de Marcado Extensible) viene a ser una versión reducida de SGML y su primera versión fue aprobada en 1998.

XML no es un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Una de las necesidades primordiales es el manejo de un estándar para la representación de datos de forma estructurada; datos cuyo significado puede ser variable.

XML es la espina dorsal de muchas tecnologías implicadas en el manejo de la información. Además de ser aplicado en Internet como un estándar para la formación de estructuras de contenidos, sirve también como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y con otras herramientas para la manipulación de la información. Así mismo, se complementa con otras tecnologías que la hacen aún mucho más potente y con más posibilidades, lo que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

3.1.1 Antecedentes históricos

XML se convirtió en un lenguaje estándar con la publicación del documento Extensible Markup Language 1.0 Recomendación W3C 10 de febrero de 1998.

Posteriormente en octubre del año 2000, apareció una segunda edición con algunas correcciones al documento antes citado, pero debemos tener en cuenta que no son una nueva versión de XML ya que las recomendaciones siguen siendo las mismas.

3.1.2 XML como un tipo de objeto de datos

Las recomendaciones XML 1.0 del W3C definen los documentos XML como un tipo de objetos de datos que están formados físicamente por unidades de almacenamiento, denominadas *entidades*, cuyo contenido pueden ser datos analizados y no analizados.

- *Los datos analizados* (PCDATA, Parsed Character Data), están compuestos por datos de caracteres y marcas. Las marcas son símbolos especiales utilizadas por el lenguaje XML para indicar que el texto que viene a continuación debe de ser procesado, ya que se trata de una etiqueta, marcas < y >, o una referencia, marca &. Las marcas permiten establecer la estructura lógica y de almacenamiento del documento.
- *Los datos no analizados* o datos de caracteres (CDATA, Character Data) representarían datos de contenido textual que no han de ser analizados por el procesador XML.

XML es, por lo tanto, un lenguaje diseñado para trabajar con datos y estructuras. Otro punto interesante es que las aplicaciones destinadas a trabajar con documentos XML han de incorporar un módulo de software especial denominado *Procesador XML*, que permitirá leer el documento XML y acceder a su estructura y contenido.

3.1.3 Tipos de documentos XML

Realizar la definición de tipos de documentos (DTD, Document Type Definition) es una parte opcional de un documento XML. El propósito principal de la DTD es restringir el tipo de información presente en el documento. Sin embargo, DTD no restringe en realidad los tipos en el sentido de tipos básicos como entero o cadena. En su lugar solamente restringe el aspecto de subelementos (elementos hijos) y atributos en un elemento (elemento padre).

DTD es principalmente una lista de reglas que indican el patrón de subelementos que aparecen en un elemento. En seguida se muestra un fragmento de la DTD del sistema SCA (Sistema de Control de Afiliaciones).

```
<!DOCTYPE ListOfAccount [
    <!ELEMENT ListOfAccount ((Account)+)>
    <!ELEMENT Account (
        XMLName, FileID, TipoNegocio, NombreCorporativo,
        …
        ListOfMerchantPhoneNumbers )>

        <!ELEMENT XMLName  (#PCDATA)>
        <!ELEMENT FileID  (#PCDATA)>
        …
        <!ELEMENT ListOfMerchantPhoneNumbers (
MerchantPhoneNumbers  )>
        …
        <!ELEMENT  MerchantPhoneNumbers( TipoTelefono,
    DescripcionTelefono, Telefono )>
        …
]>
```

Listado 3.1 Frangmento DTD de una lista de Account

En esta DTD se puede observar que el elemento “Account” contiene elementos tales como “XMLVersion”, etc., los cuales tienen que ser declarados como otro ELEMENT debido a que serán elementos hijos de “Account”, y si estos elementos aún contienen elementos hijos, ellos también tendrán que ser declarados dentro de otro ELEMENT; tal es el caso de

“ListOfMerchantPhoneNumbers” el cual contiene un elemento hijo llamado “MerchantPhoneNumbers”.

EL operador “+” especifica “una o mas”, para este caso se puede observar que el elemento “ListOfAccount” podrá tener *uno o más* elementos “Account”. Algunos operadores que se pueden utilizar en la definición de una DTD son:

- Asterisco ' * ' el cual significa “cero o más”.
- ' | ' especifica “ o ”
- ' ? ' elemento opcional “cero o uno”

La palabra clave “*#PCDATA*” indica dato de texto; su nombre deriva históricamente de “parsed character data”. En la DTD anterior se observa que el elemento “FileID” es del tipo #PCDATA, lo cual indica que será un dato de texto, como por ejemplo <FileID>23AC42B321</FileID>. Otros tipos de especiales de declaraciones son “*empty*” que indica que el elemento no tiene ningún contenido y “*any*” que indica que no hay restricción sobre los subelementos del elemento; es decir, cualquier elemento puede ser subelemento del elemento, incluso los no mencionados en la DTD.

Un punto interesante, es que los elementos pueden tener atributos, como por ejemplo el elemento “ListOfAccount” podría tener un atributo llamado “idLista”, así la declaración sería la siguiente:

```
<!DOCTYPE ListOfAccount [
        <!ELEMENT ListOfAccount ((Account)+)>
        <!ATLIST  ListOfAccount idLista ID # REQUIRED>
        …
]>
```

Listado 3.2 DTD manejando atributo idLista

Los atributos pueden especificarse como alguno de los siguientes tipos:

- El tipo CDATA simplemente dice que el atributo contiene datos de caracteres.
- Un atributo de tipo ID proporciona un identificador único para el elemento; un valor de un atributo ID de un elemento no debe aparecer en ningún otro elemento del mismo documento.
- Un atributo del tipo IDREF es una referencia a un elemento, este debe contener un valor que aparezca en el atributo ID de algún elemento en el documento.
- El tipo IDREFS permite una lista de referencias separadas por espacios.

En el fragmento de la DTD anterior podemos percatarnos que el elemento “ListOfAcount” contiene un atributo, el cual servirá como identificador único dentro de un documento, un ejemplo de esto es el siguiente:

```
<ListOfAccount idLista=”123”>
    <Account>
    …
    </Account>
</ListOfAccount>
```

Las definiciones de tipos de documentos están fuertemente relacionadas con la herencia del formato del documento XML, lo cual significa que la DTD podrá ser utilizada solamente para validar la estructura del documento, quedando fuera las validaciones sofisticadas como el de condicionar a un elemento a que este sea un entero.

3.2 Bases de datos relacionales y XML

La conversión de datos XML a una forma relacional es normalmente sencilla si los datos se han generado desde un principio siguiendo un esquema relacional y el XML solamente se usó como un formato de intercambio de datos, sin embargo existen muchas aplicaciones

que no siguen lo descrito, debido a que su esquema relacional puede no ser tan sencillo. Para manejar esta situación, existen algunas alternativas que nos pueden ayudar, como por ejemplo las siguientes:

- Almacenamiento con cadenas
- Representación con árboles
- Publicación y fragmentación de datos XML
- Almacenamiento nativo en bases de datos relacionales

3.2.1 Almacenamiento con cadenas

Algunos Sistemas de administración de bases de datos, permiten almacenar los documentos XML como cadenas en tuplas de una base de datos relacional. Los documentos XML cuyo elemento de nivel superior tenga muchos hijos se pueden tratar almacenando cada elemento hijo como una cadena en una tupla separada de la base de datos. Aunque esta representación es fácil de usar, la principal desventaja que tenemos es que el sistema de base de datos no conoce el esquema de los elementos almacenados, por lo que será imposible consultar los datos directamente.

3.2.2 Representación con árboles

Un documento XML se puede modelar como un árbol, de tal manera que si a un nodo padre se le agrega un identificador único, y a los nodos hijos se le agregan tanto su propio identificador como el del padre, estos podrían ser manipulados con facilidad, debido a que toda la información contenido en el documento XML se encuentra representada de una forma relacional, esta es una gran ventaja, debido a que se puede acceder directamente a dicha información. El motivo que frecuentemente impide utilizar ésta alternativa, es que cada elemento del XML tiene un gran número de componentes y peor aún si por alguna razón se quiere saber la posición de cada uno de los ellos, si fuese este el caso, se necesitaría un atributo más que permitiera almacenar la ubicación del nodo, en otras palabras, para volver a ensamblar los subelementos en un solo elemento, se necesitan un gran número de combinaciones.

3.2.3 Publicación y fragmentación de datos XML

Cuando se usa XML para intercambiar datos entre aplicaciones de negocio, los datos se originan muy frecuentemente en bases de datos relacionales.

Los datos en bases de datos relacionales deben ser *publicados*, es decir, convertidos a formato XML para su exportación a otras aplicaciones. Los datos de entrada deben ser *fragmentados*, es decir, convertidos de XML a un formato normalizado de relación y almacenado en una base de datos relacional. Aunque el código de la aplicación pueda ejecutar las operaciones de publicación y fragmentación, las operaciones son tan comunes que la conversión se debería realizar de manera automática, sin escribir ningún código en la aplicación, siempre que sea posible.

Una base de datos con capacidades XML permite una correspondencia automática de su modelo relacional interno con XML. Esta correspondencia puede ser sencilla o compleja. Una correspondencia sencilla podría asignar un elemento XML a cada fila de una tabla y hacer de cada columna un subelemento del elemento XML.

3.2.4 Almacenamiento nativo en las bases de datos relacionales

Recientemente, las bases de datos han comenzado a dar soporte al almacenamiento nativo de XML. Estos sistemas almacenan datos XML como cadenas o representaciones binarias más eficientes, sin convertir los datos a la forma relacional. Se introduce el nuevo tipo de datos "*xml*" para representar datos XML, aunque los tipos CLOB y BLOB pueden proporcionar el mecanismo subyacente de almacenamiento. Se incluyen lenguajes de consultas tales como *XPath* y *XQuery* para las consultas de datos XML. Se puede utilizar una relación con un atributo de tipo "xml" para almacenar una colección de documentos XML. Estos sistemas que proporcionan soporte nativo para datos XML, también permiten que las consultas XQuery se incorporen dentro de las consultas SQL. Cada consulta XQuery se puede ejecutar en un solo documento XML y se puede incorporar dentro de una consulta SQL para permitir que se ejecute en cada colección de documentos, con cada documento almacenado en una tupla diferente.

3.3 Serialización de XML

A partir de un lenguaje de programación orientada a objetos, se puede hacer que la estructura de un documento XML pueda ser representada a partir de un objeto, logrando de esta manera almacenar los datos en las propiedades del objeto y así poder realizar una manipulación de los mismos. La manipulación más común para los desarrolladores con POO es: leer, modificar, y almacenar sobre algún medio, generalmente base de datos o un archivo.

La ***serialización*** de XML es el proceso de convertir un objeto con propiedades públicas a un formato serial (como XML) para poderlo transportar o almacenar. Las propiedades de un objeto son contenidas en archivos XML (.xml) los cuales pueden ser almacenados en la memoria de la aplicación, en una base de datos o en algún dispositivo de almacenamiento. La ***deserialización*** es el proceso inverso, recrea el objeto en su estado original de la salida XML para poder ser manipulado por la aplicación. Se puede pensar que la serialización es una forma de guardar el estado de un objeto dentro de un flujo o buffer.

Pero, para el programador, dependiendo de qué tecnología se utilice, puede presentarse problemas para acceder a los archivos XML, a pesar de que XML es texto plano. El gasto necesario para "*parsear*" (analizar) el XML es generalmente un precio tan alto a pagar por aplicaciones más pequeñas. Una tecnología que facilita el parseo de los archivos XML a un precio relativamente bajo es *XStream*, el cual es una simple librería para serializar objetos a XML y viceversa. En el capitulo 5 se hablara un poco más de ello.

Capítulo 4

Bases de datos y el Lenguaje JAVA

4.1 Introducción

El primer obstáculo al que se enfrentan los programadores que usan el modelo relacional de datos es el limitado sistema de tipos de datos soportados por este modelo. De esta manera, el acceso a los datos desde los programas escritos en lenguajes de programación orientada a objetos se hace complicado. Tener que expresar el acceso a las bases de datos mediante un lenguaje (SQL) que es diferente del lenguaje de programación hace más difícil el trabajo del programador. Es deseable, para muchas aplicaciones, contar con extensiones del lenguaje de programación que permita el acceso directo a los datos de la bases de datos, sin tener que pasar por un lenguaje intermedio como SQL.

4.2 Programación orientada a objetos

El término de Programación Orientada a Objetos (POO) indica más una forma de diseño y una metodología de desarrollo de software que un lenguaje de programación. Básicamente, permite a los desarrolladores escribir software de forma que esté organizado en la misma forma que el programa que trata de modelar, buscando expresar el problema en sentido del manejo de sus características y acciones.

La POO aporta un nuevo enfoque, convirtiendo la estructura de datos en el centro sobre el que las operaciones actúan. De esta forma, cualquier modificación de la estructura de datos tiene efecto inmediato sobre las acciones a realizar en ella, siendo esta una de las diferencias radicales respecto a la programación estructurada.

La filosofía de la POO es programar encapsulando datos y código para formar objetos, que interactúan para obtener los resultados esperados. Asimismo, la POO posee los mecanismos de herencia y polimorfismo como característica que dan mayor poder y flexibilidad a la programación.

En la POO, un objeto es un paquete que contiene datos y código en forma de subrutinas que operan sobre los datos del propio objeto. Dicho de otra forma, un objeto es una entidad que tiene unos atributos particulares, las *propiedades*, y acciones o formas de operar sobre ellos, los *métodos*. Tanto las propiedades como los métodos constituyen los *miembros* del objeto. Los objetos se comunican enviándose *mensajes*, llamadas a métodos, y así generan los resultados correspondientes.

4.2.1 Características de la POO

La **abstracción** permite no preocuparse de los detalles no esenciales, sino generalizar y centrarse en los aspectos que permitan tener una visión global del problema.

El **encapsulamiento** permite ver un objeto como una caja negra en la que se ha introducido de alguna manera toda la información relacionada con dicho objeto. Esto permite manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

La abstracción y el encapsulamiento están representadas por la clase. Una clase es un tipo de objeto definido por el usuario. La **clase** es una abstracción porque en ellas se definen las propiedades de un determinado conjunto de objetos con características comunes, y es una encapsulación porque constituye una caja negra que encierra los datos que almacena cada objeto como los métodos que permiten manipularlos.

La **herencia** es el proceso mediante el cual un objeto se define adquiriendo las propiedades de otro, hereda las propiedades y métodos de un objeto superior. La herencia permite el acceso automático a la información contenida en otras clases. De esta forma, la reutilización del código está garantizada. Con la herencia todas las clases están clasificadas en una jerarquía estricta.

El **polimorfismo** es un mecanismo mediante el cual se puede lograr que una misma operación se pueda realizar en diferentes formas, según los objetos sobre los que se aplica.

4.2.2 Estructura de datos en la POO

Hay una relación muy estrecha entre la abstracción de datos y la programación orientada a objetos, por lo que un objeto es visto como una TDA.

Puesto que la POO se apoya en la idea de la abstracción de datos, redunda en un mejor desarrollo de software. La técnica obliga a diseñar modularmente y, como consecuencia, se tiene una implementación clara, documentada y fácil de darle mantenimiento. Adicionalmente, se obliga al programador a separar claramente la construcción, a nivel físico de la estructura de datos, de la aplicación que se le dará. Las estructuras de datos implementadas como objetos pueden guardarse como unidades de software que pueden emplearse en diferentes aplicaciones.

4.2.3 Lenguaje JAVA

Java es un lenguaje de programación orientado a objetos de alto nivel, con el cual se pueden escribir tanto programas convencionales como para Internet, que favorece el desarrollo y reduce las posibilidades de cometer errores, a la vez es potente y flexible.

La plataforma Java se encuentra por encima de otras plataformas, el código que genera su compilador no es específico para una máquina física en particular, sino de una máquina virtual (Java Virtual Machine). Aún cuando existen múltiples implantaciones de la Maquina Virtual de Java, cada una específica de la plataforma sobre la cual subyace, existe una única especificación de la máquina virtual que proporciona una vista independiente del hardware y del sistema operativo sobre el que se esté trabajando. De esta manera, el código intermedio (bytecode) generado al compilar un programa Java, puede ejecutarse donde sea.

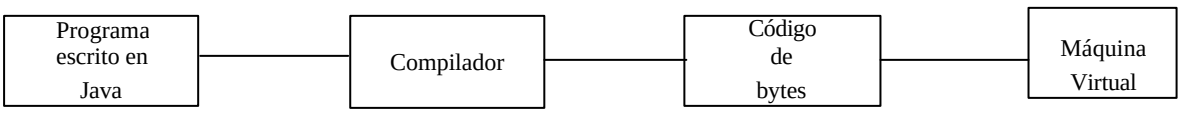

Figura 4.1 Etapas para ejecución de un programa Java.

La máquina virtual de Java se encarga de proporcionar la vista de un nivel de abstracción superior, donde además de la independencia de la plataforma antes mencionada, presenta un lenguaje de programación simple, con verificación estricta de tipos de datos, múltiples hilos, y con recolección automática de basura.

4.3 Programación para base de datos

Con el paso de los años, se han inventado múltiples tecnologías para hacer que el acceso a bases de datos sea más eficiente y seguro. Las bases de datos relacionales estándar admiten índices, activadores, procedimientos almacenados y gestión de transacciones. La tecnología JDBC, desarrollada por Sun Microsystems como un API para la programación de base de datos con el lenguaje Java, admite todas estas posibilidades.

4.3.1 El diseño de JDBC

Conforme al crecimiento de los desarrolladores en plataforma Java, tanto programadores como fabricantes de bases de datos y proveedores de herramientas, se requería que la biblioteca estándar de Java se extendiera para poder acceder a la base de datos mediante el lenguaje de consulta estructurado (SQL). Debido a tal requerimiento, Sun buscó la forma de extender Java para que pudiera comunicarse con cualquier base de datos empleando Java “puro”, pero esto era algo imposible, debido a que hay un número demasiado grande de bases de datos.

Sun decidió, después de acordar con los fabricantes y proveedores de bases de datos, proporcionar un API en Java “puro” para acceder a SQL junto con un administrador de controladores que permitiese que los controladores de terceras partes se conectaran con bases de datos concretas. Entonces, como resultado, se crearon dos interfaces. Los programadores de aplicaciones utilizan el API de JDBC y los fabricantes y proveedores de herramientas emplean el API de controladores de JDBC.

Con esto, el objetivo último de JDBC es hacer posible lo siguiente:

- Los programadores pueden escribir aplicaciones en el lenguaje de programación Java para acceder a cualquier base de datos, empleando sentencias estándar de SQL y seguir empleando las convenciones del lenguaje Java.
- Los fabricantes de bases de datos y de herramientas para bases de datos puedan proporcionar controladores de bajo nivel. De este modo, optimizan los controladores para sus propios productos.

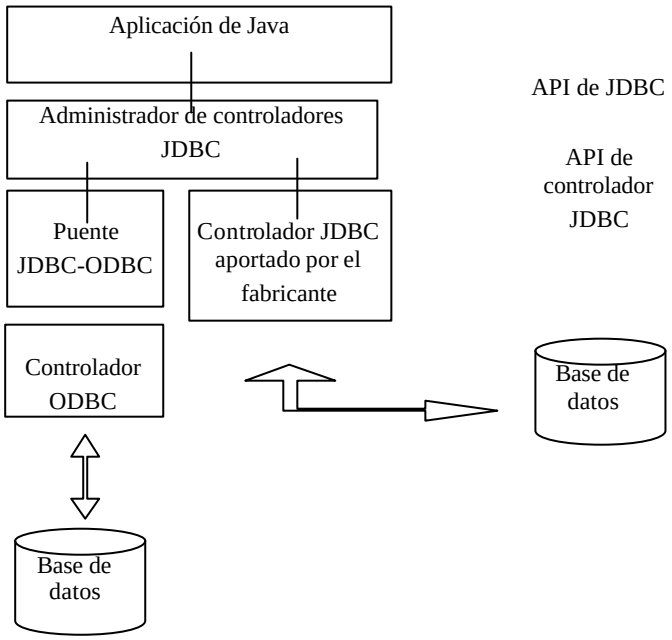

Figura 4.2 Arquitectura JDBC.

Actualmente, existe una gran cantidad de compañías que se dedican al desarrollo de aplicaciones con conexión a base de datos sobre la plataforma Java, utilizando JDBC, pero se encuentran con problemas para escalabilidad y mantenimiento de la aplicación, debido a que en el código de negocio, la programación para el acceso a la base de datos implementando lenguaje SQL es estática, lo que implica mucho trabajo al realizar una modificación, esto es porque puede tenerse código en diferentes clases que manejen la entidad a modificar de la base de datos; además de que la cantidad de código requerido crece mucho para tablas con demasiados atributos.

También se presenta el problema de manejar conjuntos de datos obtenidos de la base de datos como objetos individuales, o colecciones de objetos; sobre todo cuando el resultado involucra datos de distintas tablas, por lo se programa una mayor cantidad de código para hacer la relación de un objeto (Value Object, VO) con un registro de un conjunto de resultados de la base de datos. Como solución a este problema mencionado, y a algunos otros, existen tecnologías para la persistencia de datos que ayudan a la programación, de esto se hablará un poco más adelante.

4.3.2 Lenguaje de programación persistente

Algunos lenguajes de programación han agregado componentes a su estructura para contar con una interfaz que les permita realizar el tratamiento de datos con el lenguaje de consulta estructurada, debido a que SQL es muy efectivo en el acceso a los datos. Esta es la manera tradicional de realizar las interfaces de las bases de datos con los lenguajes de programación, un ejemplo de ello es el API JDBC de Java, mencionado en el punto anterior.

Los *lenguajes de programación persistentes* son los lenguajes de programación extendidos con estructuras para el tratamiento de los datos persistentes.

Los lenguajes de programación persistentes pueden distinguirse de los lenguajes con SQL incorporado, al menos, de dos maneras:

1. En los lenguajes incorporados el sistema de tipos del lenguaje anfitrión suele ser diferente del sistema de tipos del lenguaje para el tratamiento de los datos. Los programadores son responsables de las conversiones de tipos de datos entre el lenguaje anfitrión y SQL. Por el contrario, en los lenguajes de programación persistentes, el lenguaje de consultas se halla totalmente integrado con el lenguaje anfitrión y ambos comparten el mismo sistema de tipos.
2. Los programadores que usan lenguajes de consultas incorporados son responsables de la escritura de código explícito para la búsqueda en memoria de los datos de la base de datos. Si se realizan actualizaciones, los programadores deben escribir explícitamente código para volver a guardar los datos actualizados en la base de

datos. Por el contrario, en los lenguajes de programación persistentes, los programadores pueden trabajar con datos persistentes sin tener que escribir explícitamente código para buscarlos en la memoria o volverlos a guardar en el disco.

4.3.3 Sistemas Java persistente

Algunas de las características del modelo para la persistencia de objetos en los programas de Java, son las siguientes:

- *Persistencia por alcance*: los objetos no se crean explícitamente en la base de datos. El registro explícito de un objeto como persistente hace que el objeto sea persistente.
- *Mejora del código utilizando XML*: En lugar de declarar en el código de Java la definición de una clase persistente, se especifica en un archivo de configuración, generalmente de tipo XML.

4.4 Persistencia de datos con Hibernate

Hibernate es una tecnología basada en la arquitectura Java para el Mapeo Objeto-Relacional, mejor conocido como ORM (*Object Relational Mapping*), de libre distribución, es de las más maduras y completas. Básicamente una ORM intenta hacer todas las tareas pesadas por nosotros. Con una buena ORM, se tiene que definir la forma en que se establece la correspondencia entre las clases (objetos) y las tablas (indicando que propiedad corresponde con que columna, que clase con que tabla). Para esto, se podrá utilizar POJOs (*Plain Old Java Objects*) en la aplicación e indicarle a la ORM que los haga persistentes, un POJO será la representación de un registro de una tabla específica de la base de datos.

Actualmente su uso se está extendiendo y además esta siendo desarrollada de forma muy activa. Una característica muy importante, que distingue Hibernate de otras soluciones para el problema de la persistencia como los EJB de entidad, es que la clase Hibernate

persistente puede utilizarse en cualquier contexto de ejecución, no se necesita un contenedor especial para ello.

4.4.1 Arquitectura Hibernate

Hibernate utiliza una configuración de datos y la base de datos para proporcionar servicios de persistencia a la aplicación, está constituido por diferentes interfaces repartidas en dos capas: *persistencia* y *negocio*. La capa de negocio está situada sobre la capa de persistencia, ya que la capa de negocio actúa como un cliente de la capa de persistencia.

Una vista en alto nivel de la arquitectura de Hibernate es la mostrada en la siguiente figura:

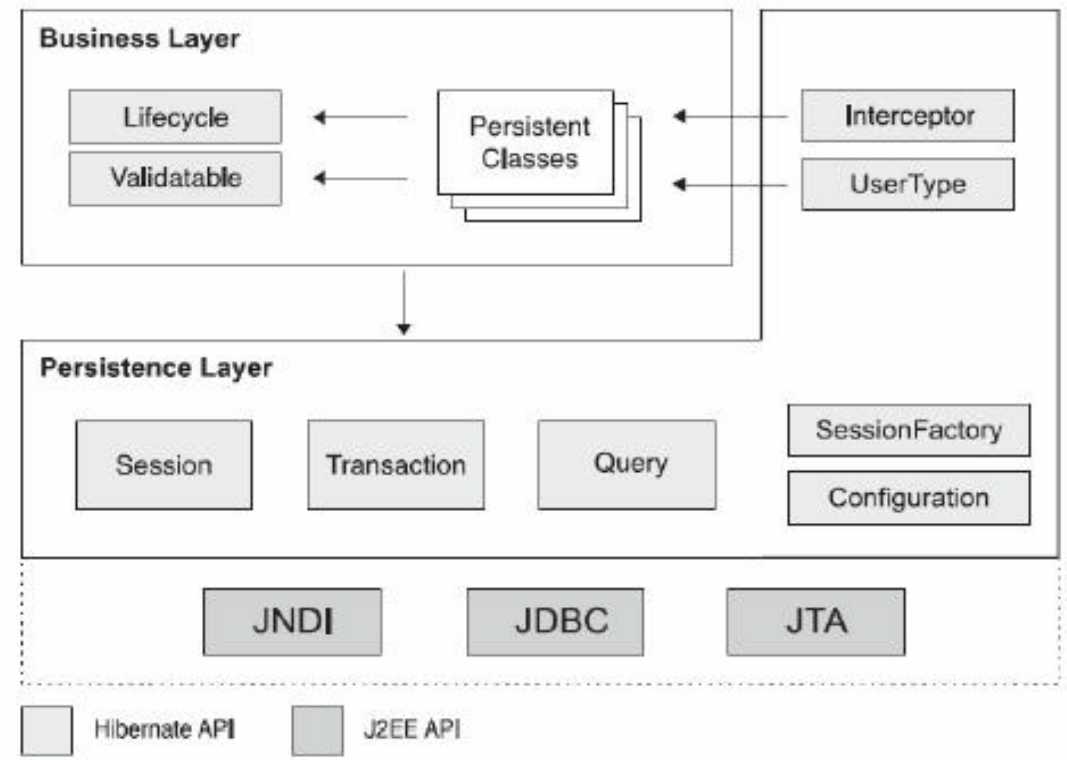

Figura 4.3 Arquitectura de Hibernate.

Las interfaces pueden clasificarse como sigue:

- Interfaces llamadas por la aplicación para realizar operaciones básicas (inserciones, borrados, consultas, etc.).
- Interfaces llamadas por el código de la infraestructura de la aplicación para configurar Hibernate.
- Interfaces callback que permite a la aplicación reaccionar ante determinados eventos que ocurren dentro de la aplicación.
- Interfaces que permiten extender las funcionalidades del mapeo de Hibernate.

Hibernate también hace uso de APIs de Java, tales como JDBC, JTA y JNDI.

4.4.2 Entorno de aplicación de Hibernate

Hibernate puede configurarse y ejecutarse en la mayoría de aplicaciones Java y entornos de desarrollo. Generalmente, Hibernate se utiliza en aplicaciones cliente-servidor de dos y tres capas, ejecutándose solamente del lado del servidor.

Para utilizar Hibernate en una aplicación, se debe iniciar con su configuración. La configuración es distinta para entornos gestionados y no gestionados:

- *Entorno gestionado.* Los *pools* de recursos, tales como conexiones a la base de datos que permiten establecer los límites de las transacciones y la seguridad, se especifican de forma declarativa, es decir, están contenidas en sus metadatos. Un servidor de aplicaciones como JBOSS, Bea WebLogic o IBM WebSphere implementan un entorno gestionado para Java.
- *Entorno no gestionado.* Proporciona una gestión básica de concurrencia a través de un *pooling de threads*. Un contenedor de servlets, como Tomcat, proporciona un entorno no gestionado para aplicaciones Web con Java. Una aplicación stand-alone también se considera como no gestionada. Los entornos no gestionados no proporcionan infraestructura para transacciones automáticas, gestión de recursos, o

seguridad. La propia aplicación es la que gestiona las conexiones con la base de datos y establece los límites de las transacciones.

4.4.3 Clases persistentes

Hibernate trabaja mejor con un modelo de dominio que implementa POJOs, es decir, con objetos Java que no siguen ningún modelo convencional ni implementan alguna interfaz especial, son serializables, tienen un constructor sin argumentos y permiten acceder a las propiedades a través de los métodos getter y setter correspondientes. Los pocos requerimientos que Hibernate impone ante el modelo de dominio son también las mejores prácticas para el modelo de programación de POJOs. Así, la mayoría de los POJOs son compatibles con Hibernate sin necesidad de hacer algún cambio. Algunas propiedades de un POJO representan asociaciones a algún otro POJO.

A continuación se muestra un ejemplo de la implementación de un POJO para una entidad llamada cuenta:

```
public class Cuenta {
        private String login;       //Atributo identificador
        private String password;
        public Cuenta () {
        }
        public Cuenta (String login, String password) {
                this.login = login;
                this.password = password;
        }
        public void setLogin (String login) {
                this.login = login;
        }
```

cuenta		
PK	login	varchar(10)
	password	varchar(15)

```
        public String getLogin () {
                return login;
        }
        public void setPassword (String password) {
                this.password = password;
        }
        public String getPassword () {
                return password;
        }

}
```

Listado 4.1 Clase persistente para la entidad cuenta

4.4.4 Definiendo el mapeo de metadatos

Las herramientas de ORM necesitan el formato Metadato para especificar el mapeo entre clases y tablas, propiedades y columnas, asociaciones y llaves foráneas, tipos de datos de Java y tipos SQL de la aplicación. Esta información, llamada "***mapeo de metadatos objeto/relacional***", define la transformación entre los diferentes tipos de sistemas y representaciones de la relación.

4.4.4.1 Metadato en XML

Actualmente, el formato de metadatos objeto/relacional más popular es XML. Los documentos de mapeo escritos en XML son ligeros, son entendibles para el humano, son fáciles de manipular y pueden ser personalizados en tiempo de despliegue.

A continuación se muestra un ejemplo del archivo de mapeo XML para la clase Cuenta definida anteriormente:

```
<hibernate-mapping package="paquete.pojos">
        <class name="Cuenta" table="CUENTA">
                <id name="login"
                        column="LOGIN"
                        lenght=”10”
                        type=”string”>
                </id>
                <property name="password"
                        column="PASSWORD"
                        length=”15”
                        type=”string”
                        not-null="true"/>
        </class>
</hibernate-mapping>
```

Listado 4.2 Metadato para la clase persistente Cuenta

1. En el mapeo de varias clases con Hibernate, se recomienda ser declarado en un archivo de mapeo diferente; se requiere para la validación sintáctica del XML. La convención establece que el nombre del archivo será el mismo que el nombre de la clase mapeada, agregando el sufijo *hbm*, por ejemplo, Cuenta.hbm.xml.
2. Los mapeos son declarados dentro del elemento <hibernate-mapping>. Es posible, pero no recomendable (ver punto anterior), declarar el mapeo de distintas clases tantas como se requieran, usando multiples elementos <class>.
3. La clase Cuenta es mapeada a la entidad CUENTA. Cada renglón en esta tabla representa una instancia del tipo Cuenta.

4. El elemento <id> es utilizado para definir los detalles de la identidad del objeto. Cada renglón de la tabla Cuenta será referenciada a través de una llave primaria, según el valor de dicho elemento, la cual coincide con la identidad del objeto cuya instancia se encuentra en memoria. En este caso, el valor de la llave primaria será proporcionada por el usuario y no automáticamente, es decir, no será asignada por Hibernate mediante algún algoritmo propio.
5. La propiedad password de tipo java.lang.String es mapeado a la base de datos como la columna PASSWORD. Se puede observar que el tipo declarado en el mapeo (string) es un tipo de Hibernate, y no el tipo de dato de Java o el tipo SQL de la columna.

4.4.4.2 Modelo de asociaciones

El manejo de las asociaciones entre clases y sus relaciones entre tablas es el alma de ORM. El mayor problema en el que se ve envuelto un ORM es la administración de asociaciones. El modelo de asociaciones de Hibernate es extremadamente rico, pero no es sencillo.

4.4.4.2.1 Multiplicidad

Al describir y clasificar asociaciones, casi siempre se maneja la asociación de multiplicidad: "muchos-a-uno" (*many-to-one*), inversamente de "uno-a-muchos" (*one-to-many*), "muchos-a-muchos" (*many-to-many*); en algunos casos se maneja la asociación "uno-a-uno" (*one-to-one*).

En la persistencia de objetos con Hibernate, no se restringe un número, como mínimo ni como máximos, de los objetos al hablar de "muchos" en la asociación. Con una propiedad del objeto de persistencia, se representa la asociación de *muchos* como una colección de tipo *java.lang.Set* (para evitar repetición de objetos de acuerdo al método *equals()*) y la asociación de *uno* como un objeto de tipo persistente, por ejemplo Cuenta.

	Representación en POJO	***Mapeo***

Asociación muchos	...
	private Set objetos = new HashSet();
	...
	//metodos getter y setter correspondientes
	...
		...
		<set name="objetos" >
		...
		</set>
		...

Asociación uno	... private Cuenta cuenta; ... //metodos getter y setter correspondientes ...	... <many-to-one name="cuenta" column="LOGIN" not-null="false " class="Cuenta"/> ...

Tabla 4.1 Mapeo de asociaciones de multiplicidad

4.4.5 El ciclo de vida de la persistencia

Desde que Hibernate es un mecanismo de persistencia transparente (que separa completamente las clases de persistencia de la lógica de persistencia de si misma) es posible escribir lógica de la aplicación que es inconsistente de si los objetos operan sobre un estado persistente o un estado temporal que existe sólo en memoria. La aplicación no debería necesitar cuidar que un objeto este persistente cuando se invoque su método.

Sin embargo, en una aplicación con estado persistente la aplicación debe interactuar con la capa persistente siempre que necesite propagar el estado llevado a cabo en memoria a la base de datos.

Diferentes implementaciones de ORM utilizan diferente terminología y define diferente estado y transiciones de estado para el ciclo de vida de la persistencia. Por otra parte, los estados de los objetos usados internamente pueden ser diferentes a los expuestos a la aplicación cliente. Hibernate define tres estados: transitorio (*transient*), persistente (*persistent*) y separado (*detached*).

4.4.5.1 Objetos transitorios

En Hibernate, usando el operador ***new*** los objetos no son inmediatamente persistentes. Su estado es *transitorio*, lo cual significa que no están asociados con algún renglón de alguna tabla de la base de datos, así que su estado se pierde tan pronto que dejan de ser referenciados por la aplicación.

4.4.5.2 Objetos persistentes

Un objeto persistente es una identidad de base de datos (database identity). Esto significa que un objeto persistente es una representación en memoria de un renglón específico de alguna tabla de la base de datos. Para esto, el objeto debe cumplir con la identidad (object identity) e igualdad (object equality) de un objeto Java, que se refiere a la localidad de memoria que ocupa y a los valores de sus atributos, respectivamente.

La instancia persistente puede ser un objeto instanciado por la aplicación y entonces persistido al llamar a un método del administrador de persistencia (clase Session del API de Hibernate). Las instancias persistentes participan en transacciones (su estado es sincronizado con la base de datos al termino de la transacción). Cuando una transacción hace *commit*, el estado oculto en memoria es propagado a la base de datos por la ejecución de las sentencias SQL: INSERT, UPDATE y DELETE.

4.4.5.3 Objetos separados

Cuando una transacción se completa, la instancia persistente asociada con el administrador de persistencia todavía existe, sin embargo, la instancia pierde su asociación con el administrador de persistencia al utilizar el método que cierra el contexto persistente.

4.4.6 Recuperar objetos persistentes

Regresar objetos persistentes desde la base de datos es una de las partes más interesantes y complejas de trabajar con Hibernate. Este proporciona las siguientes maneras de obtener objetos fuera de la base de datos:

- Navegando a través de un objeto persistente, cargado previamente, para tener acceso a los objetos asociados a través de los métodos de acceso. Es decir, si por ejemplo la clase Cuenta contara con una asociación de "uno" de alguna otra clase (Perfil), y se quisiera conocer una propiedad (nombre) de esa clase, entonces se podría hacer mediante:

```
cuenta.getPerfil().getNombre();
```

Hibernate automáticamente cargara las propiedades del objeto asociado si el contexto persistente está abierto.

- Regresando el objeto a partir de su identificador, lo cual es más conveniente y se ejecuta el método sólo cuando el valor del identificador único de un objeto es conocido.
- Usando el *Lenguaje de Consulta de Hibernate* (HQL, por sus siglas en inglés), el cual es un lenguaje de consulta parecido a SQL pero totalmente orientado a objetos y es a través de la manipulación de cadenas.
- Usando la clase *Criteria* de Hibernate, el cual proporciona un tipo seguro y una forma para ejecutar consultas orientado a objetos sin la necesidad de manipular cadenas de consultas.

- Usando consultas SQL nativas, donde Hibernate tiene cuidado de mapear el conjunto de resultados de JDBC a gráficos de objetos persistentes.

En la aplicación junto con Hibernate, se puede hacer una combinación de estas técnicas. Cada método para la obtención de los objetos usa diferente estrategia de ejecución, es decir, que parte del objeto persistente será regresado. La meta es encontrar el mejor método para la aplicación al mismo tiempo minimizar el numero de consultas SQL para el mejor funcionamiento.

Capítulo 5

Analizador XML

5.1 Introducción

Las herramientas o programas que leen el lenguaje XML y comprueban si el documento es válido sintácticamente, se denominan *analizadores* o "***parsers***". Un parser XML es un módulo, biblioteca o programa que se ocupa de transformar un archivo de texto en una representación interna. En el caso de XML, como el formato siempre es el mismo, no necesitamos crear un parser cada vez que hacemos un programa, sino que existen un gran número de analizadores sintácticos disponibles que pueden averiguar si un documento XML cumple con una determinada gramática.

Podemos hacer una distinción entre las herramientas que son *"validantes"* y las que son "*no validantes"*. La diferencia radica en que las validantes verifican que el documento, además de estar bien formado de acuerdo a las reglas de XML, responda a una estructura definida en una DTD.

5.2 Analizadores

Un analizador XML ofrece los servicios de serializar y deserializar documentos XML, es decir, de pasar de un formato serial (el texto contenido en el documento) a un formato no serial (como una estructura de datos o un objeto con llamadas a métodos, uno por cada componente del documento) seleccionado por el programador. Los API más habituales son SAX y DOM.

5.2.1 SAX

La Simple API para XML o *Simple API for XML* (SAX) es una interfaz simple para aplicaciones XML. Funciona por eventos y métodos asociados. A medida que el analizador va leyendo el documento XML y encuentra los componentes (eventos) del documento

(elementos, atributos, valores, etc) o detecta errores, va invocando a las funciones que ha asociado el programador

El analizador SAX notifica los eventos pero no almacena el documento en forma alguna; es responsabilidad de los manejadores de eventos decidir si desean o no construir una estructura de datos.

Siempre que se utiliza un analizador SAX, es necesario un manejador que defina las acciones para los distintos eventos del analizador. La interfaz *ContentHandler* define varios métodos de retrollamada que ejecuta el analizador a medida que analiza el documento, como por ejemplo:

- **startElement** y **endElement**: se invocan cuando se llega a un marcador de comienzo o final.
- **characters**: se invoca cuando se encuentran datos de tipo carácter.
- **startDocument** y **endDocument**: cada uno se invoca una vez, al principio y al final del documento.

Por ejemplo, cuando se está analizando el fragmento:

```
<Account>
        <FileID>GNM234345234</FileID>
</Account>
```

El analizador se asegura de generar las llamadas siguientes:

- **startElement**, nombre del elemento: **Account**
- **startElement**, nombre del elemento: **FileID**
- **characters**, contenido: **GNM234345234**
- **endElement**, nombre del elemento: **FileID**

- **endElement**, nombre del elemento: **Account**

El manejador de eventos tiene que rehacer estos métodos y hacer que ejecuten la acción oportuna mientras se va analizando el archivo.

5.2.2 DOM

El Modelo de Objetos de Documento o *Document Object Model* (DOM) es un modelo de objetos estandarizado para documentos HTML y XML. DOM define la estructura lógica de los documentos y el modo en que se accede y manipula un documento.

DOM proporciona una representación de un documento XML en forma de árbol. El árbol se puede recorrer y transformar. El principal inconveniente es el árbol debido a que: 1) sólo se accede a los datos una vez que se han leído todos; y, 2) el árbol es un objeto cargado en memoria; esto es problemático para documentos grandes y complejos. El analizador DOM se basa en un analizador SAX. Va construyendo el árbol a medida que recibe eventos del analizador.

Al igual que DOM, existen diversas APIs que hacen uso de un analizador SAX para realizar la serialización y deserialización en un nivel alto, permitiendo reducir la cantidad de código a programar, o bien, facilitando la programación, cabe considerar que cada una cuenta con sus propias ventajas y desventajas. Estas APIs están inclinadas a algunos lenguajes de programación en específico, como la POO, un ejemplo de ello es el lenguaje de programación Java.

5.2.3 XStream

XStream es una API que permite pasar tanto de documentos XML a objetos Java (de forma muy sencilla) así como también realizar el proceso inverso; es decir, serializar objetos Java en formato XML.

Características:

- Es capaz de serializar cualquier objeto Java.
- No incluye dependencia con Java de ningún tipo.
- Soporta referencias a otras clases.
- Soporta sin problemas la serialización de listas de objetos.
- Emplea tecnología propia para la lectura del documento XML.

5.2.3.1 Arquitectura XStream

La arquitectura de XStream cuenta con dos principales componentes que son los siguientes:

- **Convertidores.** Cada vez que XStream encuentra un objeto que necesita ser convertido a XML, utiliza un convertidor para pasar de tipos de objetos particulares a XML (y viceversa). XStream cuenta con convertidores para tipos de datos comunes, incluyendo a los primitivos(int, long, double, etc), cadenas(String), colecciones(Collection), arreglos(array), nulos(null), fechas(Date), etc. XStream también cuenta con un convertidor por default, que es usado cuando ninguno de los convertidores coincide con el tipo de dato analizado.

- **Drivers.** XStream es abstracto desde la perspectiva de datos XML, esto se logra utilizando las interfaces HierarchicalStreamWriter y HierarchicalStreamReader, que son usadas para la serialización y deserialización respectivamente. Ésta abstracción permite a XStream leer XML directamente desde flujos (secuencias de bytes), usando un parseo XML o manipulando directamente su estructura (tal como DOM). XStream cuenta con implementaciones de las interfaces Reader y Writer para una gran cantidad de librerías XML.

Capítulo 6

Web Service

6.1 Introducción

Las aplicaciones requieren a menudo datos externos, de otro departamento de la misma o diferente empresa, contenidos en distinta base de datos, para efectuar alguna tarea específica. En estas situaciones, la empresa externa o el departamento no están dispuestos a permitir acceso directo a su base de datos usando SQL, sino a proporcionar información limitada a través de interfaces predefinidas.

6.1.1 ¿Qué es un servicio Web?

Un servicio Web es una pieza de la lógica de negocio situado en alguna parte de Internet, este es accesible a través de protocolos tales como HTTP o SMTP. Es un nuevo tipo de tecnología basado en una estandarización XML, el cual provee un lenguaje neutral que permite adoptar una forma para representar los datos que serán compartidos entre diversas tecnologías.

Características de un servicio Web:

- *Base*: Utiliza XML para hacer una representación de todos los datos para todos los protocolos y tecnologías de servicios Web que son creadas.
- *Bajo acoplamiento*: Se debe tomar en cuenta que la interfaz de un servicio Web puede cambiar en cualquier momento, y aunque esto suceda, no debe de tener consecuencias que perjudiquen la interacción de un cliente con el servicio Web. Una solución es utilizar alguna técnica que nos permita desarrollar programas con un bajo acoplamiento. Un alto acoplamiento quiere decir que el cliente y la lógica del servicio están fuertemente ligados, lo que implica que si uno cambia, el otro forzosamente tendrá que adaptarse al nuevo cambio. La adopción de una arquitectura con un bajo acoplamiento, trae como beneficio realizar un mantenimiento de software mucho más rápido y eficiente. Esta es una de las grandes ventajas de los Servicios Web.

- *Síncrono o Asíncrono*: Sincronizado se refiere a la relación entre el cliente y la ejecución del servicio. En la invocación sincronizada, el cliente tiene que esperar hasta que el servicio sea completado, para posteriormente continuar con sus tareas. En una operación asíncrona, el cliente puede invocar un servicio y después ejecutar otras funciones, sin tener que esperar a que el servicio concluya. El cliente asíncrono, recupera el resultado en determinado lapso de tiempo, mientras que los clientes síncronos, reciben el resultado cuando el servicio es completado.
- *RPCs (Supports Remote Procedure Calls)*: Los servicios Web permiten a los clientes invocar procedimientos, funciones y métodos de manera remota, utilizando un protocolo basado en XML. Los procedimientos remotos exponen parámetros de entrada y salida que el Servicio Web debe soportar.
- *Intercambio de documentos*: Los Servicios Web soportan un intercambio transparente de documentos, que facilitan la integración de negocio.

6.2 SOAP

El Protocolo Simple de Acceso a Objetos o Simple Object Access Protocol (SOAP) define una norma que permite invocar procedimientos usando XML, con el cual se representa la entrada y la salida de los procedimientos.

SOAP define un esquema XML estándar para representar el procedimiento, el nombre del procedimiento y de los indicadores de estado del resultado, como fallo y error. Los parámetros y resultados de los procedimientos son datos XML dependientes de las aplicaciones incorporadas en las cabeceras de SOAP.

HTTP se usa normalmente como el protocolo de transporte para SOAP, pero también se puede emplear un protocolo basado en mensajes (como el correo electrónico sobre el protocolo SMTP).

SOAP está definido por su propio XML Scheme y hace un gran uso de los Espacios de Nombre (*Namespaces*). Todos los mensajes SOAP que son enviados, consisten en documentos XML basados en elementos del estándar SOAP y datos de aplicación, dentro de los cuales, los espacios de nombre proveen una ayuda importante, debido a que se puede diferenciar los datos del estándar SOAP.

```
<?xml version='1.0' encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://shcemas.xmlsoap.org/soap/envelope">
    <env:Header />
    <env:Body>
        <ListOfAccount xmlns="http://www.unam.com/sca">
            <Account>
            . . .
            </Account>
        </ListOfAccount>
    <env:Body>
</env:Envelope>
```

Listado 6.1 Estructura SOAP básica

El propósito principal de SOAP es establecer un XML estándar para realizar un empaquetado de los datos de la aplicación que serán intercambiados entre las diferentes plataformas. Para hacer esto, SOAP define un conjunto de elementos, cada uno para contener diferentes tipos de datos. La etiqueta *<Envelope>* es el elemento raíz de un mensaje SOAP, y tiene dos hijos directos, *<Header>* y *<Body>*. La etiqueta <Header> generalmente es usado para transportar la infraestructura de los datos, como los tokens de seguridad, los Ids de transacción e información de ruteo. La etiqueta <Body> transporta la información de la aplicación que será intercambiada.

6.3 WSDL

El Lenguaje de Definición de Servicios Web o Web Service Definition Language (WSDL) es una tecnología XML que describe la interfaz de un servicio Web en una forma estandarizada. WSDL es un estándar, basada en SOAP, que dicta cómo un servicio Web debe representar los parámetros de entradas y salidas de una invocación externa, las estructuras de sus funciones y la naturaleza de la invocación (si permite puras entradas; entradas y salidas; etc.), y los protocolos de los servicios. En otras palabras, WSDL permite que los clientes entiendan cómo deben interactuar con el servicio. Además WSDL especifica la URL y el número de puerto que se utilizarán para invocar el servicio Web.

Supongamos que queremos implementar un componente de un servicio Web utilizando el lenguaje Java, como se muestra en la siguiente declaración:

```
public interface Account {
        public String makeAccount(FileID, TipoNegocio, MerchantID);
}
```

Listado 6.2 Interface Account

Todas las aplicaciones no podrían invocar este método usando únicamente SOAP, debido a que están implementadas en lenguajes diferentes. Esto es porque no todos entienden el lenguaje Java, y una forma de solucionarlo es describir el servicio Web mediante XML, especialmente con el lenguaje de marcado WSDL, con el cual podríamos describir el tipo de mensaje SOAP que podría ser enviado para poder invocar el método makeAccount(). El documento que describe el método mencionado podría lucir de la siguiente manera:

```
<?xml version="1.0"?>
        <definitions name="Account"
                xmlns="http://schemas.xmlsoap.org/wsdl/"
```

```
                xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:sca="http://www.unam.mx/sca"
                targetNamespace="http://www.unam.mx/sca">
        <!--El elemento message  describe parámetros y valores de retorno -->
        <message name="RequestMessage">
                <part name="FileID" type="xsd:string" />
                <part name="TipoNegocio" type="xsd:int" />
                <part name="MerchantID" type="xsd:int" />
        </message>
        <message name="ResponseMessage">
                <part name="AfiliacionSCAState" type="xsd:string" />
        </message>
        <!-- El elemento portType describe la interfaz abstracta de un servicio web -->
        <portType name="Account">
                <operation name="makeAccount">
                        <input message="sca:RequestMessage"/>
                        <output message="sca:ResponseMessage"/>
                </operation>
        </portType>
        <!-- El elemento binding nos dice qué protocolos y estilos de codificación se van a
usar -->
        <binding name="AccountBinding" type="sca:Account">
            <soap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http"/>
           <operation name="makeAccount">
                <soap:operation soapAction="" />
```

```
                    <input>
                        <soap:body use="literal"
                            namespace="http://www.unam.mx/sca"/>
                    </input>
                    <output>
                        <soap:body use="literal"
                            namespace="http://www.unam.mx/sca"/>
                    </output>
                </operation>
            </binding>
            <!-- El elemento service nos dice la dirección electrónica del servicio web -->
            <service name="AccountService">
                <port name="AccountPort" binding="sca:AccountBinding">
                    <soap:address
location="http://www.unam.mx/webservices/sca/account" />
                </port>
            </service>
</definitions>
```

Listado 6.3 WSDL de la interface Account

6.3.1 El elemento <definitions>

El elemento raíz del documento WSDL es la etiqueta <definitions>. Usualmente el documento WSDL declara los espacios de nombre XML dentro de este elemento. En el ejemplo anterior, el elemento <definitions> crea cuatro espacios de nombre.

El XML, de entrada tendrá el espacio de nombre xmlns="http://schemas.xmlsoap.org/wsdl/", que es el propio del documento WSDL. El prefijo **xsd** es asignado para XML Scheme, este

último es usado para utilizar los tipos de datos tales como *xsd:string*, *xsd:int*, y *xsd:dateTime*, como se muestra en el elemento <message>. El prefijo **sca** es asignado a la URL del Sistema de Control de Afiliaciones (SCA), que indica el espacio de nombre de SCA para su acceso.

6.3.2 Los elementos <portType> y <message>

El elemento <portType> describe las operaciones del servicio Web que pueden ser realizadas (métodos Java) entre el cliente y el servidor. Las operaciones pueden tener mensajes de entrada, salida y de falla. Los mensajes de entrada describen el tipo de mensaje SOAP que el cliente quiere enviar al servicio. Los mensajes de salida describen el tipo de mensaje SOAP que el cliente espera como respuesta del servicio. Los mensajes de falla describen cualquier mensaje de error SOAP que el servicio puede regresar.

El elemento <message> proporciona una abstracción común para el paso de mensajes entre el cliente y el servidor. Puede haber más de un elemento <message> en un documento WSDL, uno para cada mensaje que se comunica entre el cliente y el servidor. Cada mensaje contiene uno o más elementos "*part*" que describen las piezas del contenido del mensaje, es decir, describen los tipos para los datos de entrada y salida de la operación del servicio descrito en el elemento <porType>.

6.3.3 El elemento <types>

Si el servicio Web necesita utilizar sus propios tipos de datos, estos pueden definirse dentro del elemento <types> el cual es el primer hijo del elemento <definitions>.

6.3.4 Los elementos <binding> y <service>

El elemento <binding> contiene las definiciones de un protocolo como SOAP a un determinado bindingType. Las definiciones binding especifican detalles de formatos del mensaje y el protocolo, es decir, describe el tipo de codificación usado para enviar y recibir mensajes.

El elemento <service> especifica el puerto para la comunicación entre el cliente y el servidor. Un puerto es el extremo concreto de un servicio Web al que se hace referencia por una dirección única. Cada elemento <port> se asocia de uno en uno a una ubicación con un elemento <binding>. Puede haber más de un elemento <service> en un documento WSDL; el atributo *name* distingue un servicio de otro. Debido a que pueden existir varios puertos en un servicio, estos disponen también de un atributo *name*.

6.4 UDDI

UDDI (Universal Description, Discovery, and Integration) es una especificación que describe el estándar para publicar y descubrir servicios Web sobre Internet. UDDI no es una pieza clave de un servicio web, como lo es XML, SOAP, y WSDL, su objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL.

La analogía usada para describir UDDI es que provee páginas electrónicas blancas, amarillas y verdes. Mediante las páginas blancas podemos localizar una compañía utilizando un nombre o identificador; con las amarillas se pueden encontrar por el tipo de negocio o categoría de producto; y, también podemos obtener información sobre un servicio Web examinando los elementos técnicos aportados por las propias compañías (páginas verdes). En otras palabras, UDDI es un directorio electrónico que permite a las organizaciones publicar sus negocios y servicios Web para que puedan ser localizados por otras organizaciones y servicios Web.

Capítulo 7

Sistema SCA

7.1 Introducción

De acuerdo al ejemplo práctico que se planteo en el capitulo de introducción, existen varias organizaciones asociadas para el desarrollo de una solución, enfocada a la sociedad, que facilite el manejo de los recursos monetarios de los ciudadanos; tanto para los consumidores que requieren solventar sus gastos diarios así como para los comerciantes que proporcionan algún tipo de servicio, además brindar cierta confiabilidad para la realización de las transacciones.

El proyecto a grandes rasgos, a través de bancos mexicanos, tiene como finalidad instalar terminales punto de venta (máquinas de cobro con tarjetas de crédito y débito) en pequeños y medianos comercios.

Las tiendas misceláneas o de abarrotes son pequeños comercios que nunca pueden faltar en las colonias populares o en las grandes ciudades del país, y es por eso que se han buscado maneras para que los clientes puedan realizar sus compras con una tarjeta bancaria dentro de estos establecimientos. Una solución, fue la de colocar terminales puntos de venta en estas tiendas, las cuales benefician al consumidor que no necesariamente debe cargar efectivo para adquirir algún producto. Se quiere, con este proyecto, abarcar a todos los comercios por muy pequeños que sean, a través de una afiliación con algún banco mexicano que elija.

Se prevé que con el crecimiento de la red de terminales, esta solución puede mejorar añadiendo más programas y promociones, en beneficio de los usuarios, como ofrecer premios para el consumidor por el simple uso del plástico.

7.2 Análisis del sistema

La infraestructura de este proyecto será el sistema realizado por cada una de las organizaciones para la comunicación de datos de los comercios que requieran manejar terminales en su negocio. El proceso de la información será dividida en varias etapas, en las cuales se ven involucradas algunas de las organizaciones. Uno de los sistemas más robusto es el que denominamos Sistema de Control de Afiliaciones (SCA), el cual se ve involucrado en todos los procesos del manejo de la información, dicha información corresponderá a la afiliación del comercio.

7.2.1 Etapas del proceso de la Afiliación

El proceso de Afiliación tiene dos posibles orígenes: desde el punto de vista de SCA:

1. Un comercio puede contactar al centro de contacto de una de las organizaciones vía telefónica, esta organización, al cual llamamos CRM, recolectará todos los datos necesarios para realizar una solicitud de afiliación de dicho comercio. CRM puede recibir una gran cantidad de solicitudes de distintos comercios, los cuales enviará a SCA para comenzar el proceso de afiliación.

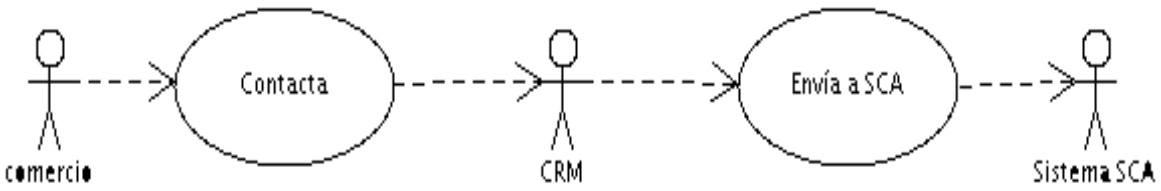

Figura 7.1 Afiliación iniciada por el comercio.

2. Un afiliador, quien se encarga de ofrecer terminales de punto de venta a domicilio para distintos comercios, recolecta la información suficiente de cada unos de ellos, de acuerdo a los intereses del comercio, para generar la solicitud de afiliación.

Dichas solicitudes serán comunicadas a través del centro de contacto del sistema SCA, al cual podrá acceder vía Internet.

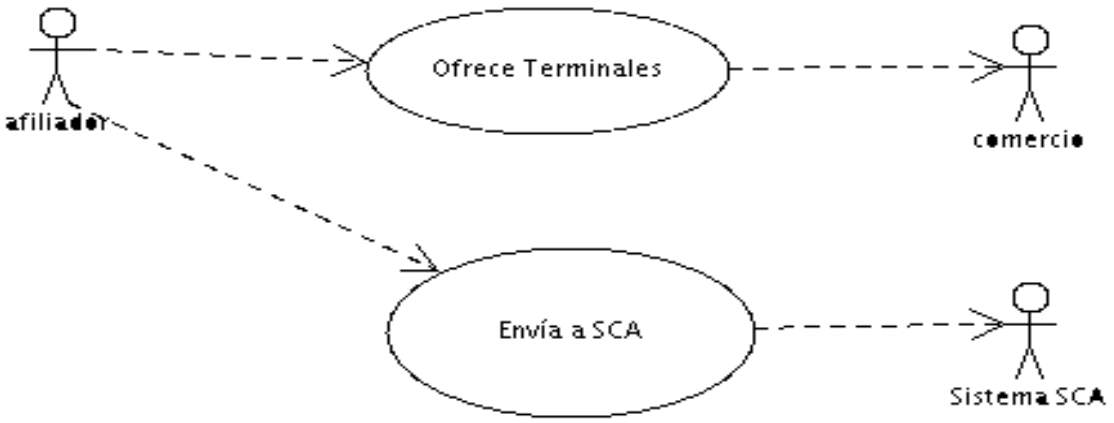

Figura 7.2 Afiliación iniciada por un afiliador.

Cada uno de estos orígenes, dará comienzo al proceso de afiliación de los comercios. Las etapas del proceso de Afiliación, que en realidad será de la comunicación de los datos entre las organizaciones, son las siguientes:

a) **Preafiliación CRM.** Etapa en la cual se incluyen todos los datos recolectados del comercio, y entre los cuales, se incluye el banco con el que se desea afiliar (ya sea por que tiene una cuenta o por que necesita que se abra con esa entidad específicamente) y el número de terminales de puntos de venta requeridos.

   CRM proporcionará un identificador bajo el nombre de ***ticketID***, el cual estará asociado a la solicitud y con el cual se controlará el resto del proceso. CRM generará una lista de solicitudes, cada una con su propio identificador, y las enviara a SCA. Además, CRM actuará como un afiliador por defecto, al cual se le asignarán las solicitudes proporcionadas, esto para llevar el control de quien genera las solicitudes.

CRM asignará el status de la solicitud como ***preafiliado*** y un substatus como ***registrado***, así como la fecha de la preafiliación y la fecha límite para la espera de respuesta del banco solicitado, para el caso en el que CRM acepte la solicitud. Por el contrario, si la solicitud no es aceptada, CRM solo asignará el status de la solicitud como ***preafiliado*** y un substatus como ***rechazado***, y no se incluirá el ticketID. Cuando SCA reciba las solicitudes, verificará el substatus de la solicitud, en caso de ser ***registrado*** cambiará el substatus de la solicitud como ***enviado a banco*** y posteriormente proporcionará las solicitudes a la entidad afiliadora (banco) elegida en cada solicitud.

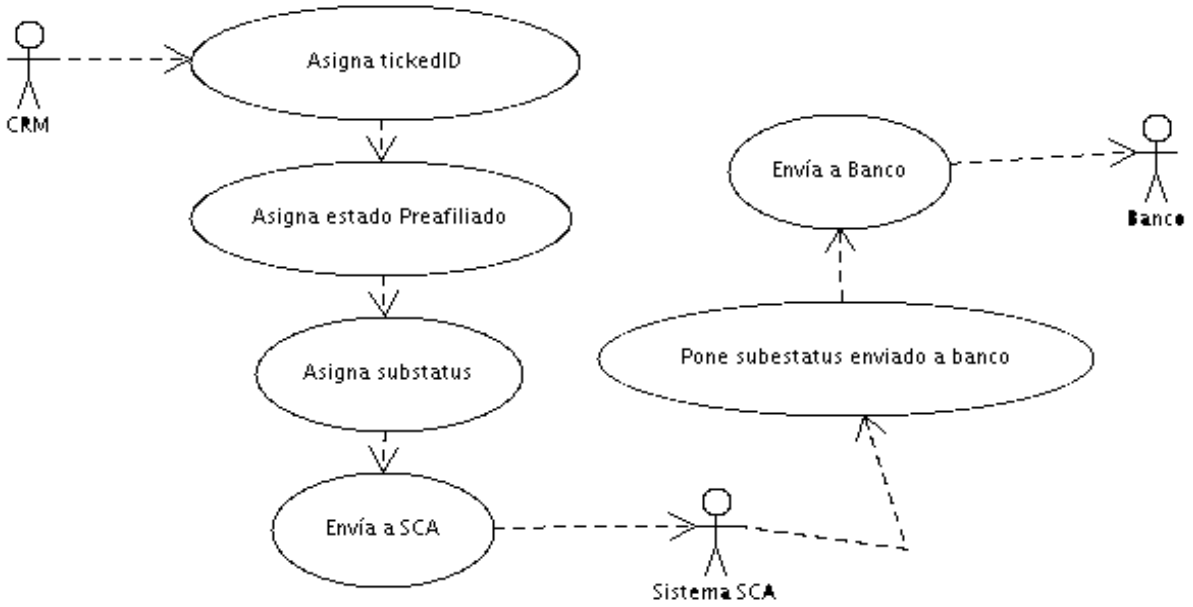

Figura 7.3 Preafiliación CRM.

Se contará con un tiempo de espera (alarma) para la respuesta del banco, si ésta se vence, SCA se encargará de avisarle a CRM de la solicitud que no ha sido atendida por el banco y los ejecutivos de CRM decidirán si otorgarle más tiempo al banco con una nueva fecha límite o proporcionar la solicitud a otro banco. SCA verificará, al recibir un documento de esta etapa, que el ticketID proporcionado exista en su banco de datos, si se da el caso (cuando CRM asigna una nueva fecha o un nuevo banco)

sólo se encargara de actualizar en su banco de datos el substatus y el banco, y reiniciará contadores para comenzar la nueva alarma.

b) **Preafiliación SCA.** En esta etapa, se incluyen los mismos datos que en la etapa de "*Preafiliación CRM*", con excepción del identificador (ticketID), ya que este sólo es proporcionado por CRM.

Se enviará la solicitud a CRM para que este le asigne un identificador, SCA solo enviará una solicitud a CRM por cada vez que se establezca una comunicación con el sistema. Se proporcionan todos los datos a CRM con el fin de que valores los mismos y decida realizar el registro de la solicitud. En esta etapa, el afiliador corresponderá al personal que proporcione la solicitud, el cual deberá estar registrado en el banco de datos de SCA. Una vez que CRM haya asignado un identificador a la solicitud, proporcionará nuevamente la solicitud a SCA como se explicó en la etapa del punto (a).

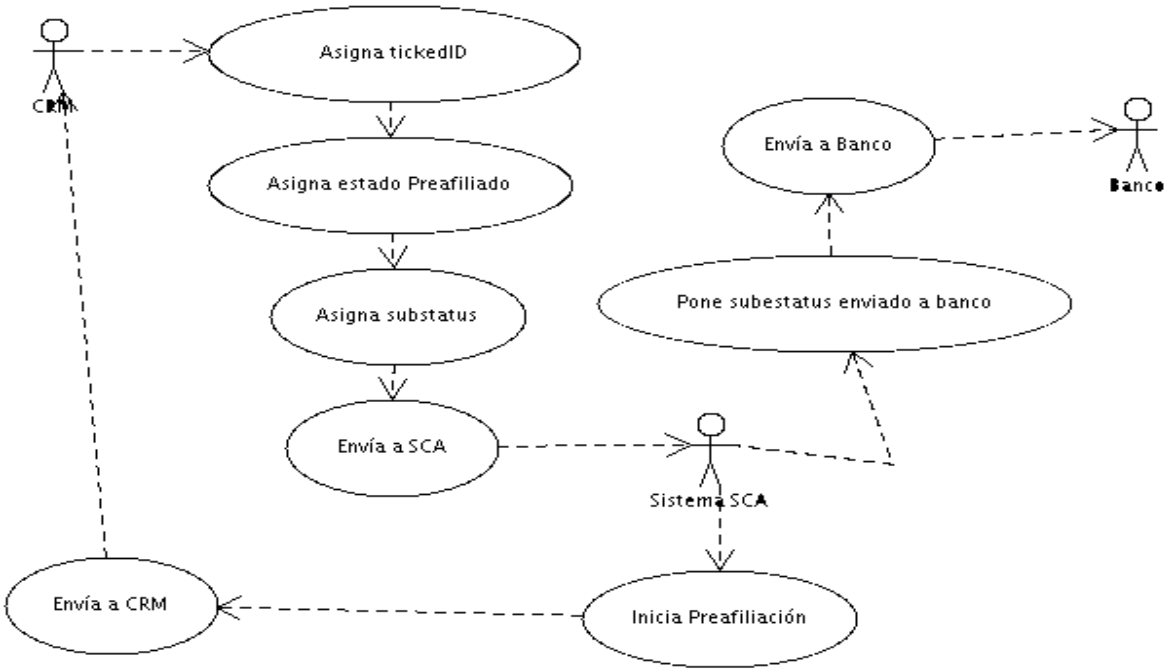

Figura 7.4 Preafiliación SCA.

c) **Respuesta Preafiliación Banco.** La preafiliación de CRM obtenida por SCA será enviada al banco quien se encargará de dar una respuesta a SCA. El banco cuenta con un plazo de tiempo (señalado en la *fecha limite)* para regresar una respuesta.

La preafiliación será evaluada por el banco y en caso de aceptar la solicitud pondrá el substatus como ***aceptado***, en caso de no aceptarla el substatus será ***rechazado***. Cuando SCA reciba la respuesta, actualizará el substatus en su banco de datos, y finalizará la alarma. Después mandará la respuesta a CRM.

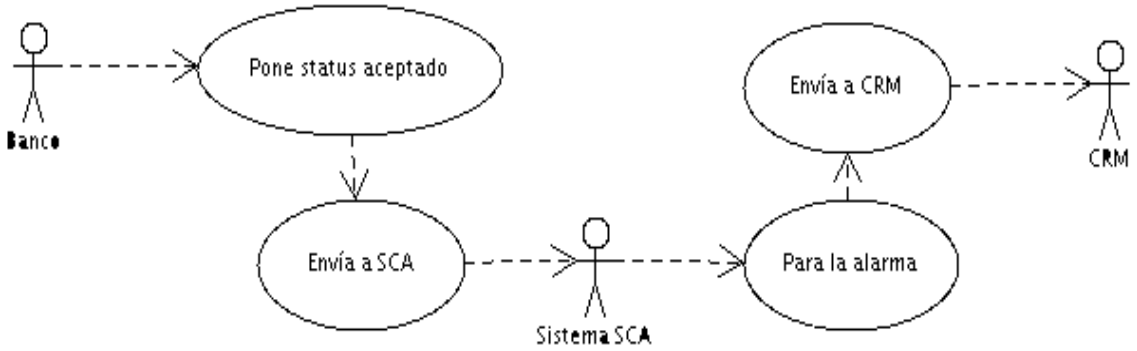

Figura 7.5 Respuesta Preafiliación Banco

d) **Afiliación Banco.** En esta etapa, nuevamente el banco, después de haber aceptado la solicitud de preafiliación de CRM enviada por SCA, proporcionará un identificador para el comercio bajo el nombre de ***merchantID.***

Además, cambiará el status de la solicitud por ***afiliado***, el substatus como ***en espera de tpvs***, y junto con el resto de los datos los enviará a SCA quien se encargará de registrar el merchantID y generar un nuevo identificador llamado ***fileID***. El identificador de archivo será utilizado para hacer referencia en el banco de datos de SCA a las solicitudes, con merchanID, almacenadas como archivo. De las solicitudes

con el identificador del banco, SCA se encargará de proporcionarle a CRM sólo el ticketID, merchantID, el fileID, el status y substatus de la afiliación.

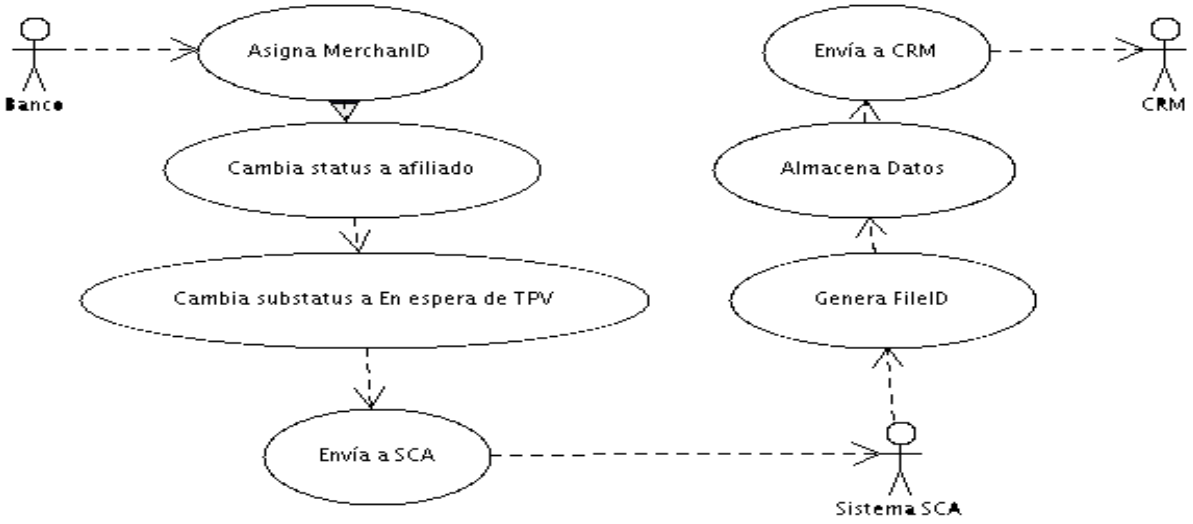

Figura 7.6 Afiliación Banco

e) **Asignación de TPV a la Afiliación.** CRM asignará una lista de identificadores y modelos para las terminales punto de venta, así como las aplicaciones que se instalarán en cada una de ellas y pondrá el substatus como ***en espera de folios de telecarga***; dicha información la proporcionará a SCA y éste al Banco respectivo encargado de cada una de las distintas afiliaciones.

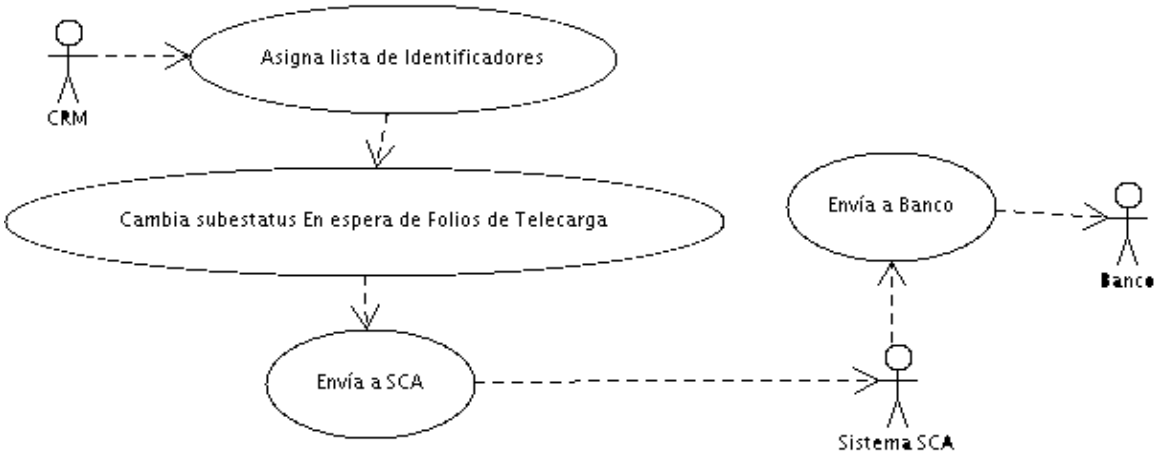

Figura 7.7 Asignación de Tpv a la Afiliación.

f) **Asignación de Folios de Telecarga a la Afiliación.** El banco, a través del modelo de la TPV proporcionado en la etapa del punto (e), creará un folio de telecarga para cada aplicación solicitada para su instalación en la terminal y cambiará el substatus por ***exitoso banco***. La información será enviada a SCA y ésta a su vez la proporcionará a CRM.

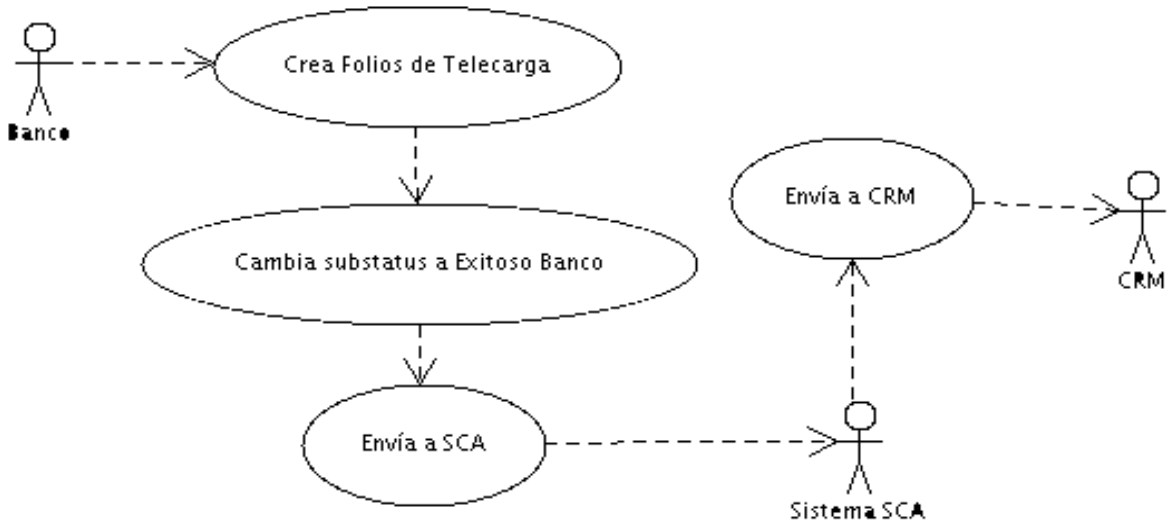

Figura 7.8 Asignación de Folios de Telecarga a la Afiliación.

g) **Instalación completa.** CRM recibe los folios de telecarga proporcionados por Banco para cada unas de las terminales, entonces procede con la instalación de las mismas en los distintos comercios y una vez que se haya concluido la instalación del número solicitado, se concluye la afiliación del comercio.

CRM se encargará de mandar a SCA una notificación de que ha concluido la instalación junto con el identificador que genera para la afiliación llamado ***afiliacionID***, además del substatus ***instalación completa***, la fecha de entrega del comercio de sus documentos impresos y la fecha que determina CRM como la de afiliación concluida, SCA registrará el nuevo identificador, y proporcionará la misma

información al banco correspondiente. Con esta etapa se concluye el proceso de la afiliación.

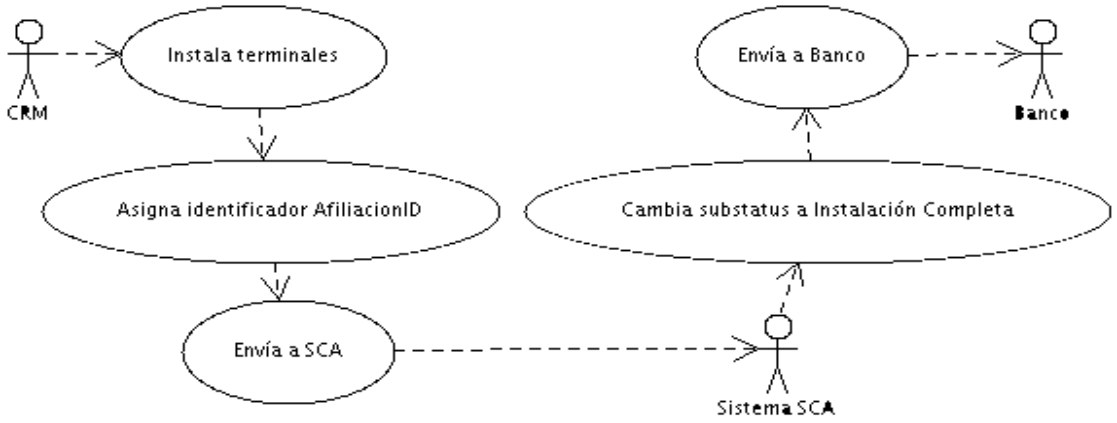

Figura 7.9 Finalización de la Instalación de Terminales.

7.2.2 Funcionalidad del sistema

7.2.2.1 Representación de la información de las Afiliaciones

La información que se compartirá y manipulará en las distintas etapas del proceso de la afiliación, se representará a través de una estructura de datos como un documento XML, de tal manera que no haya inconveniente para la identificación de la información ni para su acceso a la misma dependiendo de la tecnología optada por alguna de las organizaciones. Dicha estructura de información se diseñará a través de una DTD (ver apéndice B) para generalizar el proceso de la afiliación y posteriormente se procederá a subdividir en pequeños documentos XML para representar la estructura requerida para cada una de las etapas del proceso.

Se considera manejar una lista de solicitudes dentro de un documento XML, para minimizar el número de conexiones entre los distintos sistemas y maximizar el proceso de las afiliaciones, ya que en cada documento puede haber una variedad de solicitudes que requieran una etapa distinta del proceso. Se diferenciará cada uno de los documentos XML,

para saber a que etapa corresponde, a través del nombre del proceso, contenido en un elemento del XML.

7.2.2.2 Compartir la información de las Afiliaciones

Cada una de las organizaciones expondrá un servicio Web, y realizarán el proceso de la afiliación requerida de acuerdo al nombre del XML contenido en la estructura del documento. El cliente Web proporcionará como datos de entrada para los servicios, una cadena de caracteres, la cual contendrá la estructura XML del documento, y la cuenta de usuario del cliente Web. Como dato de salida, el servicio Web devolverá un valor entero, este indicará al ser su valor ***1*** que el documento se recibió satisfactoriamente, si el valor es ***0*** indicará que el documento no fue aceptado debido a que no cumple con la DTD establecida; y si el resultado es ***-1*** indicará que la cuenta del cliente Web no está habilitada o no es correcta.

SCA tendrá en su banco de datos las cuentas de los clientes Web de cada una las organizaciones, así como también el nombre de los datos de conexión de todos los servicios Web expuestos por cada uno de los sistemas involucrados.

SCA contará con un sólo servicio Web el cual recibirá todas las peticiones de los diferentes clientes Web; primeramente verificará que el usuario que intenta conectarse al servicio sea válido, llevará una bitácora de las conexiones fallidas y exitosas que han ocurrido, y comprobará que el documento XML cumpla con el formato convenido entre las distintas partes. Una vez realizado esto, procederá a realizar la lógica de negocio requerida para el proceso de las afiliaciones en diferentes hilos para liberar las conexiones del servidor Web.

7.2.2.3 Almacenamiento de la información de las Afiliaciones

En la lógica de negocio, se realizará la deserialización del XML en objetos POJOs de Java, con ayuda del API XStream, para acceder a los datos, de esta manera se realizaran las modificaciones pertinentes para cada etapa del proceso.

Antes de modificar cada una de las propiedades de los POJOs, se registrarán en una base de datos los datos necesarios para el control de la solicitud y de la afiliación. Debido a que los datos a almacenar no son todos los que se definieron en la DTD, la estructura de los POJOS para la persistencia diferirá de los utilizados para el parseo, por lo que se crearán nuevos POJOs con la estructura suficiente para su representación con las tablas de la base de datos mediante el mapeo con documentos XML, utilizando el API de Hibernate.

Una vez almacenado los datos elegidos por SCA, se modificarán los datos contenidos en los POJOS de Xstream y se realizará la serialización del XML para obtener una cadena de caracteres, posteriormente de acuerdo a la lógica de negocio la cual evaluará la etapa, realizará la conexión a el(los) servicio(s) Web necesarios para continuar con el proceso de la afiliación, recuperando los datos para la conexión de la base de datos.

Se efectuará un proceso similar para cada una de las etapas de la afiliación hasta que ésta haya concluido, de manera satisfactoria, o se haya cancelado, en el peor de los casos.

Conclusiones

Dado que, actualmente, la necesidad de contar con un sistema de información que ayude a manejar la gran cantidad de datos que genera una organización es cada vez mayor, y cuya necesidad no sólo se limita a un manejo local de la misma información sino que también se requiere que puedan compartirse u obtener datos generados por otra organización, es necesario que los sistemas puedan establecer una comunicación y realizar tareas automatizadas para la manipulación de los datos. Encontramos que el problema de incompatibilidad, para comunicar sistemas de información y además de compartir información entre ellos, recae en que los sistemas pueden estar desarrollados con distinto lenguaje de programación y los cuales a su vez pueden presentar limitaciones para su funcionamiento en una plataforma específica.

Por ello, para desarrollar el tema de tesis, nos enfocamos en presentar una solución que resolviera el problema de comunicar sistemas incompatibles, considerando que no sólo debería ser válido para un caso en particular sino que también debería resolver problemas similares, así que vimos que era necesario hacer uso de estándares.

Vimos que el lenguaje de marcado extensible (XML) nos permite definir estructuras de datos para un fin en particular. Dicha estructura es un estándar que nos obliga a cumplir con un formato definido por ciertas reglas, la ventaja es que las reglas no involucran a un lenguaje de programación específico, por lo que cualquier lenguaje puede cumplir con el estándar.

Con XML definimos un formato estándar para representar un conjunto de datos con una cierta estructura, el cual resulta fácil para su entendimiento, y formar documentos XML. El documento nos sirve como un medio de persistencia, y a través de éste, si cumple con las especificaciones de XML, cualquier lenguaje de programación puede acceder a los datos mediante el desarrollo de un analizador de XML bajo la misma arquitectura, o bien, se puede hacer uso de alguna API existente que permita la consulta de los datos (en algunos mas completos, permite la formación de un nuevo documento XML), además de que hay algunos que facilitan mucho la programación. Con este estándar, sólo nos preocupamos en que los

datos vayan contenidos en la estructura definida y nos olvidamos de la arquitectura del otro sistema que tenga que leer el documento.

Ahora bien, para traspasar la frontera que separa la comunicación de computadoras con distinta plataforma, sin importar la red de computadoras bajo la cual se encuentran, requerimos hacer uso de un estándar a través de un protocolo de comunicación. Con el protocolo SOAP, es posible establecer una comunicación por medio de intercambio de datos con XML, observamos que se puede realizar el empaquetado de los datos y pueden ser transmitidos a través de otro protocolo para su transporte. Por lo que con este estándar podemos hacer que dos sistemas se comuniquen sin importar en que plataforma operen.

Además, observamos que se puede implementar tecnologías ya desarrolladas, incluso se puede desarrollar nuevas tecnologías, que empleen el estándar XML de tal manera que podemos presentar un sistema con una arquitectura compleja desarrollada con una cantidad considerablemente menor de código, facilitando el mantenimiento del sistema; por ejemplo, podemos establecer una comunicación donde los sistemas realicen tareas automatizadas (el caso de un servicio Web, con el estándar WSDL) o realizar un mapeo para que haya una correspondencia con una base de datos y un lenguaje de programación (el caso de un ORM, con un estándar propietario con XML). Por lo tanto, la base de la solución que presentamos es el empleo de XML el cual, además, puede ser ocupado para otras soluciones.

Finalmente, podemos concluir que con el manejo del estándar XML y otras tecnologías de software podemos manipular la información, que procesan distintos sistemas de información, de una forma adecuada, ya que es muy valiosa para las organizaciones y es fundamental para sus operaciones y actividades, logrando superar las fronteras por incompatibilidad entre las tecnologías utilizadas por cada organización. Además de que proporcionamos un sistema de información que cumple con los requerimientos del control de las afiliaciones para las distintas organizaciones.

Bibliografía

- Martinez, Roman; Quiroga, Elda. Estructura de datos. Referencia práctica con orientación a objetos. Thomson Learning. México. 2002.
- Skonnard, Aaron; Gudgin, Martin. Essential XML Quick Reference. Addison-Wesley. E.U. 2002.
- Silverschatz, Abraham. Fundamentos de Bases de Datos. McGrawHill/Interamericana de España. España. 2006.
- S. Horstmann, Cay; Gary Cornell. Core Java 2. Séptima Edición Pretince Hall. España. 2006.
- Baver, Christian; King, Gavin. Java Persistence With Hibernate. Manning. E.U. 2006.
- Burke, Bill; Monson-Haefel, Richard. Enterprise JavaBeans 3.0. O'Reilly. E.U. 2006.

Referencias en línea:

- http://www.gestiondelconocimiento.com/conceptos_diferenciaentredato.htm
- http://unicode.org/standard/translations/spanish.html
- http://www.cl.cam.ac.uk/~mgk25/unicode.html
- http://www.monografias.com/trabajos14/datos/datos.shtml
- http://www.w3.org/XML/
- http://www.xml.com/pub/a/w3j/s3.bosak.html
- http://www.ulpgc.es/otros/tutoriales/xml/Estructura.html
- http://en.wikipedia.org/wiki/XML
- http://programarenc.webcindario.com/Cplus/capitulo1.htm
- http://java.sun.com/docs/books/tutorial/java/index.html
- http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java

- http://www.hibernate.org/
- http://xstream.codehaus.org/
- http://www.desarrolloweb.com/articulos/1557.php
- http://en.wikipedia.org/wiki/SOAP
- http://java.sun.com/webservices/
- http://en.wikipedia.org/wiki/Web_service

APÉNDICE A

A.1 Introducción a la API XStream 1.0.2

Ésta es una breve introducción de como se pueden pasar archivos XML a objetos Java y viceversa, utilizando la librería XStream. Se tomará como ejemplo el siguiente documento XML:

```
<Account>
        <FileID>GNM234345234</FileID>
        <XMLName>AFILIACION</XMLName>
         <ListOfMerchantPhoneNumbers>
                <MerchantPhoneNumber>
                        <TipoTelefono>2</TipoTelefono>
                        <Telefono>5510440090</Telefono>
                </MerchantPhoneNumber>
                <MerchantPhoneNumber>
                        <TipoTelefono>2</TipoTelefono>
                        <Telefono>5510440090</Telefono>
                </MerchantPhoneNumber>
                . . .
        </ ListOfMerchantPhoneNumbers>
</Account>
```

Listado A.1 Documento XML de Account

1. El primer paso consistirá en crear las clases necesarias para poder instanciar a un objeto que representará el archivo XML.

Para el ejemplo se necesita la clase Account y MerchantPhoneNumber descritos a continuación.

```
public class Account{
        private String FileID;
        private String XMLName;
        private List<MerchantPhoneNumber> ListOfMerchantPhoneNumbers =
                                                new  ArrayList<MerchantPhoneNumber>();
        // ... constructores y métodos
}

public class MerchantPhoneNumber{
        private int TipoTelefono;
        private int Telefono;
        // ... constructores y métodos
}
```

Listado A.2 Clases Java para el documento XML de Account

Una nota importante es que la librería XStream, no necesita los métodos getters y setters para asignar los datos al objeto, además de no tomar en cuenta la visibilidad de los atributos de las clases (private, public o protected).

2. El segundo paso consistirá en crear una instancia de la clase XStream y manipularlo de la siguiente forma:

```
public class TestXMLToAccount{
    public static void main( String[] args ) {
        // Nueva instancia de la clase XStream, utilizando el driver DomDriver
        XStream xstream = new XStream( new DomDriver() );
```

```
            // Creación de alias, para realizar un mapeo de las tags del archivo XML
            // con las clases que las representan.
            xstream.alias( "Account",  Account.class );
            xstream.alias( "ListOfMerchantPhoneNumbers", List.class);
            xstream.alias( "MerchantPhoneNumber", MerchantPhoneNumber.class );
            // Recuperacion del archivo XML mediante la clase FileInputStream
            FileInputStream xml = new FileInputStream( "ruta/archivo.xml" );
            // Obtención del objeto Account
            Account account = (Account) xstream.fromXML( xml );
        }
    }
```

Listado A.3 Clase para convertir el documento XML en un objeto Account

```
public class TestAccountToXML{
    public static void main( String[] args ){
        // Nueva instancia de la clase XStream, utilizando el driver DomDriver
        XStream xstream = new XStream( new DomDriver() );
        // Creación de alias
        xstream.alias( "Account",  Account.class );
        xstream.alias( "ListOfMerchantPhoneNumbers", List.class );
        xstream.alias( "MerchantPhoneNumber", MerchantPhoneNumber.class );
        // Creación del Objeto Account
        Account account = new Account()
        List<MerchantPhoneNumber> ListOfMerchantPhoneNumbers =
                new  ArrayList<MerchantPhoneNumber>();
        ListOfMerchantPhoneNumbers.add(  new  MerchantPhoneNumber( 1, 55123456 )  );
        account.setFileID( “abc123” );
```

```
            account.setXMLName( “AFILIACION”);
            account.setListOfMerchantPhoneNumbers( ListOfMerchantPhoneNumbers );
            // Conversión del objeto Account a una cadena XML
            String xmlString = xstream.toXML( account );
            // Visualización en consola de la cadena XML
            System.out.println( xmlString );
            }
}
```

Listado A.4 Clase para convertir un objeto Account en una cadena XML

La salida que mostrará ésta clase de prueba anterior es la siguiente:

```
<Account>
        <FileID>abc123</FileID>
        <XMLName>AFILIACION</XMLName>
        <ListOfMerchantPhoneNumbers>
                <MerchantPhoneNumber>
                        <TipoTelefono>1</TipoTelefono>
                        <Telefono>55123456<Telefono>
                </MerchantPhoneNumber>
        </ListOfMerchantPhoneNumbers>
</Account>
```

Listado A.5 Cadena XML formada a partir de un objeto Account

APÉNDICE B

B.1 Diseño del sistema

En cuanto al diseño del sistema SCA, solo nos enfocaremos en la parte expuesta en el capitulo de introducción y en la etapa de análisis, omitiremos la explicación completa de la arquitectura y el desarrollo del sistema Web.

B.1.1 Diseño de la DTD para el proceso de Afiliación

```
<!DOCTYPE ListOfAccount [

    <!ELEMENT ListOfAccount ( (Account)+ )>
    <!ELEMENT Account (
                    FileID, XMLVersion, XMLName,TipoNegocio, NombreCorporativo,
                    RazonSocial, NombreCorto, RFC, CURP, TipoPersona, FechaApertura,
                    GiroComercio, DescripcionGiro, NumeroEmpleados, ClasificacionComercio,
                    PropiedadLocal, HorarioAtencion,    SitioWeb, ListOfMerchantPhoneNumbers,
                    ListOfCommunications, ListOfBusinessAddress, ListOf Contacts, ServiceRequest )>

    <!ELEMENT FileID (#PCDATA)>
    <!ELEMENT XMLVersion (#PCDATA)>
    <!ELEMENT XMLName  (#PCDATA)>
    <!ELEMENT TipoNegocio  (#PCDATA)>
    <!ELEMENT NombreCorporativo  (#PCDATA)>
    <!ELEMENT RazonSocial  (#PCDATA)>
    <!ELEMENT NombreCorto  (#PCDATA)>
    <!ELEMENT RFC  (#PCDATA)>
    <!ELEMENT CURP  (#PCDATA)>
    <!ELEMENT TipoPersona  (#PCDATA)>
    <!ELEMENT FechaApertura  (#PCDATA)>
    <!ELEMENT GiroComercio  (#PCDATA)>
```

```
<!ELEMENT DescripcionGiro  (#PCDATA)>
<!ELEMENT NumeroEmpleados  (#PCDATA)>
<!ELEMENT ClasificacionComercio  (#PCDATA)>
<!ELEMENT PropiedadLocal  (#PCDATA)>
<!ELEMENT HorarioAtencion  (#PCDATA)>
<!ELEMENT SitioWeb (#PCDATA)>

<!ELEMENT ListOfMerchantPhoneNumbers ( (MerchantPhoneNumbers)+ )>
<!ELEMENT MerchantPhoneNumbers( TipoTelefono, DescripcionTelefono, Telefono )>
<!ELEMENT TipoTelefono (#PCDATA)>
<!ELEMENT DescripcionTelefono (#PCDATA)>
<!ELEMENT Telefono (#PCDATA)>

<!ELEMENT ListOfCommunications ( (Communications)+ )>
<!ELEMENT Communications( TipoComunicacion, DescripcionComunicaciones )>
<!ELEMENT TipoComunicacion (#PCDATA)>
<!ELEMENT DescripcionComunicaciones (#PCDATA)>

<!ELEMENT ListOfBusinessAddress ( (BusinessAddress)+ )>
<!ELEMENT BusinessAddress(
                    CodigoPostal,  Estado, DelegacionMunicipio, Colonia, Ciudad,
                    Calle, NumeroExterior, NumeroInterior, EntreCalle1, EntreCalle2,
                    PuntoReferencia )>
<!ELEMENT CodigoPostal (#PCDATA)>
<!ELEMENT Estado (#PCDATA)>
<!ELEMENT DelegacionMunicipio (#PCDATA)>
<!ELEMENT Colonia (#PCDATA)>
<!ELEMENT Ciudad (#PCDATA)>
<!ELEMENT Calle (#PCDATA)>
<!ELEMENT NumeroExterior (#PCDATA)>
<!ELEMENT NumeroInterior   (#PCDATA)>
<!ELEMENT EntreCalle1 (#PCDATA)>
```

```
<!ELEMENT EntreCalle2 (#PCDATA)>
<!ELEMENT PuntoReferencia (#PCDATA)>

<!ELEMENT ListOfContacts ( (Contact)+ )>
<!ELEMENT Contact(
                    Nombre,  ApellidoPaterno, ApellidoMaterno, Cargo,
                    DescripcionCargo, FechaNacimiento, ListOfEMails, ListOfContactPhoneNumbers )>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT ApellidoPaterno (#PCDATA)>
<!ELEMENT ApellidoMaterno (#PCDATA)>
<!ELEMENT Cargo (#PCDATA)>
<!ELEMENT DescripcionCargo (#PCDATA)>
<!ELEMENT FechaNacimiento (#PCDATA)>

<!ELEMENT ListOfEMails( (Emails)+ )>
<!ELEMENT Emails( CorreoElectronico )>
<!ELEMENT CorreoElectronico(#PCDATA)>

<!ELEMENT ListOfContactPhoneNumbers ( (ContactPhoneNumbers)+ )>
<!ELEMENT ContactPhoneNumbers( TipoTelefono, DescripcionTelefono, Telefono )>

<!ELEMENT ServiceRequest(
                    TicketID, TipoServicio, OrdenServicio, CuentaBanco, Banco, CuentaBancaria,
                    CantidadTerminales,  EstadoTicket, SubEstado, FechaPreafiliacion, FechaLimite,
                    FechaDocumentacion, Comentarios, Afiliador, Afiliacion )>
<!ELEMENT TicketID (#PCDATA)>
<!ELEMENT TipoServicio (#PCDATA)>
<!ELEMENT OrdenServicio (#PCDATA)>
<!ELEMENT CuentaBanco (#PCDATA)>
<!ELEMENT Banco (#PCDATA)>
<!ELEMENT CuentaBancaria (#PCDATA)>
<!ELEMENT CantidadTerminales (#PCDATA)>
```

```
<!ELEMENT EstadoTicket (#PCDATA)>
<!ELEMENT SubEstado (#PCDATA)>
<!ELEMENT FechaPreafiliacion (#PCDATA)>
<!ELEMENT FechaLimite (#PCDATA)>
<!ELEMENT FechaDocumentacion (#PCDATA)>
<!ELEMENT Comentarios (#PCDATA)>
<!ELEMENT Afiliador (#PCDATA)>

<!ELEMENT Afiliacion(
                MerchantID, AfiliacionID, FechaAfiliacion, ListOfTPV )>
<!ELEMENT MerchantID (#PCDATA)>
<!ELEMENT AfiliacionID (#PCDATA)>
<!ELEMENT FechaAfiliacion (#PCDATA)>

<!ELEMENT ListOfTPV ( (TPV)+ )>
<!ELEMENT TPV( ComunicacionTPV, TPVID, Proveedor, Fabricante, Modelo,  ListOfApplications )>
<!ELEMENT ComunicacionTPV (#PCDATA)>
<!ELEMENT TPVID (#PCDATA)>
<!ELEMENT Proveedor (#PCDATA)>
<!ELEMENT Fabricante (#PCDATA)>
<!ELEMENT Modelo (#PCDATA)>

<!ELEMENT ListOfApplications ( (Applications)+ )>
<!ELEMENT Applications( TipoAplicacion, FolioTelecarga )>
<!ELEMENT TipoAplicacion (#PCDATA)>
<!ELEMENT FolioTelecarga (#PCDATA)>
```

Listado B.1 DTD completo para el proceso de Afiliaciones

B.1.2 Diseño de los documentos XML para las etapas del proceso de Afiliación

B.1.2.1 Documento XML para Preafiliación de CRM a SCA

Documento para cuando CRM acepta la solicitud del comerciante.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
        <Account>
                <XMLVersion>1.0</XMLVersion>
                <XMLName>PREAFILIACION CRM </XMLName>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                <TipoNegocio></TipoNegocio>
                <NombreCorporativo></NombreCorporativo>
                <RazonSocial></RazonSocial>
                <NombreCorto></NombreCorto>
                <RFC></RFC>
                <CURP> </CURP>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                <TipoPersona></TipoPersona>
                <FechaApertura></FechaApertura>
                <GiroComercio></GiroComercio>
                <DescripcionGiro></DescripcionGiro>
                <NumeroEmpleados></NumeroEmpleados>
                <ClasificacionComercio></ClasificacionComercio>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                <PropiedadLocal></PropiedadLocal>
                <HorarioAtencion></HorarioAtencion>
                <SitioWeb></SitioWeb>
                <ListOfMerchantPhoneNumbers>
                        <MerchantPhoneNumbers>
```

```
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <TipoTelefono></TipoTelefono>
                                <DescripcionTelefono></DescripcionTelefono>
                                <Telefono></Telefono>
                        </MerchantPhoneNumbers>
                </ListOfMerchantPhoneNumbers>
                <ListOfCommunications>
                        <Communications>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <TipoComunicacion></TipoComunicacion>
                                <DescripcionComunicaciones></DescripcionComunicaciones>
                        </Communications>
                </ListOfCommunications>
                <ListOfBusinessAddress>
                        <BusinessAddress>
                                <CodigoPostal></CodigoPostal>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <Estado></Estado>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <DelegacionMunicipio></DelegacionMunicipio>
                                <Colonia></Colonia>
                                <Ciudad></Ciudad>
                                <Calle></Calle>
                                <NumeroExterior></NumeroExterior>
                                <NumeroInterior></NumeroInterior>
                                <EntreCalle1></EntreCalle1>
                                <EntreCalle2></EntreCalle2>
                                <PuntoReferencia></PuntoReferencia>
                        </BusinessAddress>
                </ListOfBusinessAddress>
<ListOfContacts>
```

```
            <Contact>
                <Nombre></Nombre>
                <ApellidoPaterno></ApellidoPaterno>
                <ApellidoMaterno></ApellidoMaterno>
                    <!--Identificador de un catalogo de la base de datos de SCA -->
                <Cargo></Cargo>
                <DescripcionCargo></DescripcionCargo>
                <FechaNacimiento></FechaNacimiento>
                <ListOfEMails>
                    <Emails>
                        <CorreoElectronico></CorreoElectronico>
                    </Emails>
                </ListOfEMails>
                <ListOfContactPhoneNumbers>
                    <ContactPhoneNumbers>
                        <!--Identificador de un catalogo de la base de datos de SCA -->
                        <TipoTelefono></TipoTelefono>
                        <DescripcionTelefono></DescripcionTelefono>
                        <Telefono></Telefono>
                    </ContactPhoneNumbers>
                </ListOfContactPhoneNumbers>
            </Contact>
        </ListOfContacts>
        <ServiceRequest>
                <!--Identificador de un catalogo de la base de datos de CRM-->
            <TicketID></TicketID>
            <TipoServicio></TipoServicio>
            <CuentaBanco></CuentaBanco>
                <!--Identificador de un catalogo de la base de datos de SCA -->
            <Banco></Banco>
            <CuentaBancaria></CuentaBancaria>
            <CantidadTerminales></CantidadTerminales>
```

```
                    <!--Identificador  de un catalogo de la base de datos de SCA -->
                <Afiliador></Afiliador>
                    <!--Identificador de un catalogo de la base de datos de SCA. 1 : PREAFILIADO-->
                <EstadoTicket>1</EstadoTicket>
                    <!--Identificador de un catalogo de la base de datos de SCA. 1 : REGISTRADO-->
                <SubEstado>1</SubEstado>
                <FechaPreafiliacion></FechaPreafiliacion>
                <FechaLimite></FechaLimite>
                <Comentarios></Comentarios>
            </ServiceRequest>
        </Account>
</ListOfAccount>
```

Listado B.2 Documento XML de Preafiliación para cuando CRM acepta la solicitud del comerciante

Documento para cuando CRM rechaza la solicitud del comerciante.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
        <Account>
            <XMLVersion>1.0</XMLVersion>
            <XMLName>PREAFILIACION CRM</XMLName>
            <ServiceRequest>
                    <!--Identificador de un catalogo de la base de datos de SCA. 1 : PREAFILIADO-->
                <EstadoTicket>1</EstadoTicket>
                    <!--Identificador de un catalogo de la base de datos de SCA. 2 : RECHAZADO -->
                <SubEstado>2</SubEstado>
                <Comentarios></Comentarios>
            </ServiceRequest>
        </Account>
</ListOfAccount>
```

Listado B.3 Documento XML de Preafiliación para cuando CRM rechaza la solicitud del comerciante

B.1.2.2 Documento XML para Preafiliación de SCA a CRM

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
        <Account>
                <XMLVersion>1.0</XMLVersion>
                <XMLName>PREAFILIACION SCA</XMLName>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                <TipoNegocio></TipoNegocio>
                <NombreCorporativo></NombreCorporativo>
                <RazonSocial></RazonSocial>
                <NombreCorto></NombreCorto>
                <RFC></RFC>
                <CURP> </CURP>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                <TipoPersona></TipoPersona>
                <FechaApertura></FechaApertura>
                <GiroComercio></GiroComercio>
                <DescripcionGiro></DescripcionGiro>
                <NumeroEmpleados></NumeroEmpleados>
                <ClasificacionComercio></ClasificacionComercio>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                <PropiedadLocal></PropiedadLocal>
                <HorarioAtencion></HorarioAtencion>
                <SitioWeb></SitioWeb>
                <ListOfMerchantPhoneNumbers>
                        <MerchantPhoneNumbers>
                                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                                <TipoTelefono></TipoTelefono>
                                <DescripcionTelefono></DescripcionTelefono>
                                <Telefono></Telefono>
                        </MerchantPhoneNumbers>
```

```
                </ListOfMerchantPhoneNumbers>
                <ListOfCommunications>
                        <Communications>
                                <!--Identificador de un catalogo de la base de datos de SCA -->
                                <TipoComunicacion></TipoComunicacion>
                                <DescripcionComunicaciones></DescripcionComunicaciones>
                        </Communications>
                </ListOfCommunications>
                <ListOfBusinessAddress>
                        <BusinessAddress>
                                <CodigoPostal></CodigoPostal>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <Estado></Estado>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <DelegacionMunicipio></DelegacionMunicipio>
                                <Colonia></Colonia>
                                <Ciudad></Ciudad>
                                <Calle></Calle>
                                <NumeroExterior></NumeroExterior>
                                <NumeroInterior></NumeroInterior>
                                <EntreCalle1></EntreCalle1>
                                <EntreCalle2></EntreCalle2>
                                <PuntoReferencia></PuntoReferencia>
                        </BusinessAddress>
                </ListOfBusinessAddress>
<ListOfContacts>
                        <Contact>
                                <Nombre></Nombre>
                                <ApellidoPaterno></ApellidoPaterno>
                                <ApellidoMaterno></ApellidoMaterno>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <Cargo></Cargo>
```

```
                                    <DescripcionCargo></DescripcionCargo>
                                    <FechaNacimiento></FechaNacimiento>
                                    <ListOfEMails>
                                        <Emails>
                                            <CorreoElectronico></CorreoElectronico>
                                        </Emails>
                                    </ListOfEMails>
                                    <ListOfContactPhoneNumbers>
                                        <ContactPhoneNumbers>
                                            <!--Identificador de un catalogo de la base de datos de SCA -->
                                            <TipoTelefono></TipoTelefono>
                                            <DescripcionTelefono></DescripcionTelefono>
                                            <Telefono></Telefono>
                                        </ContactPhoneNumbers>
                                    </ListOfContactPhoneNumbers>
                                </Contact>
                            </ListOfContacts>
                            <ServiceRequest>
                                <TipoServicio></TipoServicio>
                                <CuentaBanco></CuentaBanco>
                                    <!--Identificador de un catalogo de la base de datos de SCA -->
                                <Banco></Banco>
                                <CuentaBancaria></CuentaBancaria>
                                <CantidadTerminales></CantidadTerminales>
                                    <!--Identificador de un catalogo de la base de datos de SCA -->
                                <Afiliador></Afiliador>
                            </ServiceRequest>
                        </Account>
</ListOfAccount>
```

Listado B.4 Documento XML de Preafiliación de SCA a CRM

B.1.2.3 Documento XML para Preafiliación de SCA a Banco

Mismo documento que envía CRM, sólo cambia el substatus.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
    <Account>
        <XMLVersion>1.0</XMLVersion>
        <XMLName>PREAFILIACION CRM </XMLName>
        .
        .
        .
        <ServiceRequest>
            . . .
                <!--Identificador de un catalogo de la base de datos de SCA. 1 : PREAFILIADO-->
            <EstadoTicket>1</EstadoTicket>
                <!--Identificador de un catalogo de la base de datos de SCA. 3 : ENVIADO A BANCO--
>
            <SubEstado>3</SubEstado>
            . . .
        </ServiceRequest>
    </Account>
</ListOfAccount>
```

Listado B.5 Documento XML de Preafiliación de SCA a Banco

B.1.2.4 Documento XML para respuesta de Preafiliación de Banco a SCA

Documento para cuando Banco acepta la solicitud del comerciante.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
    <Account>
        <XMLVersion>1.0</XMLVersion>
```

```
            <XMLName>RESPUESTA PREAFILIACION BANCO</XMLName>
            <ServiceRequest>
                        <!--Identificador de un catalogo de la base de datos de SCA. 1 : PREAFILIADO-->
                  <EstadoTicket>1</EstadoTicket>
                        <!--Identificador de un catalogo de la base de datos de SCA. 4 : ACEPTADO-->
                  <SubEstado>4</SubEstado>
                  <Comentarios></Comentarios>
            </ServiceRequest>
      </Account>
</ListOfAccount>
```

Listado B.6 Documento XML de Preafiliación para cuando Banco acepta la solicitud del comerciante

Documento para cuando Banco rechaza la solicitud del comerciante.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
      <Account>
            <XMLVersion>1.0</XMLVersion>
            <XMLName>RESPUESTA PREAFILIACION BANCO</XMLName>
            <ServiceRequest>
                        <!--Identificador de un catalogo de la base de datos de SCA. 1 : PREAFILIADO-->
                  <EstadoTicket>1</EstadoTicket>
                        <!--Identificador de un catalogo de la base de datos de SCA. 2 : RECHAZADO-->
                  <SubEstado>2</SubEstado>
                  <Comentarios></Comentarios>
            </ServiceRequest>
      </Account>
</ListOfAccount>
```

Listado B.7 Documento XML de Preafiliación para cuando Banco rechaza la solicitud del comerciante

B.1.2.5 Documento XML para Afiliación de Banco a SCA

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
    <Account>
        <XMLVersion>1.0</XMLVersion>
        <XMLName>AFILIACION BANCO</XMLName>
            <!--Identificador  de un catalogo de la base de datos de SCA -->
        <TipoNegocio></TipoNegocio>
        <NombreCorporativo></NombreCorporativo>
        <RazonSocial></RazonSocial>
        <NombreCorto></NombreCorto>
        <RFC></RFC>
        <CURP> </CURP>
            <!--Identificador  de un catalogo de la base de datos de SCA -->
        <TipoPersona></TipoPersona>
        <FechaApertura></FechaApertura>
        <GiroComercio></GiroComercio>
        <DescripcionGiro></DescripcionGiro>
        <NumeroEmpleados></NumeroEmpleados>
        <ClasificacionComercio></ClasificacionComercio>
            <!--Identificador  de un catalogo de la base de datos de SCA -->
        <PropiedadLocal></PropiedadLocal>
        <HorarioAtencion></HorarioAtencion>
        <SitioWeb></SitioWeb>
        <ListOfMerchantPhoneNumbers>
            <MerchantPhoneNumbers>
                    <!--Identificador  de un catalogo de la base de datos de SCA -->
                <TipoTelefono></TipoTelefono>
                <DescripcionTelefono></DescripcionTelefono>
                <Telefono></Telefono>
            </MerchantPhoneNumbers>
```

```
                </ListOfMerchantPhoneNumbers>
                <ListOfCommunications>
                        <Communications>
                                <!--Identificador de un catalogo de la base de datos de SCA -->
                                <TipoComunicacion></TipoComunicacion>
                                <DescripcionComunicaciones></DescripcionComunicaciones>
                        </Communications>
                </ListOfCommunications>
                <ListOfBusinessAddress>
                        <BusinessAddress>
                                <CodigoPostal></CodigoPostal>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <Estado></Estado>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <DelegacionMunicipio></DelegacionMunicipio>
                                <Colonia></Colonia>
                                <Ciudad></Ciudad>
                                <Calle></Calle>
                                <NumeroExterior></NumeroExterior>
                                <NumeroInterior></NumeroInterior>
                                <EntreCalle1></EntreCalle1>
                                <EntreCalle2></EntreCalle2>
                                <PuntoReferencia></PuntoReferencia>
                        </BusinessAddress>
                </ListOfBusinessAddress>
<ListOfContacts>
                        <Contact>
                                <Nombre></Nombre>
                                <ApellidoPaterno></ApellidoPaterno>
                                <ApellidoMaterno></ApellidoMaterno>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                <Cargo></Cargo>
```

```
                    <DescripcionCargo></DescripcionCargo>
                    <FechaNacimiento></FechaNacimiento>
                    <ListOfEMails>
                        <Emails>
                            <CorreoElectronico></CorreoElectronico>
                        </Emails>
                    </ListOfEMails>
                    <ListOfContactPhoneNumbers>
                        <ContactPhoneNumbers>
                            <!--Identificador de un catalogo de la base de datos de SCA -->
                            <TipoTelefono></TipoTelefono>
                            <DescripcionTelefono></DescripcionTelefono>
                            <Telefono></Telefono>
                        </ContactPhoneNumbers>
                    </ListOfContactPhoneNumbers>
                </Contact>
            </ListOfContacts>
            <ServiceRequest>
                    <!--Identificador de un catalogo de la base de datos de CRM-->
                <TicketID></TicketID>
                <TipoServicio></TipoServicio>
                <CuentaBanco></CuentaBanco>
                    <!--Identificador de un catalogo de la base de datos de SCA -->
                <Banco></Banco>
                <CuentaBancaria></CuentaBancaria>
                <CantidadTerminales></CantidadTerminales>
                    <!--Identificador de un catalogo de la base de datos de SCA -->
                <Afiliador></Afiliador>
                    <!--Identificador de un catalogo de la base de datos de SCA. 2 : AFILIADO-->
                <EstadoTicket>2</EstadoTicket>
                <!--Identificador de un catalogo de la base de datos de SCA. 1 : EN ESPERA DE TPVS -->
                <SubEstado>1</SubEstado>
```

```
                    <FechaPreafiliacion></FechaPreafiliacion>
                    <FechaLimite></FechaLimite>
                    <Comentarios></Comentarios>
                    <Afiliacion>
                            <!--Identificador de un catalogo de la base de datos de Banco-->
                        <MerchantID></MerchantID>
                    </Afiliacion>
                </ServiceRequest>
        </Account>
</ListOfAccount>
```

Listado B.8 Documento XML de Afiliación de Banco a SCA

B.1.2.6 Documento XML para Afiliación de SCA a CRM

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
        <Account>
                <XMLVersion>1.0</XMLVersion>
                <XMLName>AFILIACION BANCO</XMLName>
                    <!--Identificador de un catalogo de la base de datos de SCA-->
                <FileID></FileID>
                <ServiceRequest>
                            <!--Identificador de un catalogo de la base de datos de CRM-->
                    <TicketID></TicketID>
                            <!--Identificador de un catalogo de la base de datos de SCA. 2 : AFILIADO-->
                    <EstadoTicket>2</EstadoTicket>
                    <!--Identificador de un catalogo de la base de datos de SCA. 1 : EN ESPERA DE TPVS-->
                    <SubEstado>1</SubEstado>
                    <Afiliacion>
                            <!--Identificador de un catalogo de la base de datos de Banco-->
                        <MerchantID></MerchantID>
                    </Afiliacion>
```

```
            </ServiceRequest>
        </Account>
</ListOfAccount>
```

Listado B.9 Documento XML de Afiliación de SCA a Banco

B.1.2.7 Documento XML para asignación de TPV a Afiliación de CRM a SCA

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
    <Account>
        <XMLVersion>1.0</XMLVersion>
        <XMLName>ASIGNACION DE TPV A AFILIACION</XMLName>
        <ServiceRequest>
            <!--Identificador  de un catalogo de la base de datos de CRM-->
            <TicketID></TicketID>
            <!--Identificador de un catalogo de la base de datos de SCA. 2 : AFILIADO-->
            <EstadoTicket>2</EstadoTicket>
    <!--Identificador de un catalogo de la base de datos de SCA. 2 : EN ESPERA DE FOLIOS DE TELECARGA -->
            <SubEstado>2</SubEstado>
            <Afiliacion>
                <!--Identificador  de un catalogo de la base de datos de Banco-->
                <MerchantID></MerchantID>
                <ListOfTPV>
                    <TPV>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                        <ComunicacionTPV></ComunicacionTPV>
                        <!--Identificador  de un catalogo de la base de datos de CRM-->
                        <TPVID></TPVID>
                        <!--Identificador  de un catalogo de la base de datos de SCA -->
                        <Proveedor></Proveedor>
```

```
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                        <Fabricante></Fabricante>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                        <Modelo></Modelo>
                                        <ListOfApplications>
                                                <Applications>
                                        <!--Identificador de un catalogo de la base de datos de SCA -->
                                                        <TipoAplicacion></TipoAplicacion>
                                                </Applications>
                                        </ListOfApplications>
                                </TPV>
                        </ListOfTPV>
                </Afiliacion>
        </ServiceRequest>
    </Account>
</ListOfAccount>
```

Listado B.10 Documento XML para asignación de TPV a afiliación

B.1.2.8 Documento XML para asignación de folios de Telecarga a Afiliación de Banco a SCA

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
    <Account>
        <XMLVersion>1.0</XMLVersion>
        <XMLName>ASIGNACION DE FOLIOS DE TELECARGA A AFILIACION</XMLName>
        <ServiceRequest>
                <!--Identificador de un catalogo de la base de datos de CRM-->
            <TicketID></TicketID>
                <!--Identificador de un catalogo de la base de datos de SCA. 2 : AFILIADO-->
            <EstadoTicket>2</EstadoTicket>
```

```
                    <!--Identificador de un catalogo de la base de datos de SCA. 3 : EXITOSO BANCO -->
                <SubEstado>3</SubEstado>
                <Afiliacion>Kasimasi
                        <!--Identificador  de un catalogo de la base de datos de Banco -->
                    <MerchantID></MerchantID>
                    <ListOfTPV>
                        <TPV>
                            <!--Identificador  de un catalogo de la base de datos de CRM -->
                            <TPVID></TPVID>
                            <ListOfApplications>
                                <Applications>
                            <!--Identificador  de un catalogo de la base de datos de Banco -->
                                    <FolioTelecarga></FolioTelecarga>
                            <!--Identificador  de un catalogo de la base de datos de SCA -->
                                    <TipoAplicacion></TipoAplicacion>
                                </Applications>
                            </ListOfApplications>
                        </TPV>
                    </ListOfTPV>
                </Afiliacion>
            </ServiceRequest>
        </Account>
</ListOfAccount>
```

Listado B.11 Documento XML para asignación de folios de Telecarga a Afiliación

B.1.2.9 Documento XML para Instalación completa de CRM a SCA

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfAccount>
        <Account>
            <XMLVersion>1.0</XMLVersion>
            <XMLName>INSTALACION COMPLETA</XMLName>
```

```
            <ServiceRequest>
                        <!--Identificador  de un catalogo de la base de datos de CRM-->
                  <TicketID></TicketID>
                        <!--Identificador de un catalogo de la base de datos de SCA. 2 : AFILIADO-->
                  <EstadoTicket>2</EstadoTicket>
                  <!--Identificador de un catalogo de la base de datos de SCA. 4 : INSTALACION COMPLETA-->
                  <SubEstado>4</SubEstado>
                  <FechaDocumentacion></FechaDocumentacion>
                  <Afiliacion>
                              <!--Identificador  de un catalogo de la base de datos de Banco-->
                        <MerchantID></MerchantID>
                              <!--Identificador  de un catalogo de la base de datos de CRM-->
                        <AfiliacionID></AfiliacionID>
                        <FechaAfiliacion></FechaAfiliacion>
                  </Afiliacion>
            </ServiceRequest>
      </Account>
</ListOfAccount>
```

Listado B.12 Documento XML para Instalación completa

B.1.3 Diseño del servicio Web para el proceso de Afiliación

B.1.3.1 Interfaz del servicio Web

```
package mx.unam.sca.ws.xml.wss.services;

public interface AfiliacionSCAWS {

      public int processRequest (String docxml, String userws, String passws) ;


}
```

Listado B.13 Interfaz del servicio Web para la Afiliación

B.1.3.2 WSDL del servicio Web

```
<?xml version="1.0"?>
        <definitions
                xmlns="http://schemas.xmlsoap.org/wsdl/"
                xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        <!--Indicamos a quién pertenece el nombre de espacio de este servicio -->
                xmlns:sca="http://www.unam.mx/SCA"
                targetNamespace="http://www.unam.mx/SCA">
        <!--El método proccessRequest() recibirá tres parámetros -->
        <message name="requestMessage">
                <part name="docxml" type="xsd:string" />
                <part name="userws" type="xsd:string" />
                <part name="passws" type="xsd:string" />
        </message>
        <!--El método proccessRequest() regresará como valor de retorno un dato numérico -->
        <message name="responseMessage">
                <part name="responseValue" type="xsd:int" />
        </message>
        <!-- Describimos la operación disponible del servicio, es decir, -->
        <!-- el método processRequest() de la interfaz AfiliacionSCAWS -->
        <portType name="AfiliacionSCAWS">
                <operation name="processRequest">
                        <input message="sca:requestMessage"/>
                        <output message="sca:responseMessage"/>
                </operation>
        </portType>
```

```
        <!-- Indicamos que los mensajes SOAP, de la operación que declaramos,
        <!-- se transmitirán por HTTP a través de llamadas RPC, enviando un mensaje
        <!-- y obteniendo una respuesta  -->
        <binding name="AfiliacionSCABinding" type="sca:AfiliacionSCAWS">
            <soap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="processRequest">
                <soap:operation soapAction="" />
                <input>
                        <soap:body use="literal"
                                namespace="http://www.unam.mx/sca"/>
                </input>
                <output>
                        <soap:body use="literal"
                                namespace="http://www.unam.mx/sca"/>
                </output>
            </operation>
        </binding>
        <!-- Indicamos como acceder al servicio  -->
        <service name="AfiliacionSCAService">
            <port name="AfiliacionSCAPort" binding="sca:AfiliacionSCABinding">
                <soap:address location="http://www.unam.mx/sca/webservices/SCAService" />
            </port>
        </service>

</definitions>
```

Listado B.14 WSDL del servicio Web para la Afiliación

B.1.4 Diseño de la base de datos

B.1.4.1 Diagrama Entidad – Relación

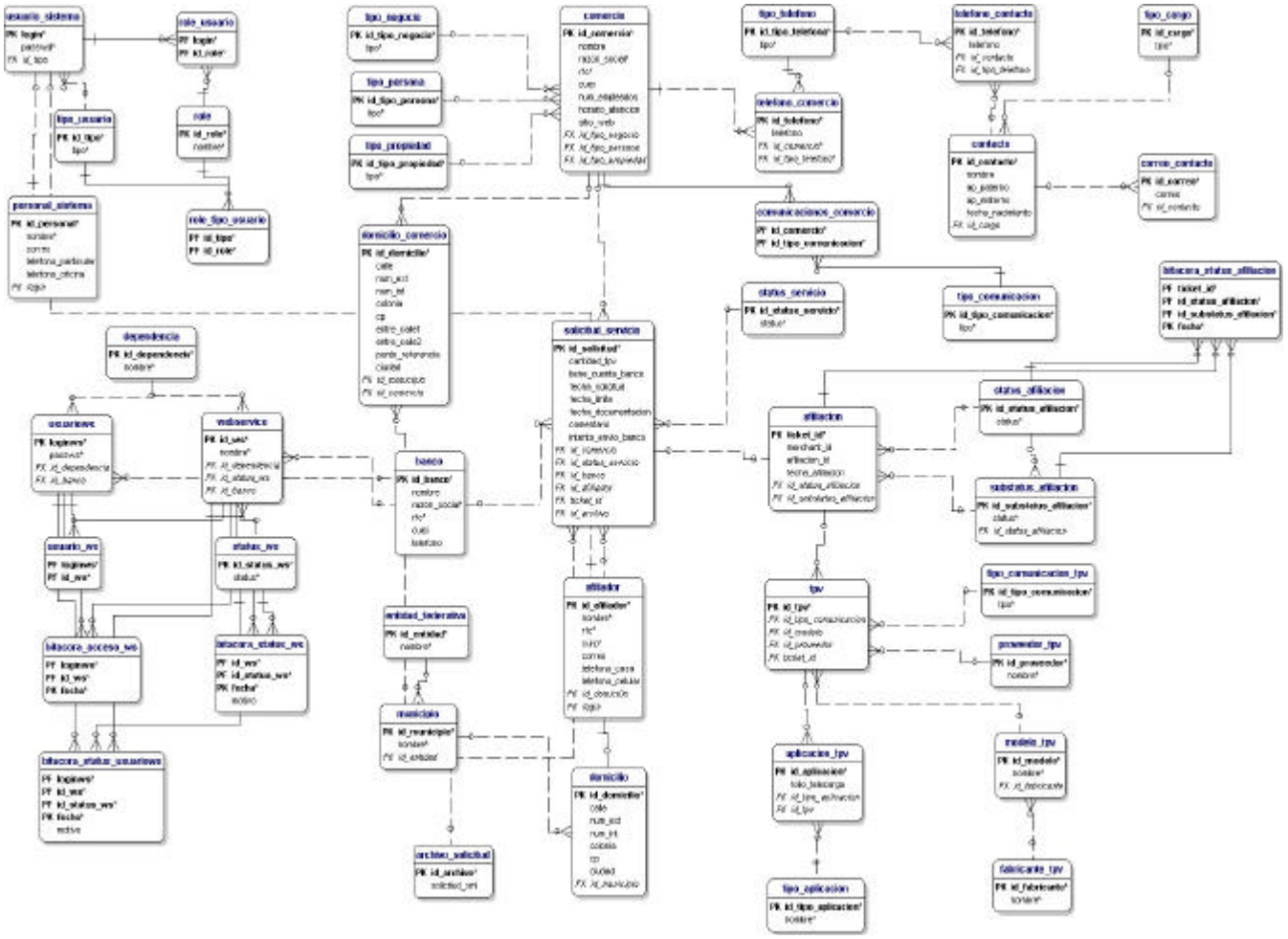

Figura 7.10 Diagrama Entidad - Relación del Sistema SCA