UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

SISTEMA DE TRANSFERENCIA DE INFORMACIÓN, BASADO EN XML Y SERVICIOS WEB, APLICADO A BURÓ DE CRÉDITO

Titulación mediante tesina y examen profesional

QUE PARA OBTENER EL TITULO DE

Licenciado en Matemáticas Aplicadas y Computación

PRESENTA

Oscar Manzo Camacho

Asesor: Licenciada Ada Ruth Cuéllar Aguayo

Fecha: Octubre 2009

Sistema de transferencia de información, basado en XML y Servicios Web, aplicado a Buró de Crédito.

Oscar Manzo Camacho

Índice

Introducción

El intercambio de información requiere de un mutuo acuerdo entre los actores que necesitan recabar los datos, sin embargo, dicho acuerdo no siempre es el más apropiado y puede requerir de una cantidad considerable de esfuerzo el cumplir con esté. Algunos de los factores que intervienen en la fluidez de esté intercambio son los estándares o formatos que se imponen a la información para ser compartida, además de la falta de coordinación entre los actores para la entrega de los datos.

Un ejemplo de esta situación se presenta en un requerimiento de Buró de Crédito hacia las Entidades Financieras del país que solicitan una contratación, requiriendo la integración de la base de datos en el estándar técnico INTF para Personas Físicas y en Cinta para Personas Morales. El presente trabajo se enfocará a la satisfacción de la integración de los datos para Personas Físicas.

Se propone adoptar las especificaciones de XML para la generación de estructuras de datos que faciliten la adaptación de la información crediticia al esquema necesario para integrarla a la base de datos de Buró de Crédito. Siendo XML un estándar ampliamente aceptado por diversas herramientas y servicios por su flexibilidad, independencia de plataforma y gran capacidad de almacenamiento.

Los primeros capítulos del presente trabajo se concentran en la introducción y el manejo de documentos bajo el estándar XML, resaltando sus fundamentos y características, además de mencionar las tecnologías principales para su procesamiento, basadas en el lenguaje Java, siendo esté una de las principales plataformas de desarrollo para aplicaciones empresariales. Ya que el requisito indispensable es la integración de la información, se contempla una breve descripción de las similitudes y estrategias que existen para adaptar los documentos XML a bases de datos relacionales y viceversa.

Con el propósito de agilizar el proceso y dar cumplimiento a los requerimientos en tiempo y forma, se contempla la automatización del envío incorporando un protocolo de transferencia de archivos (FTP) y realizando la notificación de cumplimiento mediante Servicios Web, que pondrá a disposición la aplicación de servidor de Buró de Crédito dando un valor agregado al sistema de transferencia.

La parte final del trabajo integra las herramientas y tecnologías mencionadas a través de los capítulos para dar forma al *Sistema de transferencia de información, basado en XML y Servicios Web, aplicado a Buró de Crédito*, donde se describen los componentes desarrollados para el proceso de extracción y validación de los datos, así como la coordinación que tendrán los sistemas de las entidades financieras con los servicios web, concluyendo con el proceso de poblado de datos a la base de Buró de Crédito.

A continuación se da una breve descripción de los capítulos que componen el proyecto:

- Fundamentos de XML: Se da una introducción a las bases y fundamentos de XML, destacando sus características principales. Se adentrara en la sintaxis de XML

cómo lenguaje de marcado, mostrando los diversos tipos de elementos que contiene y las estructuras que éstos conforman.

- Procesamiento de XML con Java: Se tratan las dos principales herramientas proporcionadas en Java para el procesamiento de documentos XML, las cuales son SAX (*Simple API for XML*) y DOM (*Document Object Model*).

- XML y Bases de Datos: Se muestra la relación que hay entre la estructura de un documento XML y una base de datos relacional, denotando las clasificaciones de documentos XML según la estructura de sus elementos. Se contemplarán las estrategias de almacenamiento de la información, adaptadas al tipo de documento procesado.

- Servicios Web: Se da una introducción a las bases y conceptos de un Servicio Web, resaltando la importancia que tiene XML dentro de los mismos, siendo que un Servicio Web esta formado por un conjunto de tecnologías que tienen a XML cómo factor común.

- Sistema de Transferencia de información crediticia, requerida por Buró de Crédito, basada en XML y Servicios Web: Se verán entrelazadas las tecnologías mencionadas, mediante el desarrollo del sistema mencionado que cumpla con el objetivo de agilizar los procesos de Buró de Crédito.

Capítulo 1

Fundamentos de XML

En los años setenta se perseguía la idea de estructurar los documentos de forma organizada, con el fin de facilitar el intercambio y manipulación de datos. IBM creó GML (Lenguaje de Marcado Generalizado) para satisfacer las necesidades de sus sistemas internos de edición. Se empleo GML para producir libros, informes y otros documentos a partir de un sólo archivo fuente.

La primera tecnología de la información estandarizada y estructurada de cierta importancia fue SGML (Lenguaje de Marcado Generalizado Estándar) procedente de igual forma de IBM. Posteriormente, se expandió y adapto para ser utilizado en diversos sectores cómo estándar de información de propósito general. No fue hasta 1986 que SGML emergió cómo estándar ISO, sin embargo, a pesar de que SGML es muy potente, resulta muy complejo y es necesaria una gran cantidad de software para su procesamiento. En 1989, Tim Berners Lee y Anders Berglund, dos investigadores del Laboratorio Europeo de Física de Partículas, crearon un lenguaje basado en etiquetas para marcar documentos técnicos a fin de poder distribuirlos en Internet. Este lenguaje fue finalmente ampliado en una aplicación simplificada de SGML al cual llamaron HTML, que supuso el primer formato de información estándar en la Web. [1, pp. 3]

HTML fue originalmente diseñado cómo forma de presentar información estática sólo para fines de despliegue, el hecho de agregar interactividad esperando que representara información dinámica, generó un cambio a su propia naturaleza. Aún más importante que este cambio, fue el hecho de que se esperaba que HTML sirviera para almacenar tipos específicos de datos cómo transacciones financieras, catálogos de productos y reportes. HTML estaba cambiando de lenguaje de información basada en presentaciones a una tecnología de software, la limitación más importante de HTML es su conjunto fijo de etiquetas, es imposible añadir etiquetas personalizadas a HTML, lo cual sería de mucha utilidad al tratar con datos pertenecientes a una aplicación o sector específico.

Fue en febrero de 1998 cuando la W3C (World Wide Web Consortium) lanzó la especificación XML 1.0 (Extensible Markup Language), el cual es un subconjunto simplificado de SGML que incorpora muchas de sus características en las que se incluyen las tres más importantes: extensibilidad, estructura y validación. XML es una

especificación que sirve para diseñar lenguajes de marcado, es decir XML es un metalenguaje[1]. [1, pp. 6]

1. Fundamentos de XML

XML (Extensible Markup Language) es un metalenguaje que se usa para describir lenguajes de etiquetas, las cuales proporcionan información acerca del contenido para el que se está creando la estructura de documento. [2, pp.8-9]

Con su soporte para vocabularios de etiquetas personalizados, XML brinda grandes ventajas a la descripción de datos y las relaciones que hay entre las distintas piezas de éstos. Al concentrarse en el contenido, XML presenta una forma de incluir metadatos en los documentos.

Para la creación de documentos XML válidos, es necesario identificar los tipos de datos, el contenido que se va a incluir en el documento y proporcionar una pieza de estructura que rodea a cada pieza de información identificada en el documento.

- Procesador XML*:* Un procesador XML es un módulo de software que lee un documento XML y que proporciona acceso a su contenido y estructura. Las aplicaciones XML utilizan los procesadores para acceder al documento lo que puede confundir la distinción entre la aplicación y el procesador. Sin embargo, esta distinción es muy importante para los desarrolladores de aplicaciones cuando se requiere integrar la funcionalidad de componentes de software que se puedan integrar en una aplicación para procesar un documento XML. [1, pp.8]

- Extensibilidad*:* XML hace una gran función al describir datos estructurados cómo texto, y el formato está abierto para ser extendido. Esto significa que cualquier dato puede ser descrito en texto y así puede ser anidado en etiquetas. Las extensiones del lenguaje sólo necesitan seguir la sintaxis básica del XML, los limites de los datos son impuestos por lo propios datos, mediante reglas de sintaxis y directivas de validación. [5, pp.5]

- Estructura*:* La estructura de un documento XML es usualmente compleja a simple vista, pero lectores de XML y otras herramientas están diseñadas para trabajar con este tipo de documentos aún en sus más complejas formas. Además XML fue diseñado para ser un formato abierto de intercambio de datos, por lo tanto la representación de datos es usualmente muy extensa que el formato original de la información. [5, pp.5]

El bloque de construcción básico de un documento XML es la entidad, que contiene datos analizados y no analizados sintácticamente, los datos analizados sintácticamente están compuestos por datos o marcas de caracteres que son procesados mediante un procesador XML. Los Datos no analizados sintácticamente se manejan cómo texto y no serán procesados. [1, pp.8]

Componentes de un Elemento	Estructura de una Etiqueta

[1] Lenguaje usado para hacer referencia a otros lenguajes

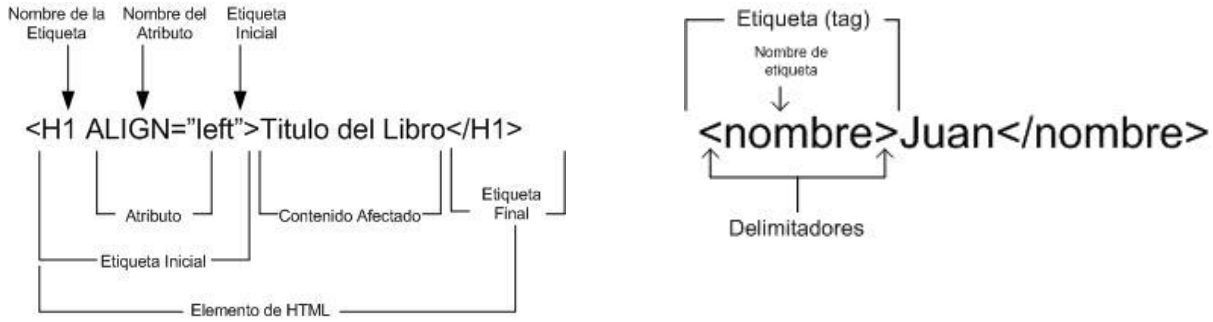


Figura 1.1 - Estructura de elemento y etiqueta

XML permite imponer restricciones al diseño y estructura a un documento, especificando las relaciones que hay entre componentes de etiquetado. La sintaxis XML describe esencialmente las construcciones empleadas para definir la estructura y diseño de los documentos XML. [1, pp.8]

Los caracteres que se encuentran entre las etiquetas de marcado representan el contenido de la información real de un documento XML, mientras el marcado restante especifica la estructura del mismo.

Las entidades describen la estructura física de un documento y la estructura lógica está determinada por los elementos que hay en el documento junto con las relaciones que hay entre tales elementos, una analogía de la estructura física-lógica de un documento XML es una base de datos relacional que emplea tablas y campos para la estructura lógica y generalmente archivos binarios para la estructura física. [1, pp.9]

1.1. Estructura física: Entidades

Un documento XML puede consistir en una o más unidades de almacenamiento virtual, llamadas entidades. Cada documento comienza con una entidad “documento”, también llamada raíz, que sirve como punto de inicio para el procesador XML y puede contener el documento completo.

Todas las entidades tienen contenido y todas ellas están identificadas por un nombre. [15, estructura física]

Las entidades son bloques de construcción de documentos XML, suelen estar formados por otras entidades a través de referencias que se utilizan para asignar alias a piezas de datos. [1, pp.11]

El signo *&* (ampersand) seguido por un nombre, y éste a su vez seguido de un punto y coma se utilizan para representar entidades tales cómo referencias hacia imágenes gráficas o caracteres especiales por ejemplo, „ñ’ para la „ñ’.

```
<idioma>Espa&ntilde;ol</idioma>
```

1.2. Estructura lógica

Un documento XML está compuesto de declaraciones, elementos, comentarios, referencias a caracteres o entidades e instrucciones de procesamiento, los cuales están indicados por una marca explícita.

Los documentos se pueden dividir en componentes (capítulos y artículos) que a su vez se componen de títulos, párrafos, figuras, etc. dando así una estructura.

Cada componente lógico del documento es representando por un elemento que tiene un tipo, identificado por un nombre, denominado identificador genérico, y puede tener un conjunto de especificaciones de atributos, que describen las propiedades de dicho elementos.

Los límites de los elementos están delimitados por etiquetas de comienzo y de final o, en el caso de elementos vacíos, por una etiqueta de elemento vacío.

La visión jerárquica del documento nos habla de una estructura en árbol del mismo. El elemento que contiene a los demás se le conoce cómo elemento raíz o elemento de documento porque indica que es el único que no depende de otro. [15, estructura lógica]

```
<libro>
  <titulo>Procesamiento de XML</titulo>
  <capitulo numero='1'>
     <cap_titulo>XML<cap_titulo>
       <parrafo>
```

El lenguaje de marcado extensible XML es un metalenguaje que permite a los usuarios diseñar un propio lenguaje de etiquetas que definirán la manera abstracta de describir o presentar la información en una determinada clase de documento. La información en un documento XML generalmente está estructurada en forma de árbol

```
      </parrafo>
    ...
  </capitulo>
  ...
</libro>
```

Código 1.1 - Estructura de Documento XML

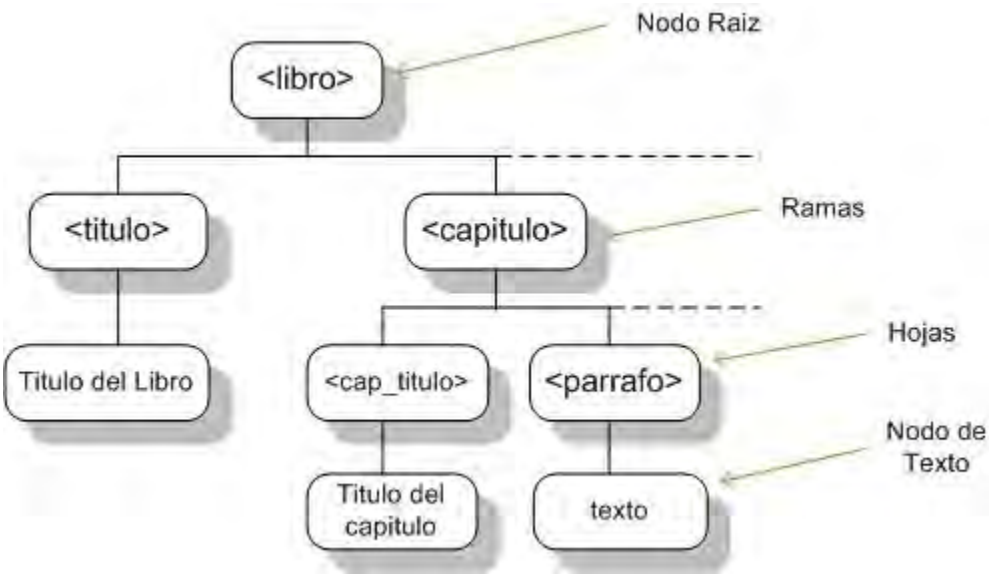


Figura 1.2 - Estructura de Árbol

1.3. Documentos XML válidos y bien formados

A causa de los requerimientos de sintaxis que forman a un documento XML, los datos representados cómo XML pueden opcionalmente ser validados por estructura y contenido basado en dos estándares de validación de datos. El estándar original para validación de datos XML es DTD (Document Type Definition) y la tecnología más reciente XML Schemas. [5, pp.6]

1.3.1. Documentos XML bien formados

Básicamente se dice que un documento es bien formado si cuenta con una estructura legible y si cumple todos los criterios identificados en las especificaciones de XML, cómo lo es:

- La declaración del elemento `<?xml>` que es el medio por la que el software identifica un documento XML, situando esta etiqueta siempre en la primera línea del documento, la cual indica la versión de XML utilizado, la dependencia con fuentes externas al documento y la codificación del conjunto de caracteres contenidos, por medio de atributos definidos. [2, pp.55]

- Debe de contar con un elemento raíz que contenga el resto de elementos del documento XML, además de que cada etiqueta de abertura le corresponda una de cerradura: <nombre>Juan </nombre>

- Que las etiquetas estén balanceadas, y que se indique si una etiqueta es vacía `<separados/>` en caso de no contar con su correspondiente cerradura. [2, pp.6]

- Los atributos deben de estar contenidos en las etiquetas comenzando por el nombre del atributo, seguido por un signo de igual y el valor entre comillas `<nombre edad=”25”> Juan </nombre>` [2, pp.7]:

1.3.2. Documentos XML válidos

Para que un documento XML se considere cómo válido, debe de estar bien formado y su declaración debe de estar apegada a una DTD o XML Schema que imponga las reglas de sintaxis de dicho lenguaje.

Para comprobar si un documento XML es válido, es necesaria la utilización de software especializado, conocido como analizadores sintácticos del cual hay dos tipos:

- Analizadores sintácticos no validadores
  Los analizadores sintácticos no validadores comprueban la buena formación dentro de un documento XML, no garantizan que todas las etiquetas de XML sean correctas, puesto que no conocen cuales lo son, además de verificar el balance de las etiquetas y la indicación de etiquetas vacías. [2, pp.25-26]

- Analizadores sintácticos validadores
  Los analizadores sintácticos validadores, además de comprobar la buena formación del documento, comprueban que el documento se ajuste al DTD o XML Schema, estos analizadores toman cada una de las entradas del documento XML y la comparan con el esquema vinculado. Cuando el analizador encuentra un error en el código, indica el tipo de error y la línea donde se ha detectado. [2, pp.28-30]

1.4. Elementos XML

Los elementos de un documento XML son componentes que se identifican por medio de etiquetas o marcado. La etiqueta inicial <nombre> marca el inicio de un elemento y la etiqueta final </nombre> el final de ese elemento. Cada uno de los elementos describe el papel lógico de la información dentro del documento.

```
<titulo>Procesamiento de XML</titulo>
```

En ocasiones es necesario contar con elementos XML vacíos lo cual significa que no contiene datos de caracteres analizados sintácticamente. Por ejemplo, el elemento `<separador/>` puede representar algún grafico para delimitar secciones de un documento.

Como se aprecia en la estructura de un documento XML, los elementos pueden contener caracteres de texto u otros elementos que a la vez pueden contener texto, elementos o ambos, dando la estructura de árbol.

```
<libro edicion=”2”>
    <titulo> Procesamiento de XML </titulo>
    <autor> Heather Williamson </autor>
</libro>
```

1.5. Composición de un documento DTD

Un documento DTD (Definición de Tipo de Documento, Document Type Definition) proporciona las reglas de sintaxis a las cuales debe de apegarse el lenguaje que lo incorpore. En el DTD se especifican los elementos que conforman un lenguaje y el orden de aparición dentro del documento XML, también indica las jerarquías de anidamiento entre elementos, así como la frecuencia de aparición y la lista de atributos correspondientes a cada elemento.

Las aplicaciones para procesamiento de XML tienen que emplear los documentos DTD para saber qué etiquetas deben reconocer y qué normas o reglas se deben usar para el marcado de dichos documentos. Además en el DTD se indican los conjuntos de subelementos contenidos en elementos y las listas de atributos de dichos elementos, dando forma a la sintaxis a seguir del lenguaje.

Para identificación de elementos dentro de una DTD se utiliza la sintaxis:

```
<!ELEMENT nombre_elemento contenido>
```

Donde ELEMENT es la palabra reservada para la representación de elementos, a continuación le sigue el nombre del elemento referenciado y finalmente se indica el contenido de dicho elemento.

Dependiendo del contenido, el elemento puede pertenecer a alguno de estos tipos [1, pp.41]:

Tipo de Contenido	Descripción	Ejemplo
EMPTY	Especifica que este elemento puede no tener contenido	<!ELEMENT imagen EMPTY>
ANY	Especifica que el elemento puede tener cualquier contenido, tanto texto, subelementos o ambos.	<!ELEMENT directorio ANY>
Contenido Mixto	Permite especificar el contenido exacto que se desee para el elemento.	<!ELEMENT directorio (nombre | alias | #PCDATA)>
Secundario (Children)	Especifica que elementos secundarios se pueden encontrar en el cuerpo del elemento identificado.	<!ELEMENT contacto (nombre, calle, ciudad)>

Tabla 1.1 - Contenido de elementos de DTD [2, pp.100]

El contenido de un elemento se define mediante un modelo que se especifica por medio de la combinación de símbolos especiales, nombres de elementos y de elementos secundarios. Los símbolos describen la relación que hay entre los elementos secundarios y el elemento contenedor. La tabla 1.2 muestra las reglas para la formación de un modelo de contenido.

Marcado	Definición	Ejemplo
(…)	Delimita un grupo	<!ELEMENT contacto (hogar | celular)>
A | B	Ocurre A o B pero no ambos	
A , B	Ocurren A y B en el mismo orden	<!ELEMENT contacto (nombre, calle, ciudad, estado, cp, #PCDATA)>
A & B	Ocurren A y B sin importar el orden	<!ELEMENT contacto (nombre & apellido)>
A?	A puede ocurrir cero o una vez.	<!ELEMENT contacto (nombre?)>
A*	A puede ocurrir cero o mas veces.	<!ELEMENT contacto (calle*)>
A+	A puede ocurrir una o mas veces.	<!ELEMENT directorio (contacto+)>

Tabla 1.2 - Reglas de modelo de contenido de una DTD. [2, pp. 104]

De igual manera que los elementos, los atributos tiene una sintaxis especifica para su especificación dentro de una DTD.

```
<!ATTLIST  nombre_elemento nombre_atributo tipo_atributo valor_predeterminado>
```

La palabra reservada ATTLIST indica al procesador que se trata de un atributo en lugar de un elemento o una entidad. El `nombre_elemento` indica el elemento declarado previamente que contiene dichos atributos, `nombre_atributo` es el nombre del atributo, `tipo_atributo`

indica el tipo al que pertenece el atributo (#REQUIRED, #IMPLIED, #FIXED)[2], también es posible indicar algún valor por default para el atributo por medio de valor_predeterminado. [2, pp.131-132]

Para obtener información más completa y ejemplos de la implementación de los diversos tipos de atributos, dirigirse a la referencia [2, 131-137]

1.6. Definición de XML Schemas

Los XML Schemas son una recomendación de W3C para describir el contenido de un documento XML. El XML Schema utiliza el núcleo de XML para generar un lenguaje de definición de documentos extremadamente potente y flexible, que puede proporcionar control no solamente sobre la existencia de elementos y atributos, su contenido y orden de aparición cómo las DTD, sino también sobre tipos de datos específicos, cuándo y cómo utilizar los elementos y atributos además de sus contenidos, según la jerarquía de aparición dentro del documento.

Los esquemas se utilizan para comprobar la información de un documento después de realizar un análisis sintáctico, expandir las entidades y cargar los valores predeterminados de los atributos. [2, pp. 403-404]

De acuerdo con el W3C, un esquema es un conjunto de reglas que sirve para forzar la estructura y articulación del conjunto de documentos XML.

El propósito de un XML Schema es definir un bloque de construcción legal de un documento XML [20]:

- define elementos que pueden aparecer en el documento
- define atributos que pueden encontrarse en el documento
- define los elementos que son hijo o están contenidos en otros elementos
- define el orden de los elementos hijos
- define el número de los elementos hijos
- define si un elemento es vació o puede incluir texto
- define los tipo de datos de elementos y atributos
- define los valores fijos y determinados de elementos y atributos

Estas son algunas de las ventajas de los XML Schemas sobre las DTD [20]:

- XML Schemas son extensibles para futuras adiciones
- XML Schemas son las ricos y útiles que las DTD
- XML Schemas están escritos en XML
- XML Schemas soportan tipos de datos
- XML Schemas soportan Espacios de Nombres

[2] Requerido (required), opcional (implied) y predeterminado (fixed), pueden consultarse a más detalle en la referencia [2, pp. 131]

DTD **XML Schema**

```
<?xml version="1.0"?>
<nota>
<para>Tove</para>
<de>Jani</de>
<cabecera>Reminder</cabecera>
<contenido>Don't forget me this weekend!</contenido>
</nota>
```

```
<!-- DTD del Documento XML -->

<!ELEMENT note (origen, destino,
cabecera, contenido)>
<!ELEMENT origen (#PCDATA)>
<!ELEMENT destino (#PCDATA)>
<!ELEMENT cabecera (#PCDATA)>
<!ELEMENT contenido (#PCDATA)>
```

```
<!—XML Squema del Documento -->

<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.w3schools.com"
                xmlns="http://www.w3schools.com"
                elementFormDefault="qualified">
  <xs:element name="nota">
    <xs:complexType>
  <xs:sequence>
<xs:element name="origen" type="xs:string"/>
<xs:element name="destino" type="xs:string"/>
<xs:element name="cabecera" type="xs:string"/>
<xs:element name="contenido"
            type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código 1.2 - Comparación entre DTD y Schema XML

A simple vista, se destaca la estructura para declaración de elementos dentro de un XML Schema en contraste con la localizada en una DTD del mismo documento XML. Los XML Schemas son más complejos y la cantidad de código es mayor, sin embargo permiten una descripción de los elementos más robustos y extensibles.

La definición del lenguaje para un XML Schema es extensa y compleja, puede consultar la referencia [2, cap. 18] para una comprensión y verificación de los elementos que los componen.

Capítulo 2

Procesamiento de XML con Java

Si dos aplicaciones necesitan intercambiar datos mediante XML, una aplicación puede transmitir el documento hacia la aplicación destino usando algún protocolo cómo FTP, HTTP, NFS, discos extraíbles o cualquier otro medio para movimiento de datos entre sistemas, ya que XML es completamente neutral al protocolo en que se transporte.

Actualmente las empresas mantienen una gran cantidad de datos críticos de negocios permanentemente en formato XML. La mayoría de los proveedores de bases de datos relacionales ofrecen capacidades XML en sus productos. Sin embargo ninguno de los sistemas relacionales XML proveen las mismas características de rendimiento cómo el procesamiento de datos relacionales. Esto es por que el procesamiento de XML requiere la lectura de documentos XML el cual es muy intensivo para el CPU.

El *parseo* es el proceso de leer un documento XML y reportar su contenido a una aplicación cliente mientras se checa la buena formación del documento. [4, pp.260]

A través del documento se hará referencia a los componentes para el parseo de documentos XML como "*lector XML" (parser[3])*.

Para poder procesar un documento XML adecuadamente, uno de los requerimientos mínimos es que el documento XML este bien formado, en caso contrario el lector XML no podrá procesarlo y notificarlo. Cuando un lector XML encuentra un documento mal formado, detiene su lectura y reporta el error.

[3] Parser es una librería de software que consta de un conjunto de clases que lee y verifica la buena formación de un documento XML

2.1. SAX

SAX (Simple API for XML) implementa un modelo de eventos para la lectura de documentos XML, conforme va leyendo el contenido, al encontrar el inicio de documento o algún elemento dispara el evento correspondiente y se invoca a un método determinado. Por ejemplo, al localizar la etiqueta del elemento raíz que indica el inicio del contenido se invoca el método *startDocument()* y al encontrar la etiqueta de inicio de algún elemento se invoca el método *startElement()* y así sucesivamente hasta encontrar el fin del documento o en caso de algún error lanza una *notificación* y termina el proceso.

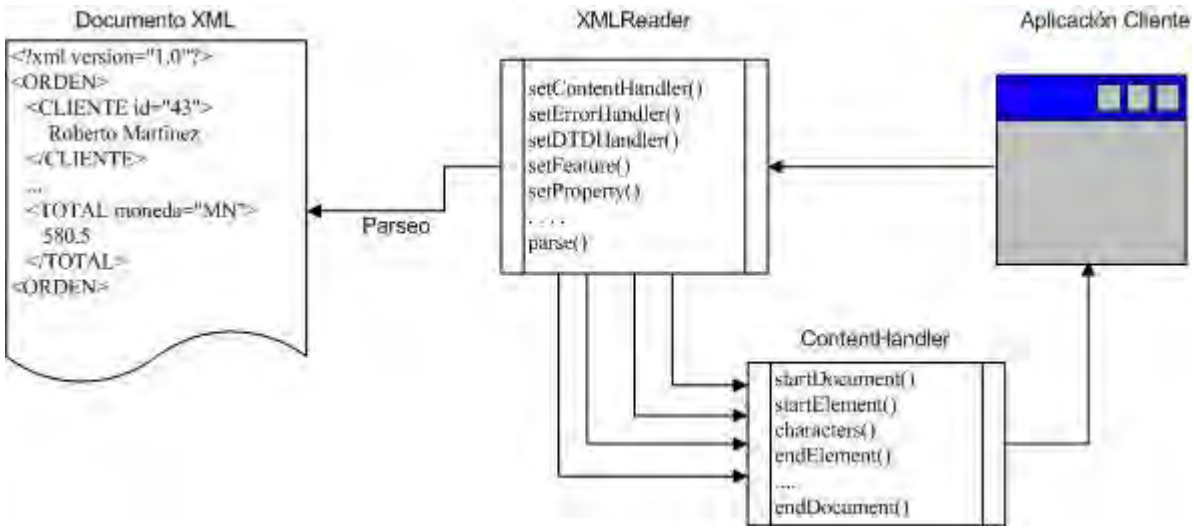


Figura 2.1 - Modelo de lectura con SAX

2.1.1. Parseo con SAX

SAX representa a un lector cómo una instancia de la interfase `org.xml.sax.XMLReader`, para obtener dicha instancia se cuenta con el método estático createXMLReader() de la clase `org.xml.sax.helpers.XMLReaderFactory`. Una vez obtenido el objeto XMLReader a través del método parse() se indica el documento XML a procesar.

```
try{
    XMLReader parser = XMLReaderFactory.createXMLReader();
    parser.parse(uri);
}catch(SAXException ex) {
  ex.printStackTrace();
}
 . . . .
```

Código 2.1 - Obtención del lector basado en SAX

De igual manera el paquete `javax.xml.parsers` provee de clases que permiten el procesamiento de documentos XML, por ejemplo la clase `SAXParser` envuelve la implementación de `XMLReader`.

2.1.2. Patrón oyente

SAX utiliza el patrón de diseño Observador (Observer) el cual es similar a la arquitectura de eventos de AWT y Swing. En la cual se implementa una interfase oyente (Listener) para recibir los eventos lanzados, después este oyente es registrado a un componente, cómo Button en el caso de AWT, mediante un método predefinido, como consecuencia, cuando el Button cacha un evento determinado, invoca un método del objeto auditor registrado. En este ejemplo Button juega el rol de Sujeto, la interfase auditora el rol de observador (observer), y la implementación definida por el cliente en la interfase el rol de observador concreto (concrete-observer). [4, pp.264]

SAX trabaja de una manera muy similar, excepto que la instancia de XMLReader juega el rol del sujeto y la interfase `org.xml.sax.ContentHandler` juega el rol de observador. La gran diferencia entre AWT y SAX es que SAX no permite registrar más de un auditor con cada `XMLReader`. [4, pp.264]

2.1.3. Tratamiento de contenido

XMLReader se registra una instancia de la interfase `org.xml.sax.ContentHandler` cómo auditor, mediante el método:

```
public void setContentHandler (ContentHandler handler);
```

La interfase `ContentHandler` tiene declarados once métodos que son invocados con forme el lector XML va leyendo el contenido del documento XML. Cuando el lector encuentra una etiqueta de inicio invoca el método startElement(), cuando lee el contenido del elemento, invoca el método characters() y cuando lee la etiqueta de fin de elemento, invoca el método endElement().

A dichos métodos les son enviados parámetros que describen la formación del elemento, cómo el espacio de nombre o atributos. [4, pp.265]

Método	Evento
startDocument	Localización de Etiqueta Raíz que indica el inicio del documento
startElement	Localización de Etiqueta de Inicio de un elemento o elemento vacío
characters	Después de indicar el inicio del elemento se invoca este método que obtiene el contenido de dicho elemento
endElement	Indica el final de un elemento
endDocument	Indica el final de Documento
startPrefixMapping	Indica el inicio del ámbito de un prefijo URI de espacio de nombres
endPrefixMapping	Indica el final del ámbito de un prefijo URI de espacio de nombres
ignorableWhitespace	Recibe notificaciones de espacios en blanco ignorados en el contenido del elemento
processingInstruction	Indica la localización de una instrucción de procesamiento
skippedEntity	Se invoca este método una vez por cada entidad saltada
setDocumentLocator	Recibe un objeto para la localización del origen de un evento dentro del documento

Tabla 2.1 - Métodos y eventos correspondientes

```
public class TextParser implements ContentHandler {

  public void characters (char ch[], int start, int length) throws SAXException{
    System.out.println(String.copyValueOf(ch,start,length));
  }

  public void startDocument () throws SAXException{    }
  public void startElement (String namespaceURI, String localName, String qName,
Attributes atts)throws SAXException{    }
   ….
}
```

Código 2.2 - Implementación de interfase *ContentHandler* [4, pp.266]

```
try{
      XMLReader parser = XMLReaderFactory.createXMLReader();
      ContentHandler textContent = new TextParser();
      parser.setContentHandler(textContent);
      parser.parse(uri);
} catch(SAXException ex) {
      ex.printStackTrace();
 }
  ….
```

Código 2.3 - Registro de *ContentHandler* por XMLReader [4, pp.268]

La clase concreta `org.xml.sax.helpers.DefaultHandler`, implementa a `ContentHandler`, dando cuerpo a los métodos además de otras características, siendo necesario solamente extender a `DefaultHandler` para contar con los métodos y sobrescribir los métodos utilizados.

En realidad SAX no reporta elementos, sino ocurrencias de etiquetas. Cuando el lector XML encuentra una etiqueta de inicio invoca el método startElement() y cuando encuentra su correspondiente etiqueta de cerradura invoca el método endElement(), Cuando se encuentra con una etiqueta vacía, se invoca el método startElement() y consecuentemente endElement(). En caso de que una etiqueta de cerradura no coincida con la correspondiente etiqueta de inicio el lector lanzará una `org.xml.sax.SAXParseException`. [4, pp.273]

```
public void startElement (String namespaceURI, String localName, String qName,
                          Attributes atts) throws SAXException;

public void endElement (String namespaceURI, String localName, String qName)
throws SAXException;
```

Los parámetros que reciben los métodos son similares e indican:

- String namespaceURI: indica el espacio de nombres, en caso de no encontrarse en un espacio de nombre su valor será una cadena vacía.
- String localName: indica el nombre del elemento después del prefijo del espacio de nombres en caso de tenerlo. Por ejemplo, xml2db:nombre en este caso tomaría el valor de nombre.

- String qName: contiene el nombre completo del elemento junto con el prefijo de espacio de nombre, xml2db:nombre.
- Attributes atts: es una colección de los atributos contenidos en la etiqueta de inicio.

Cuando el lector XML localiza un atributo no invoca algún método, los atributos pertenecientes a un elemento son enviados cómo una colección de tipo `org.xml.sax.Attributes` al método startElement() del elemento. Esta colección está diseñada cómo una lista (List) y contiene métodos para localizar un atributo especificado. [4, pp.279]

Después de invocar el método startElement() en el caso de no ser una etiqueta vacía, seguidamente se invoca el método characters() el cual recibe cómo parámetro el contenido del elemento en caso de que éste sea un elemento #PCDATA. [4, pp.284]

```
void characters(char[] charArray, int start, int lenght) throws SAXException;
```

- charArray: él contenido se pasa cómo un arreglo, por cuestiones de optimización.
- start: índice que indica el inicio del contenido.
- lenght: índice que indica la longitud de la cadena

Asignación de documentos.
Una vez creado el lector XML se le indica el documentos a procesar mediante el método *parse*(), el cual está sobrecargado, permitiendo indicar el documento XML mediante una cadena donde se especifica el URL del archivo, también se le pueden enviar un objeto `org.xml.sax.InputSource` que reúne diversos tipos de flujos de entrada direccionados hacia el documento a procesar.

```
void parse(String systemId) throws IOException, SAXException;
void parse(InputSource inputSource) throws IOException, SAXException;
```

InputSource mantiene un flujo hacia el documento XML, cuenta con tres constructores a los cuales se les pueden enviar un flujo de entrada (inputStream), un objeto Reader, y una cadena con el URL/URI hacia el documento. El objeto trata de acceder al documento por medio de su propiedad Reader, en caso de no conseguirlo, intenta mediante el inputStream y en caso de falla, tratara del abrir una conexión hacia el URL/URI, en caso de no lograr la conexión, lanzara una SAXException.

```
public InputSource() { }
public InputSource(InputStream inputStream) { }
public InputSource(Reader reader) { }
public InputSource(String string) { }

public static void main(String ars[]){
  InputSource inputSource;

  try{
    /* Flujo hacia el archivo XML */
    File file = new File(fileUri);
    FileInputStream fileSttream = new FileInputStream(file);
    inputSource = new InputSource(fileSttream);
```

```
      /* Creacion de Parser y asignacion de documento */
      XMLReader parser = XMLReaderFactory.createXMLReader();
      parser.parse(inputSource);
    }catch(SAXException ex) {
      ex.printStackTrace();
    }catch(IOException ex) {
      ex.printStackTrace();
    }
  }
```

Código 2.4 - Creación de InputSource.

Tratamiento de Errores

Mientras se procesa un documento XML el lector verifica la valides y buena formación de dicho documento, en caso de encontrar algún error en la estructura lanza una `org.xml.sax.SAXException` y detiene el proceso de lectura.

Las especificaciones de XML definen tres clases de problemas que pueden ocurrir en un documento [4, pp. 315]:

- Error Fatal: Un error de mala formación del documento XML, cuando el lector XML lo detecta, lanza una `org.xml.sax.SAXException` y detiene el proceso de lectura, el método *parse*() notifica este tipo de excepciones cuando son detectadas.

- Errores: los más comunes son de validación. En caso de que un error sea detectado se lanzara una `org.xml.sax.SAXParseException.`

- Advertencia: No es considerada un error, suele indicar un posible error de algún tipo en el documento. Los lectores XML pueden o no indicarlas y en caso de ser detectada, no se lanzara una excepción y continuará el proceso.

Los únicos errores que es seguro que el lector XML lance una excepción son los correspondientes a la mala formación del documento, en caso de necesitar ser informados de la ocurrencia de tipos distintos de errores existe la interfase `org.xml.sax.ErrorHandler` que puede ser implementada y registrada.

Para obtener información del error ocurrido se registra una implementación de la interfase `org.xml.sax.ErrorHandler` al XMLReader mediante el método setErrorHandler(), entonces el lector XML pasará una `SAXParseException` a los métodos correspondientes de la implementación. [4, pp.321-324]

```
void error(SAXParseException sAXParseException) throws SAXException
 void fatalError(SAXParseException sAXParseException) throws SAXException;
 void warning(SAXParseException sAXParseException) throws SAXException;

     /* Asignacion de atrapador de errores */
     XMLReader parser = XMLReaderFactory.createXMLReader();
     parser.parse(inputSource);
     parser.setErrorHandler(errors);
```

Código 2.5 - Registro de ErrorHandler

2.2. DOM

DOM (Document Object Model) es una estructura de datos abstractos que representa documentos XML cómo un árbol formado por nodos. Diversas interfaces representan elementos, atributos, datos de caracteres, comentarios e instrucciones de procesamiento de la estructura de árbol, la mayoría son subinterfases de la interfase común `org.w3c.dom.Node`, que provee los métodos básicos para la navegación del árbol.

La raíz del árbol es un objeto `org.w3c.dom.Document` que representa el documento completo y bien formado. El lector XML obtiene un flujo de entrada y crea el que objeto Document que representa el documento XML. La aplicación cliente utiliza los métodos de este objeto y de las demás interfaces para recorrer el árbol y extraer la información del contenido. DOM carga el árbol del documento completo en memoria dando la capacidad de insertar, mover, modificar o eliminar nodos. Además de poder crear nuevos documentos en memoria los cuales son escritos en un archivo con formato XML. [4, pp. 433]

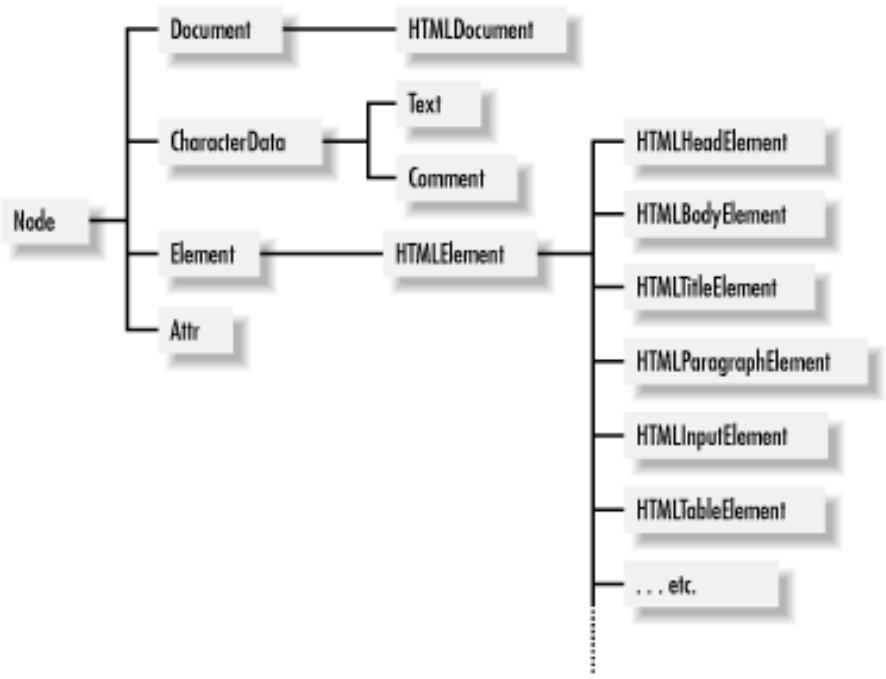


Figura 2.2 - Jerarquía de Clases de Elementos DOM

2.2.1. Árbol DOM

Dentro de DOM un documento XML es un árbol formado por nodos de diversos tipos. El árbol contiene sólo un nodo raíz y todos los demás nodos excepto la raíz cuentan con un nodo padre, sin embargo cada nodo tiene una lista de nodos hijos, en caso de que la lista está vacía, se considera a dicho nodo cómo nodo hoja. Cada nodo tiene un nombre, para elementos y atributos el nombre del nodo es el nombre predefinido.

DOM divide los nodos en 12 tipos, de los cuales 7 tipos pueden ser parte del árbol [4, pp.441]:

- Nodos Documentos
- Nodos de Elementos
- Nodos de Texto
- Nodos de Atributos
- Nodos de Instrucciones de procesamiento
- Nodos de Comentarios
- Nodos de Tipo de Documento
- Nodos de Fragmento de documento
- Nodos de Notación
- Nodos de Secciones CDATA
- Nodos de Entidades
- Nodos de Referencias a entidades

- Nodo de Documento: Dentro de la estructura de árbol DOM sólo puede contener un nodo documento.

- Nodos de elementos: Cada nodo de elemento tiene un nombre, un URI de espacio de nombres y un prefijo. Además contiene un nodo de texto con el valor de contenido.

- Nodos de Texto: Estos nodos contienen los caracteres de datos almacenados en el documento.

- Nodos de Comentarios: Los nodos de comentarios contiene un nombre correspondiente a #comment y un valor que es la cadena contenida, además de un nodo padre que lo contiene.

- Nodo de Instrucción de Procesamiento: Al igual que los nodos de comentarios, contiene un nombre, un valor y un nodo padre.

- Nodos de Secciones CDATA: Un nodo para secciones CDATA es un nodo de texto especial que representa el contenido de una sección CDATA del documento XML.

- Nodos de Referencias a Entidades: Cuando DOM encuentra una referencia incluye un nodo de referencia de entidad en la estructura de árbol del documento.

- Nodos de Tipo de Documento DTD: Un nodo de tipo de documento tiene un nombre que especifica la declaración de la DTD para el elemento raíz.

2.2.2. Parseo de Documentos

Diversos lectores XML proveen sus propias implementaciones de las interfaces estándar de DOM, ya que éste no tiene una clase o interfase que represente el lector, cada fabricante provee su propia clase o interfase.

A causa de que estas clases no comparten una interfase o clase en común, los métodos que utilizan para procesar los documentos XML pueden variar.

Todos los nodos de un árbol están representados por instancias de la interfase `org.w3c.dom.Node`. Cuya interfase ofrece métodos para agregar, mover, copiar o eliminar nodos del árbol, además de métodos para recorrer los nodos, obtener sus propiedades y valor correspondientes.

Mediante el método getNodeType() se obtiene el tipo del nodo actualmente posicionado, están definidos 12 tipos de nodos mediante el uso de constantes:

Tipos de Nodos	
Node.ELEMENT_NODE	Node.PROCESSING_INSTRUCTION_NODE
Node.ATTRIBUTE_NODE	Node.COMMENT_NODE
Node.TEXT_NODE	Node.DOCUMENT_NODE
Node.CDATA_SECTION_NODE	Node.DOCUMENT_TYPE_NODE
Node.ENTITY_REFERENCE_NODE	Node.DOCUMENT_FRAGMENT_NODE
Node.ENTITY_NODE	Node.NOTATION_NODE

Tabla 2.2 - Tipos de Nodos

Esta interfase `org.w3c.dom.Node` contiene métodos para obtener diversas propiedades del nodo, cómo getNodeName() para obtener el nombre del nodo, getNodeValue() que regresa su valor, getNodeType() indica el tipo de nodo, getNamespaceURI() indica el espacio de nombres, getPrefix() regresa el prefijo, getLocalName() regresa la parte local del nombre del nodo.

Los nodos cuentan con métodos para la navegación a través de la estructura de árbol, con estos se puede localizar el nodo padre, getFirstChild() regresa el primero o getLastChild() el ultimo nodo hijo, además de los atributos del nodo con getAttributes().

DOM almacena una lista de los hijos de cada nodo en una colección `org.w3c.dom.NodeList`, se utiliza para simplificar operaciones que requieran la iteración de los nodos hijos. Para obtener esta colección se invoca el método getChildNodes() de la interfase nodo. [4, pp.482-483]

2.2.3. Creación de documentos

El API DOM es de lectura y escritura, la interfase `org.w3c.dom.Document` tiene dos propósitos, representar un documento XML y proveer métodos para el acceso a su contenido, esquema y otras propiedades, y crear nodos de distintos tipos capaces de ser agregados al documento XML generado. [4, pp.493]

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
DOMImplementation impl = builder.getDOMImplementation();

/* Creacion de Arbol de documento */
Document doc = impl.createDocument("http://www.xmldb.com","xbd", null);

Node documento = doc.getDocumentElement();
Node elemento = doc.createElement("nombre");
Node texto = doc.createTextNode("Hector Cuesta");
Node comentario = doc.createComment("Creacion de nodos");
```

```
elemento.appendChild(texto); // se agrega el texto al elemento
documento.appendChild(elemento); // se agrega el elemento creado al arbol
documento
doc.insertBefore(comentario,documento);
```

Código 2.6 - Creación e inserción de elementos

Las mismas técnicas pueden ser utilizadas por todos los nodos en el árbol: de texto, comentarios, instrucciones de procesamiento y referencias a entidades, excepto los nodos de atributos, ya que no pueden tener nodos hijos, los atributos sólo pueden ser agregados a nodos elementos y sólo utilizar los métodos de la interfase `org.w3c.dom.Element`. [4, pp.504]

2.2.4. Interfases DOM

La funcionalidad de DOM se encuentra en el manejo de las distintas interfases que lo componen, las más relevantes se mencionan a continuación:

- Interfase Document: A diferencia de los nodos, la interfase Document cuenta con diversos métodos para el manejo exclusivo de nodos de documento, métodos para búsquedas y copias de nodos entre diferentes documentos. [4, pp.514]

Algunos de los métodos más útiles de la interfase Document son aquellos que se usan para obtener elementos a partir del nombre o identificador, sin importar la localización del nodo dentro de la estructura del árbol. [4, pp.517]

- `Document getDocumentElement(): regresa elemento raíz`
- `Element getElementById(String idElement): busca nodo a partir de identificador`
- `NodeList getElementsByTagName(String nombre): busca nodos a partir del nombre`

```
Document documento = parser.parse(inputSource);
NodeList articulos = documento.getElementsByTagName("articulo");

System.out.println("Lista de Articulos");
for (int i = 0; i < articulos.getLength(); i++){
  Node articulo = articulos.item(i);
  System.out.println("\t"+ articulo.getNodeValue());
}
```

Código 2.7 - Obtención de elemento con métodos de búsqueda.

- Interfase Element: La interfase Element es quizás la más importante de todas las interfases que componen DOM, cada documento debe de tener al menos con un nodo elemento ya que los elementos más que otros componentes definen la estructura de un documento XML. La interfase Element cuenta con métodos para obtener las propiedades de los elementos, además de los métodos que extiende de Node.

- Interfase Attr (atributos): DOM también define la interfase Attr, la interfase Element es la principal en el manejo de atributos, cada elemento puede contener no más de un atributo con el mismo nombre ya que los atributos son almacenados y obtenidos a través de su nombre. Los atributos no se consideran cómo hijos de los elementos que los contienen, ni los elementos que los contiene se consideran padres de los atributos. [4, pp.574]

- Interfase CharacterData: Esta interfase es una súper interfase genérica para nodos que en su mayoría esta compuesto de texto, incluyendo Text, CDATASection y Comment. La interfase CharacterData cuenta con métodos para manipular el texto contenido en nodos además de los métodos heredados de Node. [4, pp.559]

- Interfase Text: La interfase Text representa un nodo de texto, que puede ser hijo de un elemento, un atributo o una referencia a entidad. Sus métodos los hereda de la interfase CharacterData, el único método declarado es splitText() que sirve para manipular el contenido.

Capítulo 3

XML y Bases de datos

Las computadoras necesitaban de un lenguaje común para comunicar datos, a causa de los diversos formatos que manejan las aplicaciones en su información. Sin un lenguaje común, la diferencia entre los formatos de datos hace muy difícil su interactividad entre aplicaciones.

XML fue diseñado para simplificar la transmisión de datos estructurados en la web, logrando con esto la comunicación de datos entre computadoras mediante la utilización de un lenguaje común, sin importar la complejidad de la información y facilitando el entendimiento de los datos para los usuarios, además de proporcionar una manera común de estructuración, descripción e intercambio de datos. [17, classifications of XML documents]

Un documento XML es una base de datos solamente en el estricto sentido de la palabra. XML provee muchas de las características encontradas en una base de datos:

- Almacenamiento de datos (documentos XML).
- Esquemas que definen la estructura (DTD, Schemas XML).
- Lenguajes de recuperación de datos (querys, XQL, XQuery, Xpath, etc).
- Programación de interfaces (SAX, DOM, JDOM), etc.

Sin embargo no cuenta con muchas otras características [13, XML a Database]:

- Almacenamiento eficiente.
- Indexación.
- Transacciones.
- Integridad de datos.
- Acceso a múltiples usuarios.
- Procesos almacenados (stored procedures, triggers).
- Consultas a través de múltiples documentos.

3.1. Clasificación de documentos XML

Los documentos XML caen en dos categorías las cuales hacen denotar las diferentes necesidades para su procesamiento: data-centric y document-centric, los documentos data-centric son aquellos donde XML es usado para la transferencia de datos en su estructura física, los datos son almacenados en atributos y elementos de texto o con el uso de entidades. Los documentos document-centric son aquellos que se caracterizan por su estructura irregular y contenido mixto, además su estructura física es relevante. [18]

3.1.1. Documentos data-centric XML

Los documentos data-centric son documentos que utilizan XML para la transferencia de datos, la información contenida en este tipo de documentos puede ser de órdenes de compras, horario de vuelos, catálogos de productos, etc.

Están diseñados para su procesamiento mediante aplicaciones automatizadas. Este tipo de documentos se caracterizan por contar con una estructura regular, los elementos contenidos en éstos generalmente sólo contienen datos de texto u otros elementos anidados, pero no ambos y por lo general el orden de los elementos no es significativo. [17 Data-centric XML documents]

El tipo de datos encontrados en un documento data-centric, pueden originarse dentro de la base de datos, donde se requiere exponerlos cómo XML, y fuera de la base donde será necesario almacenarlos en la base de datos.

Por ejemplo, cualquier sitio web que construya dinámicamente documentos HTML donde se llenen plantillas con los datos extraídos de una base de datos, probablemente puede ser remplazado por una serie de documentos XML data-centric y hojas estilo XSL. [13, Data-Centric documents]

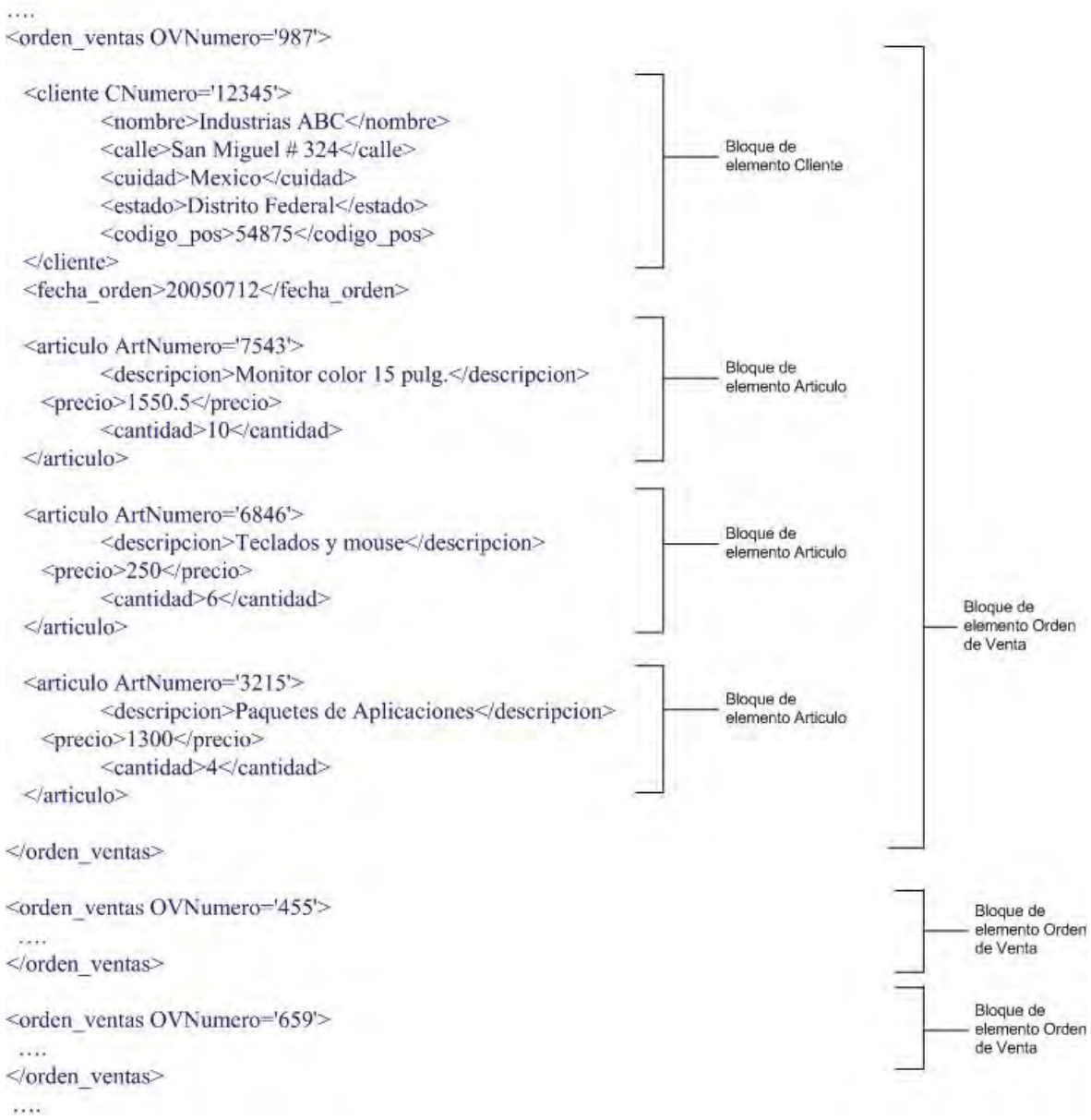


Figura 3.1 - Documento data-centric

Los documentos XML data-centric, se caracterizan por tener una estructura regular, cómo se muestra en la figura 3.1 los bloques son similares en diversos niveles.

3.1.2. Documentos document-centric XML

Los documentos XML document-centric están diseñados para ser leídos por los usuarios, éstos a diferencia de los data-centric son menos estructurados o pueden contar con una estructura establecida y por lo general los elementos contienen tanto datos cómo subelementos, además la ocurrencia de los elementos siempre es significativa. [17, Document-Centric XML documents]

Generalmente son generados a partir de otros formatos cómo RTF, PDF o SGML los cuales son transformados a XML, éstos no son generados desde una Base de datos. [13, Document-Centric Documents]

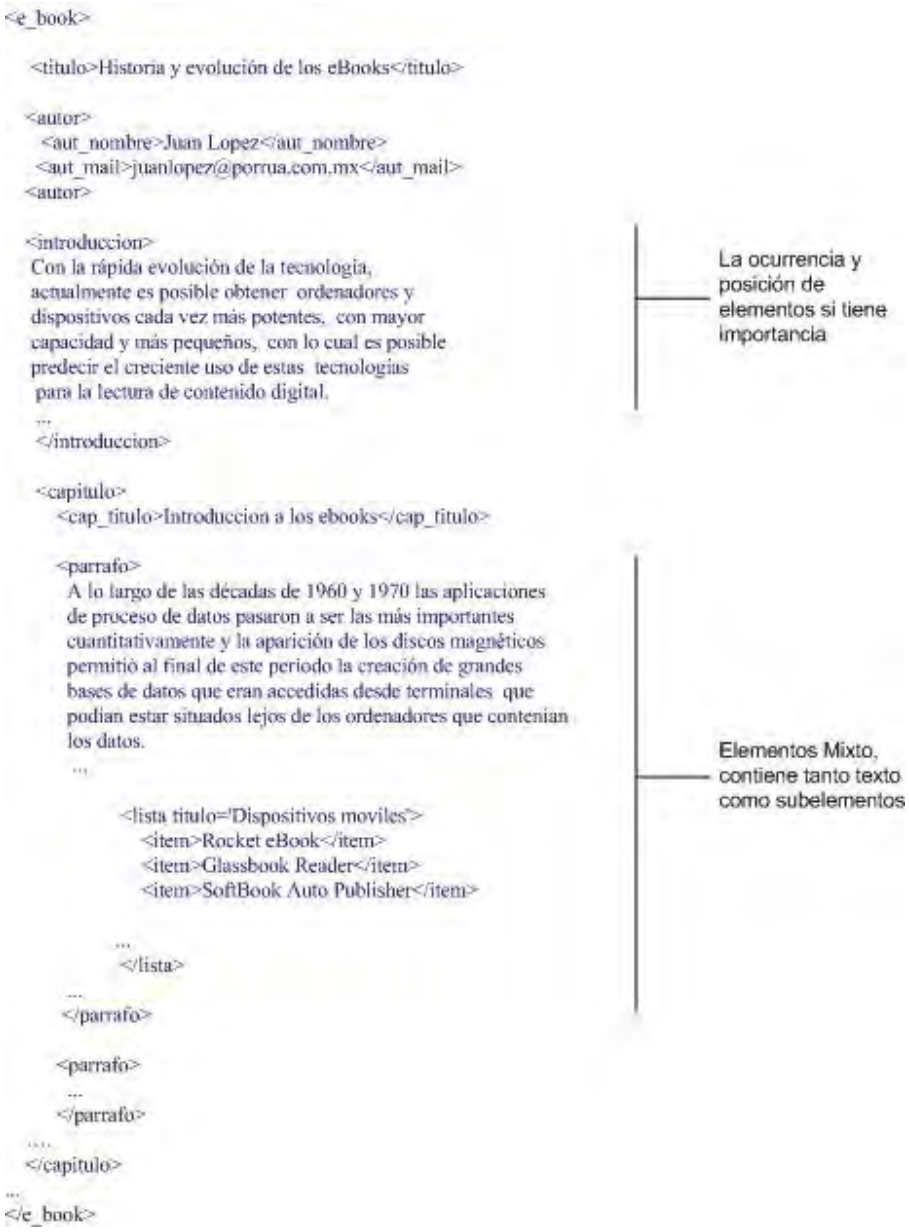


Figura 3.2 - Documento document -centric

En la figura 3.2 se muestra una posible estructura de un libro electrónico, dicha estructura puede ser semi-regular ya que los elementos contenidos pueden no aparecer en el mismo orden, la posición de los elementos es relevante, cómo el elemento introducción al inicio del libro, otro aspecto importante de los document-centric es que pueden contener elementos mixtos, es decir, que contengan texto u otros elementos anidados. Otro ejemplo de este tipo de documentos son las páginas web XHMTL/HTML.

3.2. Almacenamiento de XML en bases de datos

En ambos modelos de documentos XML es necesario en ocasiones almacenar los datos XML en algún repositorio ordenado o base de datos, que permita formas de almacenamiento o recuperación más sofisticados, especialmente si los datos XML son accedidos por múltiples usuarios.

Como ya se mencionó, las necesidades de almacenamiento de documentos XML, varían de acuerdo a su clasificación, dependiendo de la estructura y las posibles aplicaciones de los datos. [17, Storing and maintaining in a database]

3.2.1. Almacenamiento de documentos data-centric y document-centric

Ya que los documentos data-centric cuenta con una estructura regular, son representados eficientemente mediante tablas, las cuales consisten en columnas y renglones. Las bases de datos relacionales utilizan las columnas cómo campos y los renglones cómo registros para representar la estructura de datos. Por eso son recomendadas para el almacenamiento y mantenimiento de documentos data-centric. [17, data-centric]

Los documentos document-centric son manipulados manualmente y generalmente su estructura es poco regular y rígida. Las bases de datos relacionales necesitan de una estructura definida para el almacenamiento de datos XML, por lo tanto, no son aptas para almacenar datos contenidos en este tipo de documentos.

En el caso de almacenar documentos document-centric en una base de datos relacional, se pueden presentar los siguientes casos:

- Generación excesiva de tablas.
- Las tablas generadas pueden contener muchas columnas vacías.
- La estructura de la base de datos puede llegar a ser inválida en caso de que la estructura del documento XML cambie.

Una vez que el documento XML ha sido convertido a tablas relacionales es extremadamente difícil asegurar la buena formación del documento contra un esquema mientras es mantenida en la base de datos. Para asegurar que cada modificación este conforme con el esquema, el documento XML debe de ser primeramente recreado de las tablas relacionales. [17, Document-centric]

Las bases de datos nativas[4] XML fueron creadas para evitar los inconvenientes al almacenar documentos XML document-centric, sin embargo, no serán tratadas en este proyecto.

3.2.2. Mapeo de Esquemas

El mapeo entre el esquema de documentos y un esquema de base de datos, se aplica a los tipos de elementos, atributos y texto, por lo general se omiten secciones de la estructura física del documentos cómo entidades, secciones CDATA e información codificada y algunas secciones de estructura lógica cómo comentarios, instrucciones de procesamiento y secciones PCDATA, ya que las bases de datos tradicionales y aplicaciones sólo se concentran en los datos del documento XML, a consecuencia de almacenar un documento hacia una base de datos y después de la base reconstruir el documento de los datos, generalmente resulta un documento diferente.

Existen dos técnicas para mapear el esquema de documentos XML contra el esquema de una base de datos [13, Mapping]:

- Mapeo basado en tablas (table-based mapping)
- Mapeo de objetos relacional (object-relational mapping).

3.2.3. Mapeo basado en tablas

El mapeo basado en tablas es utilizado por muchos productos para transferir datos entre un documento XML y una base de datos relacional. Modela un documento XML cómo una tabla sencilla o un conjunto de tablas. Algunos productos de software que utilizan este modo de mapeo, generalmente incluyen metadatos de tablas y columnas al inicio del documento o cómo atributos en los elementos de las tablas y columnas. Esta técnica es útil para la serialización de datos relacionales, cómo transferir datos entre dos bases de datos. [13, table-based mapping]

```
<database>
      <table>
         <row>
            <column1>...</column1>
            <column2>...</column2>
            <...>
         </row>
         <row>
            <...>
          </row>
         <...>
      </table>
   </database>
```

Código 3.1 - Mapeo basado en tablas

[4] Pueden ser consultadas en la referencia [17, Native XML DB]

3.2.4. Mapeo de objetos relacional

El mapeo de objetos relacional modela los datos de un documento XML cómo un árbol de objetos que están especificados en los datos del documento. En este modelo los tipos de elementos con atributos, el contenido de elementos o contenido mixto son generalmente modelados cómo clases. Los elementos con sólo texto, atributos y secciones PCDATA son modelados como propiedades. Entonces el modelo es mapeado a una base de datos relacional. Las clases se mapean a tablas, las propiedades escalares son mapeadas a columnas y de las propiedades del objeto se genera la llave primaria o foránea del registro.

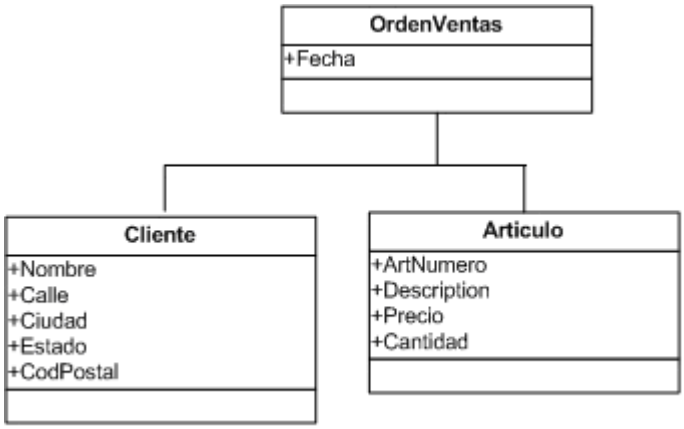


Figura 3.3 - Mapeo de Objetos Relacional

3.3. Lenguajes de Consulta

A causa de que la estructura de un documento XML es generalmente diferente a la estructura de una base de datos, es necesario implementar la utilización de otras tecnologías cómo XSLT (Extensible Stylesheet Language Transformations) que permite transformar documentos de etiquetas a la estructura establecida por el modelo antes de transferir los datos a la base o viceversa. [13, Query languages]

3.3.1. Lenguajes de consulta basados en plantillas

Son los lenguajes de consulta más comunes para obtener XML de una base de datos, no tienen predefinido un mapeo entre el documento y la base de datos, en su lugar se insertan sentencias SELECT en una plantilla, entonces el software se encarga de procesar los datos e incrustar los datos XML obtenidos de la consulta.

Los lenguajes basados en plantillas son usados exclusivamente para transferir datos de una base de datos relacional a documentos XML. [13, Template-Based]

```
<orden_ventas OVNumero='987'>
 <cliente CNumero='12345'>
 <nombre>Industrias ABC</nombre>
 <calle>San Miguel # 324</calle>
 <cuidad>Mexico</cuidad>
 <estado>Distrito  Federal</estado>

 <codigo_pos>54875</codigo_pos>
 </cliente>

<fecha_orden>20050712</fecha_orden>
<sentencia_sql>
  SELECT descripcion, precio,
  cantidad FROM articulo
  WHERE num_cliente = 12345
<sentencia_sql>
</orden_ventas>
```

```
<orden_ventas OVNumero='987'>
 <cliente CNumero='12345'>
  <nombre>Industrias ABC</nombre>
  <calle>San Miguel # 324</calle>
  <cuidad>Mexico</cuidad>
  <estado>Distrito Federal</estado>
  <codigo_pos>54875</codigo_pos>
 </cliente>
 <fecha_orden>20050712</fecha_orden>
 <articulo ArtNumero='7543'>
  <descripcion>Monitor color 15
pulg.</descripcion>
  <precio>1550.5</precio>
  <cantidad>10</cantidad>
</articulo>
<articulo ArtNumero='6846'>
 <descripcion>Teclados y
mouse</descripcion>
 <precio>250</precio>
 <cantidad>6</cantidad>
</articulo>
<articulo ArtNumero='3215'>
 <descripcion>Paquetes de
Aplicaciones</descripcion>
 <precio>1300</precio>
<cantidad>4</cantidad>
</articulo>
</orden_ventas>
```

Código 3.2 - Consulta incrustada.

3.3.2. Lenguaje de Consulta basado en SQL

Estos lenguajes utilizan sentencias de SQL modificadas, el resultado es transformado a formato XML. Es decir, son funciones incrustadas en las sentencias SQL a las que indican el campo y posiblemente el nombre del elemento que contendrá el dato del campo, también es posible indicar los atributos y subelementos que tendrá dicho elemento, entonces los registros obtenidos por la consulta son formateados a la estructura armada por las funciones. [13, SQL-Based]

```
SELECT Ordenes. OVNumero
   XMLELEMENT(
             NAME "orden_venta",
                     XMLATTRIBUTES(Ordenes. Id_orden AS OVNumero),
             XMLELEMENT(NAME "fecha_orden", Ordenes.fecha_reg),
             XMLELEMENT(NAME "cliente", Ordenes.cliente)
           ) AS xmldocument
FROM  Ordenes
```

```
<orden_venta OVNumero='12345'>
   <fecha_orden>20050712</fecha_orden>
   <cliente> Industrias ABC </ cliente>
```

```
</orden_venta>
```

Código 3.3 - Consulta basada en SQL

3.4. Transferencia de Datos[5]

Tomando en cuenta un formato preestablecido en los datos a transferir, es posible implementar estrategias que nos ayuden a esta tarea.

Un aspecto importante en la transferencia de datos entre documentos XML y bases de datos relacionales es el mapeo o concordancia de las estructuras que los definen. En el caso de un XML su estructura está definida a través de un esquema, mediante la utilización de DTD y XML Schemas.

3.4.1. Mapeo de Esquemas

Previamente al mapeo de los datos de un documento XML, se aplica el mapeo de su esquema, mediante el cual se describe la jerarquía de los elementos que se comparara contra la estructura de la base de datos. De igual manera se puede crean un esquema de documento XML, basado del esquema de la base de datos.

A través de la lectura de un esquema XML se va mapeando hacia un esquema de objetos que modelan los datos del documento XML, seguidamente el esquema de objetos es mapeado hacia el esquema de la base de datos, una vez que se cuenta con dichos esquemas se pueden emplear para el mapeo entre esquema de XML y el esquema de base de datos.

Esquema XML → Esquema de Objetos → Esquema de Base de datos

Por lo tanto:

Esquema XML → Esquema de Base de datos

- Mapeo de esquema XML a esquema de objetos: durante el mapeo del esquema XML los elementos que solamente contienen datos de tipo cadena son considerados como tipos de elementos simples los cuales representan simples tipos de datos, por ejemplo una propiedad del objeto que contendrá un valor simple, igual son tratados los atributos de determinado elemento.

  Los elementos que contienen elementos o contenido mixto o atributos son conocidos cómo tipos de datos complejos, estos elementos contienen una estructura y son equivalentes a clases.

- Mapeo de esquema de objetos a esquema de base de datos: en el mapeo entre el esquema de objetos contra el esquema de la base de datos, las clases son tratadas cómo tablas y las propiedades de los objetos, representan campos de dichas

[5] Esta sección esta basada en la referencia [21], donde se pueden encontrar ejemplos más detallados.

tablas, las propiedades de referencias son mapeadas cómo relaciones de llaves primarias y foráneas.

Modelo de contenido DTD	JAVA
<!ELEMENT nombre (#PCDATA)>	String nombre;
<!ATTLIST nombre edad CDATA>	int edad;
<!ELEMENT alumno (nombre, apaterno)>	Class Alumno
<!ELEMENT salon (alumno+)>; <!ELEMENT salon (alumno*)>;	Alumno[] alumnos;
<!ELEMENT clase_idioma (ingles|italiano)>	String ingles; // null String italiano; // null

Tabla 3.1 - Relación DTD – Objetos Java

3.4.2. Mapeo basado en tablas

El mapeo basado en tablas, establece que el documento XML cuente con una estructura donde se definen las tablas y campos contenidos en la base de datos destino. Dicha estructura sólo permite la transferencia de datos a tablas simples.

Comúnmente los datos a transferir son contenidos en documentos data-centric ya que se caracterizan por una estructura regular donde los elementos contenidos, sólo pueden contener subelementos o texto, pero no contenido mixto.

La utilización de SAX para el procesamiento de este tipo de documentos es de gran utilidad, al intercambiar datos de una base de datos a otra con la misma estructura, ordenando los datos contenidos en el documento de tal forma que las llaves principales sean insertadas previas a los registros dependientes.

3.4.3. Mapeo basado en objetos

Dentro del mapeo basado en tablas se limita a un conjunto pequeño de documentos. Para documentos cuya estructura es más anidada resulta muy útil el mapeo basado en objetos en el cual se arman objetos con la jerarquía de los elementos y subelementos basándose en el mapeo de esquema del documento. Una vez que se cuenta con la estructura de objetos, durante el mapeo del documento se van armando los objetos respectivos a los elementos y su posterior inserción.

Durante la generación de los objetos, los elementos de datos o atributos indicados en el esquema son considerados propiedades del objeto, así como las secuencias de elementos, corresponden a lista o colecciones de objetos e instancias de otros objetos.

3.4.4. Transferencia de XML a bases de datos relacionales

3.4.4.1. Transferencia basada en SAX

El orden de lectura que utiliza SAX en el procesamiento de un documento se podría definir cómo depth-first, width-second que se tratará más adelante en las estrategias de

transferencia. Lo que indica en caso de insertar los datos a la base, ciertos renglones tendrán que ser almacenados temporalmente en memoria durante la lectura del documento. Por ejemplo:

```
<profesor>
<idprofesor>154</idprofesor>
<nombre_prof>Juan
Perez</nombre_prof>
<titulo>Lic</titulo>
<curso>
<idcurso>40057</idcurso>
<nombre_curso>Calculo
III</nombre_curso>
<duracion>6</duracion>
<capacidad>50</capacidad>
<instalacion>
<cve_lugar>AUD00A</cve_lugar>
<lugar>auditorio A</lugar>
<horario>14:00</horario>
</instalacion>
<curso>
</profesor>
```

Código 3

```
<curso>
 <idcurso>40057</idcurso>
 <nombre_curso>Calculo III</nombre_curso>
  <duracion>6</duracion>
  <capacidad>50</capacidad>
   <profesor>
<idprofesor>154</idprofesor>
<nombre_prof>Juan Perez</nombre_prof>
    <dia>Lunes</dia>
    <dia>Martes</dia>
  </profesor>
      <profesor>
            <idprofesor>235</idprofesor>
<nombre_prof>Raul Martinez</nombre_prof>
<dia>Jueves</dia>
<dia>Viernes</dia>
</profesor>
<curso>
```

Código 4

Código 3.4 - Manejo de relaciones en documento XML

En el código 3, al procesar SAX éste fragmento XML en el código java se tendrá que almacenar los datos del profesor, sucesivamente los datos del curso y al final los datos de la instalación que se encuentran en el último nivel de anidamiento. Las inserciones dependerán de la jerarquía de las tablas contenidas en la base de datos.

Así se resalta la importancia en el orden de los datos del documento XML, ya que puede darse el caso en documentos con cientos de registros, que estos sean almacenados en memoria, lo cual es una desventaja en la utilización de SAX ya que se caracteriza por el uso mínimo de memoria.

En el caso de este fragmento del código 4 es posible insertar los registros conforme se van obteniendo durante la lectura del documento, aprovechando así las características de SAX.

3.4.4.2. Transferencia basada en DOM

DOM provee la capacidad de recorrer los nodos del documento en cualquier orden, además de poder obtener cualquier nodo varias ocasiones, con eso se evita los problemas de integridad referencial tratados en la transferencia basada en SAX. Sin embargo, DOM carga el documento completo en memoria y si el rendimiento de la aplicación es un objetivo principal tendría que tomarse en consideración su implementación.

Considerando el manejo de la memoria, existen 3 estrategias para optimizar la aplicación dependiendo de las necesidades de rendimiento:

- Depth-first, width-second: esta estrategia se basa en el orden de lectura de SAX, el cuál lee un documento de forma recursiva invocando eventos correspondientes a la entidad extraída de dicho documento. Es decir, cuando SAX localiza una etiqueta de abertura de un elemento e invoca el método correspondiente, de igual manera, al localizar una etiqueta de cerradura invoca su respectivo método, siguiendo esta lógica, si el documento contiene varios niveles de anidamiento de elementos el lector XML procesará hasta llegar al último nivel del elemento actual y regresando al nivel inicial continuando de manera consecutiva la lectura. Esta estrategia no brinda ninguna ventaja en DOM.

- Memoria mínima: esta estrategia minimiza la cantidad de memoria utilizada, almacenando sólo un renglón de datos a la vez. Recorriendo los nodos hijos de un nodo tres veces. En el primer recorrido se procesan los hijos que contienen llaves primarias conocidos cómo elementos de llave primaria. En el segundo recorrido se procesan los hijos que contienen únicamente datos, estos se conocen como elemento simples. En el tercer y último recorrido se procesan los hijos que contiene llaves foráneas, conocidos como elementos de llave foránea. Esta estrategia es más eficiente que depth-first, width-second con respecto a la cantidad de datos almacenados, sin embargo requiere recorrer tres veces los nodos.

- Apuntadores almacenados: esta estrategia reduce el número de ocasiones que un nodo es visitado, está basada en el hecho de la búsqueda a través de un árbol no indexado es lineal para cualquier conjunto de nodos hermanos, en tal caso a pesar de que el software conozca que elementos son parte de la llave primaria, elementos de datos o elementos de llave foránea no se puede llegar a ellos directamente. Por lo tanto debe de iniciar la búsqueda a través del conjunto de nodos uno a la vez. A diferencia con la estrategia de memoria mínima, cuando se recorre el conjunto de elementos en la búsqueda de elementos de llave primaria se pueden almacenar apuntadores a elementos de datos o llaves foráneas sin que sean procesados inmediatamente, evitando así un nuevo recorrido del conjunto y el número de visitas a un nodo del árbol.

Asumiendo que los apuntadores son considerablemente más pequeños que piezas de datos individuales, la estrategia de apuntadores almacenados provee de una razonable compensación entre memoria y velocidad, sin embargo SAX sigue teniendo ventaja en esos aspectos, ya que DOM necesariamente carga el documento completo. Ya que con respecto a la velocidad, SAX sólo visita cada nodo una sola vez, además que SAX no necesita tomarse el tiempo para crear un árbol del documento y es más veloz al recorrerlo, ya que recorrer el documento es parte del lector XML y no requiere de métodos extra.

Cuando se utiliza el mapeo basado en objetos para la transferencia de datos XML a bases de datos, las herramientas basadas en SAX siempre son más rápidas y utilizan menos memoria que las basadas en DOM.

La razón es por que en el mapeo basado en objetos es más útil para documentos data-centric y este tipo de documentos tienden a ser pequeños, en el caso de documentos

grandes tienden a ser serialización de bases de datos completas y la estructura de estos documentos por lo general no necesita ser almacenada completamente.

En particular, documentos que serializan una base de datos esencialmente consisten en múltiples subdocumentos centrados en datos almacenados bajo un elemento raíz, cada subdocumento representa la jerarquía de datos asociada con un simple renglón en el nivel superior de la tabla, por lo tanto cómo cualquier documento centrado en datos, relativamente pequeño. El elemento raíz solamente existe para mantener agrupados a los subdocumentos y por lo general es ignorado, por lo tanto el procesamiento de este tipo de documentos requiere de la misma cantidad de memoria que el procesamiento por separado de cada subdocumento.

En caso de que los elementos del documento XML puedan ser ordenados por elementos de llave primaria, elementos de datos y finalmente los elementos de llave foránea, SAX sólo requerirá almacenar un renglón de datos a la vez.

3.4.5. Transferencia de base de datos relacionales a XML

Los datos obtenidos de la base de datos puede ser recuperados de diversas tablas, pero en el código XML dichos datos pueden formar un simple elemento o generar nodos descendientes. A diferencia de cuando se insertan datos a la base de datos, el orden de los elementos con llave primaria o foránea no es importante, al recuperan datos de la base, estos son simplemente recorridos sin importar si su llave está ligada a la llave de registros de otras tablas.

Para la transferencia de datos de bases relacionales a XML deben de tomarse en consideración dos puntos; el primero se concentra en la una manera eficiente de recorrer la jerarquía de los datos, el segundo punto en el orden de registros hermanos, es decir, registros correspondientes a un mismo registro padre. Si el orden está almacenado en la base es suficiente recorrerlos de manera jerárquica y crear los elementos. Sin embargo, en el caso de contener el orden de los datos los elementos deben ser colocados en la posición correcta con respecto a su elemento padre.

3.4.5.1. Tablas paralelas

En esta estrategia la jerarquía de la base de datos se recorre en orden del documento, es decir, nodos padre e hijos, por lo tanto se abre un conjunto de registros de la tabla correspondiente al elemento raíz, creando un elemento por cada registro obtenido de esta tabla. Consecuentemente abre conjunto de registros correspondientes a las tablas descendientes.

Los conjuntos de registros están ordenados por sus columnas de orden, las columnas que proveen el orden de los elementos para las tablas hijas ocurren en sus tablas padres. Una vez obtenidos los conjunto de registros en paralelo, verifica el valor de la columna de orden del registro actual en cada uno de los conjunto de registros obtenidos y selecciona el registro con el menor valor, entonces compara los valores de orden de las columnas de

datos en el registro padre, si la columna de datos tiene un valor menor se crea un elemento de columna para él, repite el proceso para el resto de los registros.

Si el registro hijo tiene un valor de orden menor el proceso sigue de forma recursiva. Una desventaja de esta estrategia es que debe de ejecutar muchas sentencias SELECT, lo que ocasiona una gran cantidad de registros en memoria, la suma de todas las consultas, con la posibilidad de ocasionar un bloqueo o inconsistencia de datos en caso de que éstos sean modificados por algún otro proceso en la base.

3.4.5.2. Tablas individuales

La estrategia de tablas individuales reduce la cantidad de conjuntos de registros abiertos a la vez, en lugar de abrirlos todos juntos, los abre de forma individual, en caso de no conocer el orden de los datos se procesan en el orden en que obtienen.

Para cada registro de la tabla padre se crea un elemento así cómo los elementos para las columnas de datos, entonces se abren el conjunto de registros de las tablas hijas y se procesan una a la vez.

Una ventaja de esta estrategia es que el proceso puede adaptar los eventos de SAX al no conocer el orden de los registros, con menos conjuntos de registros abiertos que la estrategia de tablas paralelas.

3.4.5.3. Tabla universal

Esta estrategia reduce la cantidad de consultas a la base de datos, creando un simple conjunto de registros que contiene todos los datos del documento, construido mediante la agrupación y ordenamientos de llaves primarias y llaves foráneas.

La consulta puede ser construida utilizando sentencias SELECT unidas mediante las instrucciones JOIN o UNION, en el caso de los join’s regresa datos repetidos y para el union regresa valores nulos en las dichas columnas, ordenadas por las columnas de las llaves primarias y foráneas. Al procesar los registros de la consulta, se verifica cuando las columnas de las llaves cambien para crear los elementos correspondientes, así como los elementos hijos.

Cuando el orden de los elementos no esta almacenado en la base de datos, es posible adaptar las estrategias mencionadas tanto a SAX cómo a DOM, invocando los métodos correspondiente de SAX o creando un árbol mediante DOM. Evaluando las características de cada API según las necesidades de la aplicación.

Una referencia más detallada sobre el funcionamiento e implementación de las estrategias tratadas previamente, se puede encontrar en la referencia [21].

Capítulo 4

Servicios Web[6]

La especificación de JEE, se describe cómo una variedad de API's Java Empresariales que son integradas dentro de una plataforma completa. Indica cómo un servidor de aplicaciones configura, despliega y administra componentes empresariales cómo EJB, Servlets y JSP entre otros. Además indica como estos componentes de servidor (server-side) interactúan entre sí, con el servidor de aplicaciones y con diversas API's.

El objetivo principal de JEE es la portabilidad de componentes y la facilidad de desarrollo. Sin embargo, recientemente la interoperabilidad entre servidores de aplicaciones ha cobrado mayor importancia, tomando una nueva forma con las utilización de los principales estándares para WS (XML, SOAP, WSDL y UDDI) y las API's (JAX-RPC, SAAJ, JAXR, JAXP).

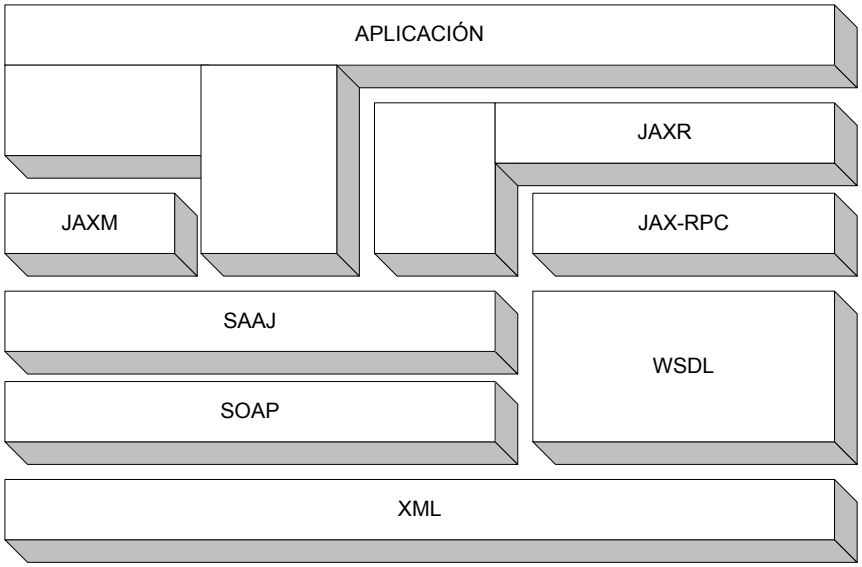


Figura 4.1 - Capas de protocolos y APIs incorporadas a los Servicios Web

[6] Este capítulo esta basado en la referencia [23], donde es posible obtener definiciones y ejemplos más detallados.

4. Servicios Web

El propósito principal de los servicios web[7] (Web Services - WS) es la interoperabilidad entre aplicaciones, desarrolladas en diferentes lenguajes o montadas sobre diversas plataformas para servicios web cómo JEE, Dot Net, Axis, etc. Para fines de este proyecto se referirá a las tecnologías de servicios web para la plataforma JEE.

El Basic-Profile[8] (BP) es un conjunto de reglas que indican cómo las aplicaciones deben utilizar las tecnologías comunes de servicios web, logrando unificar su desarrollo.

Como se ha recalcado anteriormente la interoperabilidad de los servicios web, el principal propósito de las tecnologías para servicios web es permitir que aplicaciones en diferentes plataformas intercambien datos de negocios. Las tecnologías para servicios web son utilizadas para la integración de aplicaciones A2A (Application-to-Application) también conocidas como EAI (Integración de Aplicaciones Empresariales), comunicando aplicaciones en una organización e intercambiando datos. O entre negocios B2B (Business-to-Business) es decir, comunicando múltiples organizaciones típicamente en sus patrones de negocios e intercambio de datos.

En el desarrollo de aplicaciones de software, la interoperabilidad es la capacidad de dos aplicaciones diferentes se comuniquen. Por ejemplo, una aplicación desarrollada en C++ corriendo bajo Windows pueda comunicarse con una aplicación java montada sobre Linux, para esto se deben utilizar las tecnologías de redes que son independientes a ambos sistemas operativos y hardware. Logrando esto mediante TCP/IP, DNS, HTTP, y los estándares WS (XML, SOAP, WSDL, UDDI).

Estos estándares proveen a los servicios web de un sistema de escritura mediante XML, un protocolo de mensajes SOAP, una interfase de definición de lenguajes WSDL y un registro de publicación de servicios UDDI.

XML contiene la información que será intercambiada entre las partes, mientras SOAP provee la forma de empaquetar y distribuir documentos XML para el intercambio a través de la red. WSDL permite la descripción de tipos de documentos XML y mensajes SOAP que deben de utilizarse para la interacción con el servicio web. Finalmente UDDI permite el registro del servicio web de una manera uniforme dentro de un directorio común de servicios, de tal forma que los clientes puedan localizar y accederlos.

[7] Un Servicio Web es una aplicación de software que conforma con el Basic-Profile de WS-I Organization.
[8] Las especificaciones para Servicios Web están contenidas en el publicado Basic-Profile de la WS-I (Web Services Interoperability Organización).

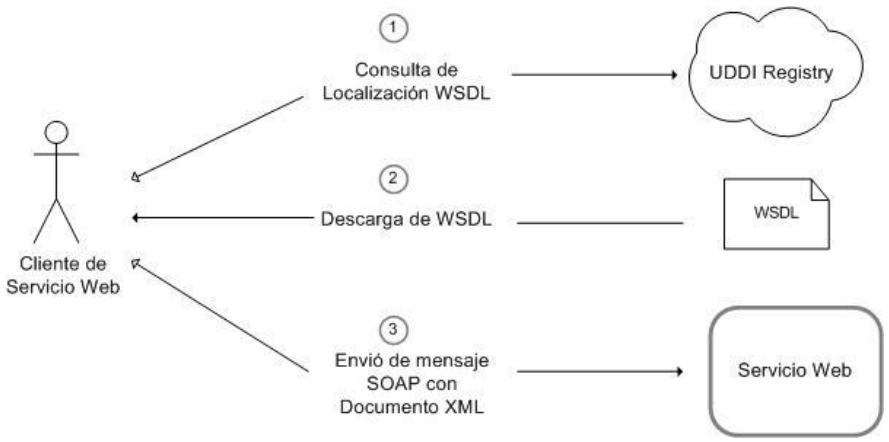


Figura 4.2 - Diagrama de Interacción de Servicios Web

4.1. SOAP

Los servicios web básicamente tratan el intercambio de documentos XML o datos que están organizados en este formato, SOAP (Simple Object Access Protocol) define un formato de empaque estándar para transmitir dichos datos entre aplicaciones sobre una red. No específicamente para un lenguaje de programación o plataforma determinadas ya que puede ser usado por cualquier tipo de aplicación a causa de que un mensaje SOAP es simplemente un documento XML. SOAP esta especialmente diseñado para contener y transmitir otros documentos XML además de información relacionada con su envió, procesamiento, seguridad, transaccionalidad y otras características de servicios.

Un documento SOAP XML es también conocido como un mensaje SOAP, transportado por medio de otros protocolos de redes como HTTP, SMTP, FTP y TCP/IP, el más común es HTTP el cual se utiliza para visualizar páginas web, con la diferencia de que los mensajes SOAP no están destinados a la visualización, cómo se muestra en la figura 4.3.

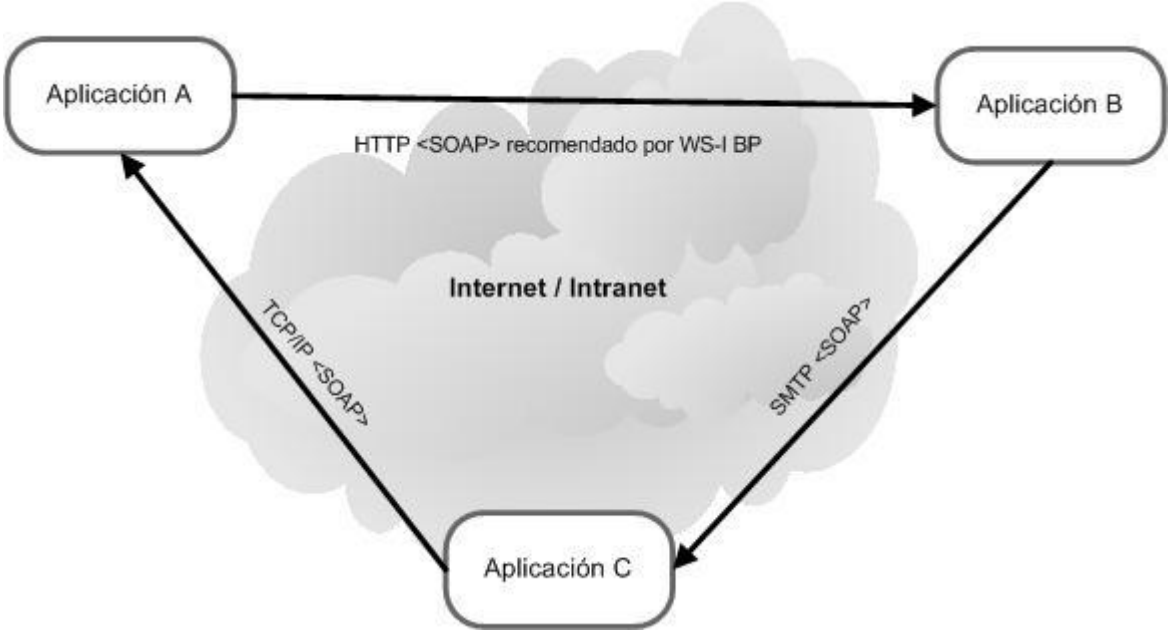


Figura 4.3 - Integración de SOAP con diversos protocolos a través de Internet

Existen dos métodos de envió de mensajes, one-way y request / response para los cuales se pueden enviar los mensajes SOAP. Respectivamente el mensaje viaja en una sola dirección del emisor al receptor, en el siguiente método el mensaje va del emisor al receptor considerando una replica de regreso al emisor.

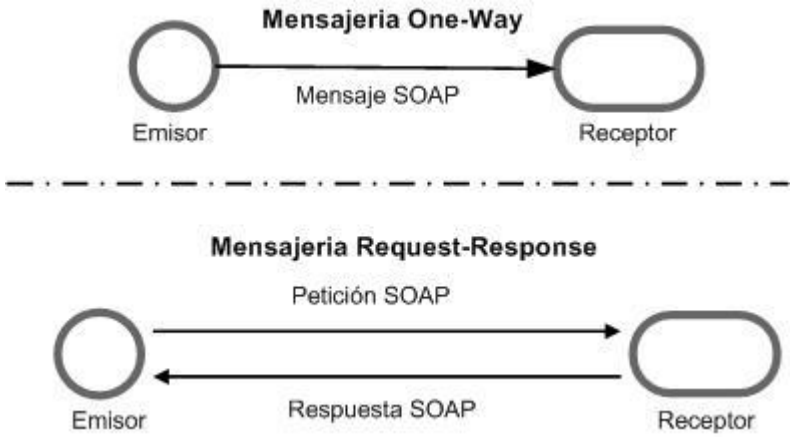


Figura 4.4 - Patrones de intercambio de mensajes

El esquema básico de un mensaje SOAP indica el elemento raíz envelope que contiene un elemento opcional para la cabecera del mensaje header y un elemento para el cuerpo del mensaje body contiene en formato XML los datos intercambiados entre las aplicaciones, delimitando los datos específicos de la aplicación.

```
<?xml version="1" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
   <soap:Header>
         <!-- bloques header -->
   </soap:Header>
   <soap:Body>
       <!-- Datos de la aplicacion -->
   </soap:Body>
</soap:Envelope>
```

Código 4.1 - Estructura de mensaje SOAP

Ya que el mensaje SOAP no limita el contenido del elemento body, los mensajes son ampliamente flexibles pudiendo intercambiar un amplio espectro de datos. Mientras tanto el elemento header contiene información acerca del mensaje mediante etiquetas definidas que describen algunos aspectos y cualidades del servicio asociado al mensaje, tales como credenciales de seguridad, identificadores de transacción, instrucciones de envió, información de pruebas y diversa información acerca del mensaje que es importante para el procesamiento de los datos contenidos por el elemento body.

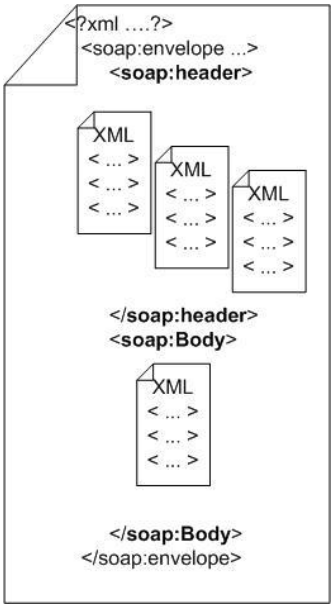


Figura 4.5 - Estructura básica de un mensaje SOAP

Los espacios de nombres son de gran importancia en los mensajes SOAP, ya que permiten integrar diferentes elementos XML dentro de los elementos header y body evitando posibles colisiones de nombres.

4.1.1. SOAP Headers y Message Path

La especificación SOAP define las reglas por las cuales los bloques de cabecera deben ser procesados a través del message path. El message path es la ruta que un mensaje SOAP toma desde un nodo emisor inicial hasta el nodo recibidor final, pudiendo ser procesado por nodos intermedios.

La aplicación que envía un mensaje SOAP se le conoce cómo sender y la aplicación que recibe el mensaje es conocida como receiver. El término cliente es comúnmente asociado al sender inicial de una solicitud de mensaje y el termino servicio web se asocia a el ultimo receiver del mensaje.

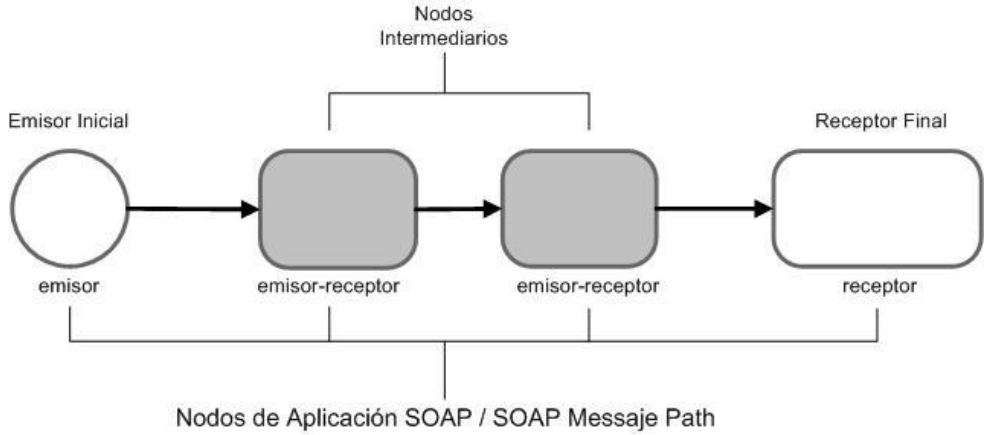


Figura 4.6 - Trayectoria de un Mensaje SOAP

4.1.2. SOAP Body

Es esquema de SOAP indica que el elemento header es opcional, sin embargo un mensaje SOAP debe forzosamente de contener un elemento body ya que este contiene los datos específicos de la aplicación, es decir, la información que se desea intercambiar entre los servicios web, en caso de ocurrir algún error durante el proceso, el contenido de body se ve reemplazado por un mensaje de error (fault message) que el nodo receptor (receiver) regresa cómo mensaje de respuesta al nodo emisor (sender) inmediato anterior en la ruta del mensaje (message path), el mensaje SOAP puede contener los datos de la aplicación o un mensaje de error, pero no ambos.

Sin embargo, cual sea el contenido del elemento body se recomienda que solamente el nodo receptor final procese dicho contenido ya que en caso de alguna modificación de la información, el nodo final no tendrá conocimiento de tal, a pesar de que los nodos intermedios en la ruta del mensaje puedan ver el contenido de body no deberán modificarlo, a diferencia de las cabeceras del mensaje que pueden ser modificadas por cualquier nodo intermedio.

4.1.3. Modos de mensajería SOAP

Exceptuando los mensajes de error, SOAP no especifica el contenido del elemento body, mientras éste sea un documento XML bien formado los datos pueden ser cualquiera, incluso puede estar vació. SOAP soporta cuatro modos de mensajeria los cuales son: RPC/Literal, Document/Literal, RPC/Encoded y Document/Encoded. Sin embargo el Basic Profile sólo permite los dos primeros mencionados y prohíbe explícitamente la utilización de RPC/Encoded y Document/Encoded.

- Document/Literal: En este modo el contenido de body contiene un fragmento de documento XML, un elemento XML bien formado que contiene arbitrariamente los datos de la aplicación.

- RPC/Literal (Remote Procedure Calls): El modo RPC/Literal permite que un mensaje SOAP modele la invocación de procedimientos o métodos con parámetros y valores de retorno. En este modelo de mensajes el contenido de body siempre lleva un formato, un mensaje de invocación RPC contiene el nombre del método y los parámetros de entrada, a su vez un mensaje de respuesta RPC contiene el valor de retorno y los parámetros de salida o posiblemente de error.

Es así cómo SOAP define un formato estándar XML para mensajería estilo RPC llamada RPC/Literal el cual especifica cómo los métodos y sus parámetros son representados dentro del elemento body de un mensaje SOAP.

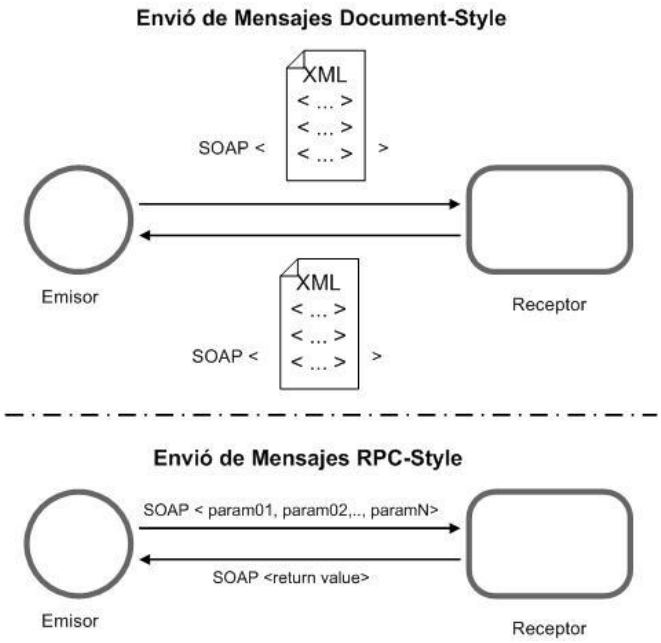


Figura 4.7 - Estilos de mensajería

Errores SOAP (SOAP faults)
Los mensajes de error SOAP son un mecanismo por el cual una aplicación SOAP reporta los errores a los nodos iniciales en la ruta del mensaje. Los errores SOAP son generados por nodos receptores. El receptor es requerido para enviar un mensaje de error SOAP de regreso al emisor inmediato sólo si el modo de mensajeria es Request/Response ya que en el caso de emplear el modo one-way el receptor no puede notificar del error al nodo emisor.

El mensaje SOAP que contiene un elemento fault dentro de body es llamado cómo un mensaje de error SOAP (SOAP fault). Los errores SOAP son análogos a las excepciones de java.

Cuando un error ocurre, el elemento body solamente contiene un elemento fault el cual a su vez contiene los elementos faultcode, faultstring, faultactor y detail. Mediante los cuáles se especifica el tipo y causa del error.

HTTP es un protocolo Request/Response lo cual indica que toda petición espera una respuesta del receptor.

```
SOAP Request sobre HTTP
POST /catalogo/precios HTTP/1.1
Host: www.libros.online.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 295
SOAPAction=""

<?xml version="1" encoding="UTF-8"?>
<soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/sopa/envelope"
        xmlns:lib="http://www.libros.online.com/catalogo" >
   <soap:Body>
      <lib:obtenPrecioLibro>
            <isbn>1254155848</isbn>
      </lib:obtenPrecioLibro>
   </soap:Body>
</soap:Envelope>
```

Código 4.2 - Petición SOAP sobre HTTP

4.2. WSDL[9]

WSDL (Web Service Description Language) es un estándar para describir la estructura de los datos XML intercambiados entre dos sistemas utilizando SOAP. Cuando se crea un nuevo servicio web es posible crear un documento WSDL que escriba el tipo de datos que están siendo intercambiados. Al igual que SOAP, WSL está basado en XML. Es utilizado para describir mensajes SOAP que pueden ser transmitidos entre servicios web clientes y servidores.

Su estructura básica cuenta con los siguientes elementos contenidos dentro de un elemento raíz definitions:

- types: utiliza el lenguaje XML Schema para declarar tipos de datos complejos y elementos que son usados a través de todo el documento WSDL.
- import: se utiliza para importar definiciones WSDL de otros documentos WSDL.
- message: definición tipificada de los datos a ser comunicados.
- portType, operation: mediante estos elementos se describen la interfase de un servicio web y sus métodos.
- binding: este elemento asigna un elemento portType y sus elementos operation a un protocolo en particular y un estilo de codificación.
- service: el elemento service es responsable de asignar una dirección de Internet a un binding específico.
- documentation: el elemento documentation indica ciertos aspectos del documento WSDL.

[9] Se puede consultar http://www.w3.org/TR/wsdl para una referencia completa de las especificaciones

```
<?xml  versio="1.0"  encoding="UTF-
8"?>
<definitions ... >
<types> ... </types>
<message ... ></message>
<portType ... ></portType>
<binding ... ></binding>
<service ... ></service>
</definitions>
```

Código 4.3 - Estructura Básica de Documento WSDL

4.3. UDDI[10]

UDDI (Universal Description, Discovery and Integration) define un conjugo estándar de operaciones WS que son utilizadas para almacenar y localizar información acerca de otros servicios web. Se puede utilizar un registro UDDI para encontrar un tipo particular de servicio web. Cuando se busca información acerca de un servicio web en un registro UDDI se puede invocar la búsqueda mediante diversas categorías, cada entrada en un registro UDDI provee información sobre donde esta localizado el servicio y cómo comunicarse con el además de que provee información acerca de la organización que hospeda un servicio web en particular.

UDDI es una especificación para crear un servicio de registros que catalogan organizaciones y sus respectivos servicios web, su implementación es denominada UDDI registry mediante la cual se conforma una base de datos capaz de soportar un conjunto de estructuras de datos estándar, definidos por las mismas especificaciones de UDDI.

Las especificaciones de UDDI describen tres tipos de operaciones SOAP que permites agregar, actualizar, eliminar y localizar información contenida en un UDDI registry. Diversos productos UDDI proveen interfases web para acceder a la información de un registro.

4.4. JAX-RPC[11]

El objetivo principal de JAX-RPC consiste en facilitar la comunicación entre aplicaciones java y de otras plataformas, utilizando las tecnologías para servicios web, se provee de un API orientada a objetos que implementan dichas tecnologías. Pudiendo utilizar JAX-RPC para acceder servicios web que corren en otros ambientes ajenos a java.

Es posible hospedar terminales (endpoints) de los servicios web en un servidor de aplicaciones JEE permitiendo que aplicaciones en otros lenguajes puedan acceder los servicios. JAX-RPC está diseñado cómo un API de java para servicios web, logrando la interoperabilidad entre aplicaciones JEE y otras plataformas.

[10] Véase http://uddi.xml.org/ para una referencia completa de las especificaciones
[11] Véase https://jax-rpc.dev.java.net/ para una referencia completa de las especificaciones

JAX-RPC es la tecnología central de los servicios web la cual define los modelos estándar de programación para servicios web clientes y servidores (endpoints) en plataformas JEE:

- El modelo de programación del lado del cliente: permite acceder a servicios web remotos cómo si fuera cualquier objeto local, utilizando métodos que representan operaciones SOAP.

- El modelo de programación del lado del servidor: permite el desarrollo de terminales de servicios web cómo objetos java o Enterprise JavaBean EJB.

4.4.1. Modelo de programación del lado del servidor

JAX-RPC define dos modelos de programación del lado del servidor: JAX-RPC service endpoint y EJB endpoint.

Terminal de servicio JAX-RPC (JAX-RPC service endpoint)
Una terminal de servicio JAX-RPC (JSE) son componentes JEE los cuales corren almacenados en un sistema contenedor de servlets, en lugar de utilizar flujos HTTP, utilizan mensajes SOAP mediante métodos definidos en la interfase java.rmi.Remote.

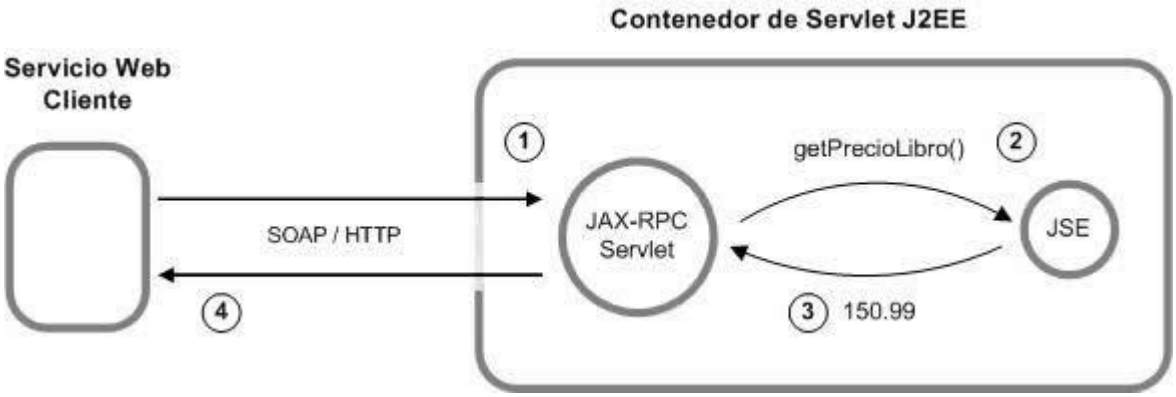


Figura 4.8 - Servlet JAX-RPC

Terminales EJB (EJB endpoint)
Ya que JAX-RPC invoca métodos mediante interfases remotas, de igual manera es posibles utilizarlos para acceder EJB's los cuales son accedidos mediante Java RMI, por lo tanto mediante el uso de JAX-RPC es posible utilizar EJB's cómo terminales de servicios web. Cómo SOAP es un protocolo de mensajería sin estado, sólo los EJB's de sesión sin estado (stateless session bean) pueden ser utilizados como terminales EJB.

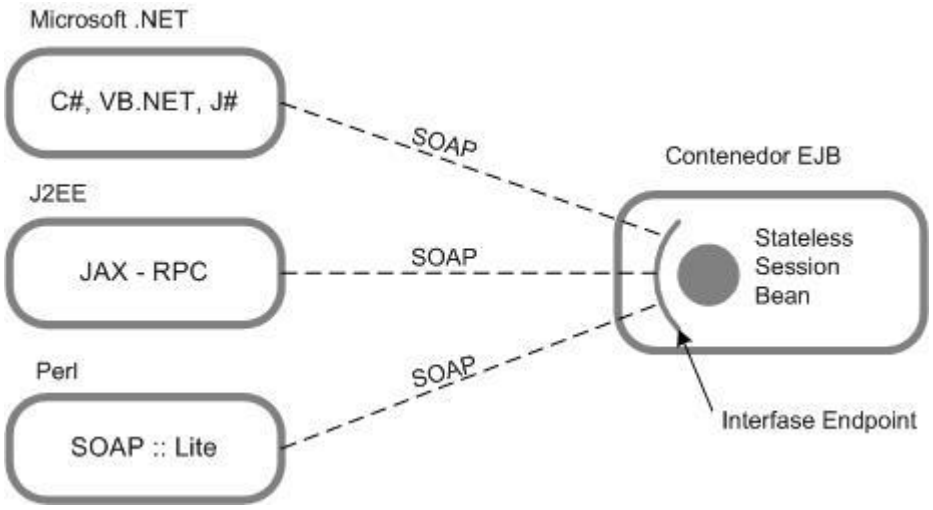


Figura 4.9 - Utilización de SOAP para acceder EJB's

4.4.2. Modelo de programación del lado del Cliente

JAX-RPC define tres modelos de programación cliente que son:

Generated stub
El compilador JAX-RPC genera una interfase remota java y la clase stubs que implementan las operaciones de la terminal descritas en el documento WSDL, es el de mayor aceptación.

Cuando un método es invocado el "stub" envía un mensaje SOAP hacia la terminal del servicio web, por ejemplo en el caso de un servicio web en Dot Net, la terminal .Net procesa el mensaje SOAP y regresa la respuesta hacia el stub, entonces el stub extrae los resultados del mensaje SOAP recibido y los envía al cliente.

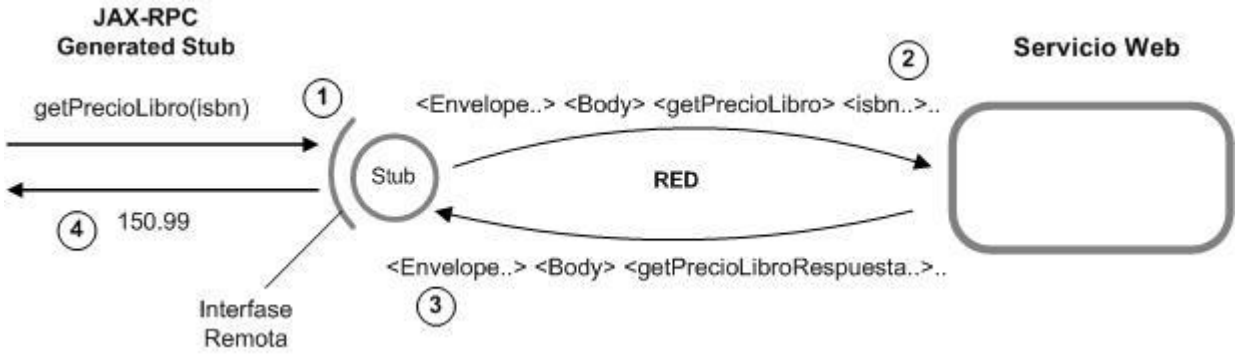


Figura 4.10 - Comunicación entre un Generated Stub y un Servicio Web

Dynamic proxy
Un dynamic proxy es utilizado de la misma forma que los generated stubs, excepto que la implementación stub de las interfases remotas son generadas dinámicamente en tiempo de ejecución, utilizando el API de Java Reflection.

Dynamic Invocation Interface (DII)
Este modelo permite ensamblar llamadas SOAP a métodos dinámicamente en tiempo de ejecución, similar al manejo de CORBA. JAX-RPC DII es del tipo de Java Reflection lo cual permite obtener referencias de un objeto que representa una operación de servicio web en la forma de un método invocado sin la necesidad de un stub o una interfase remota.

4.5. SAAJ[12]

SAAJ es una API basada en SOAP, es decir modela la estructura de los mensajes SOAP (*SOAP* Messages with Attachments "SwA"), puede ser utilizada para crear, leer o modificar mensajes SOAP usando el lenguaje java, que incluye clases e interfases que modelan los elementos envelope, body, header y fault de la estructura SOAP.

4.6. JAXR[13]

JAXR (Java API for XML Registries) es un API para el acceso de diferentes tipos de registros de negocios basados en XML, es utilizado principalmente para UDDI y registros ebXML. Se puede comparar JAXR con JDBC ya que provee acceso para varios tipos de bases de datos relacionales, mientras tanto JAXR provee acceso a diversos registros ebXML o UDDI.

JAXR provee de una API Java para ejecutar operaciones de registros sobre un conjunto diverso de registros, también provee de un modelos de información unificado para describir el contenido de un determinado registro. Ya que existen algunas diversidades en términos de sus capacidades entre diferentes proveedores de registros.

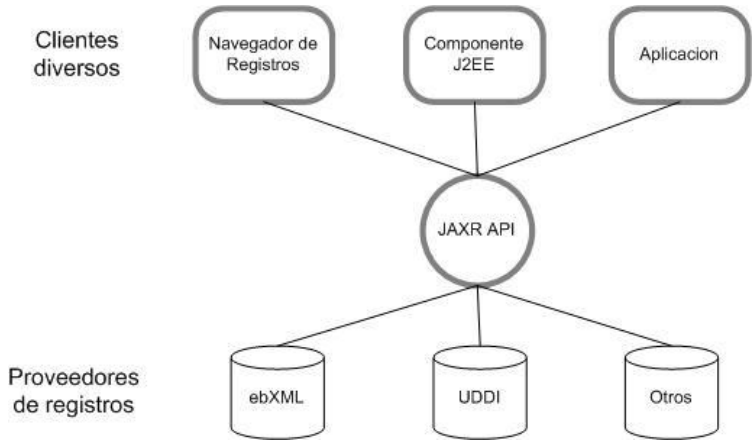


Figura 4.11 - Acceso a Proveedores de registros a través de JAXR

[12] Véase https://saaj.dev.java.net/ para una referencia completa de las especificaciones
[13] Véase http://java.sun.com/webservices/jaxr/index.jsp para una referencia completa de las especificaciones

4.7. JAXP

Dentro de las API's proporcionadas para los servicios web se encuentra JAXP (Java API for XML processing), que brinda un API para el procesamiento de dichos documentos XML. JAXP soporta los dos lectores estándar para el procesamiento de XML, los cuales son SAX y DOM, mencionados en el capitulo II.

Capítulo 5

Sistema de Transferencia de información crediticia, requerida por Buró de Crédito, basada en XML y Servicios Web

Buró de crédito requiere como parte de su contrato, la integración de la información de los consumidores que pertenezcan a la entidad financiera que solicite sus servicios. Sin embargo, el formato establecido para el intercambio de los datos es rígido y de difícil comprensión e igual de complejo para su implementación en aplicaciones.

Siendo que Buró de crédito establece que las entidades financieras que pretendan integrar su información en la base de datos, deben desarrollar sus propios componentes o sistemas que adapten los datos al formato establecido, la inversión de recursos y tiempo es considerable para dichas entidades.

Se propone el apego de los datos a las especificaciones de XML para agilizar la construcción de los archivos con la información crediticia a ser transferida. Con esto se amplía la gama de opciones y herramientas con las cuales pueden contar las entidades financieras para el cumplimiento del requisito.

Además de la generación de los archivos de registros, es necesario establecer un proceso de intercambio de los mismos, el cuál permita automatizar su envío hacia un repositorio de archivos, donde el sistema de Buró de crédito pueda procesarlos, integrando así los registros a la base de datos.

Se plantea una coordinación entre el sistema central de Buró de crédito y los sistemas de de las entidades financieras mediante el uso de servicios web, los cuales facilitan la comunicación entre aplicaciones independientes.

5.1. Caso de Estudio

Existen aplicaciones que se encargan de recabar información para su estudio y análisis, donde la información es obtenida por diversas entidades las cuales cuentan con sus propios sistemas de recuperación de datos, al final dichos datos se envían a una aplicación central para su proceso, pero la información puede llegar en diversos formatos como archivos de texto plano, RTF, archivos Excel, etc.

Para lograr integrar los datos puede requerirse de muchos recursos ya que los sistemas de gestión de datos pueden no compartir la misma estructura. Sin embargo, es posible establecer un formato que sea capaz de procesar la aplicación central, forzando a que las aplicaciones-cliente se adapten al mismo.

Un ejemplo real de esta situación, lo conforma el sistema actual de Buró de Crédito[14], ya que solicita a las Entidades Financieras del país, información crediticia sobre sus consumidores, para procesar ésta información Buró de Crédito solicita a las Entidades Financieras se apeguen al formato INTF propuesto, un punto importante es que las Entidades Financieras son responsables de desarrollar sus propios sistemas para la extracción de los datos y la generación de los documentos solicitados.

El formato INTF detallado en las secciones posteriores, a simple vista denota complejidad y rigidez, lo que hace exhaustivo el desarrollo de aplicaciones que lo generan, dependiendo de la plataforma de desarrollo y esquema de base de datos que implemente una determinada Entidad Financiera.

Otro punto importe, es la persistencia de los datos, la Entidad Financiera envía los documentos generados a Buró de Crédito el cual los procesa en su sistema, en caso de detectarse un error en el formato de la información durante el proceso, el documento es rechazado y se le solicita una nueva versión corregida, la revisión de código y regeneración de los documentos puede realizarse en varios ocasiones. Obviamente consumiendo tiempo y recursos.

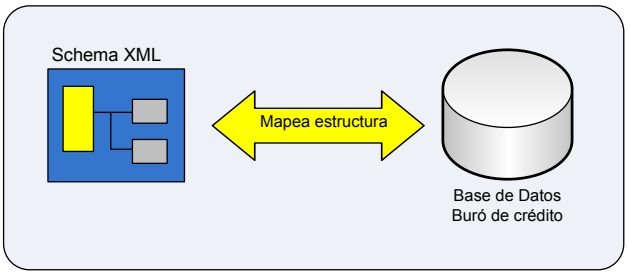
[14] Véase http://www.burodecredito.com.mx/otorgante/lib/contratacion_v1.pdf para verificar los requisitos de contratación

5.2. Modelo del Sistema

A partir del caso de estudio, se realiza una propuesta para mejorar el funcionamiento en tiempo y costos del proceso de extracción, transferencia y persistencia de datos, tanto del lado de una Entidad Financiera y el sistema de Buró de Crédito.

Actualmente XML es soportado por una gran cantidad de herramientas, mediante las cuales una Entidad Financiera puede extraer la informacion requerida para la generación de los archivos de registros.

Se propone un lenguaje XML que se adapte a los requerimientos de Buró de Crédito para la generación del archivo de registros. Para el diseño de éste lenguaje se creará un esquema que contemple una estructura que se adapte al diseño de tablas donde serán poblados los datos. Es decir, el esquema estará mapeado a ciertas tablas de la base de datos de Buró de Crédito.



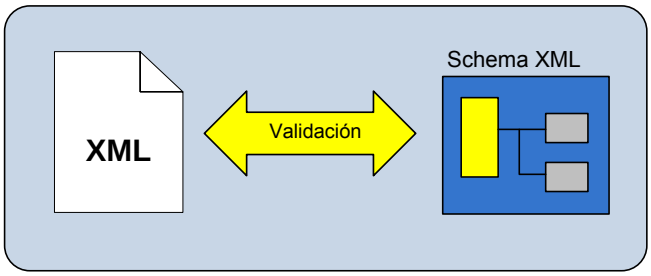


Figura 5.1 - Esquema basado en BD, Figura 5.2 – Archivo XML validado contra esquema

Los archivos de registros que se generén, tendran que ser validados contra dicho esquema para garantizar un poblado de datos exitoso. Evitando con esto el rechazo de documentos en numerosas ocasiones, logrando un ahorro de tiempo y recursos para ambos actores.

El proceso de validación será ejecutado antes de enviar los archivos, es decir, la aplicación-cliente para transferencia de archivos correrá dicho proceso previo al envio, los componentes de validación notificarán cualquier error en los datos o estructura, indicando al usuario la verificacion de su proceso de extraccion de datos.

Dicho esquema será publicado en el sitio web destinado para la aplicación de Buró de Crédito, donde podrá ser accedido libremente para las validaciones pertinentes. Con esto se brinda la libertad se utilizar cualquier otra herramienta o medio de validación independiente a la aplicación-cliente de transferencia de archivos, así las aplicaciones de las entidades financieras podrán mantener un bajo acoplamiento con los componentes proporcionados.

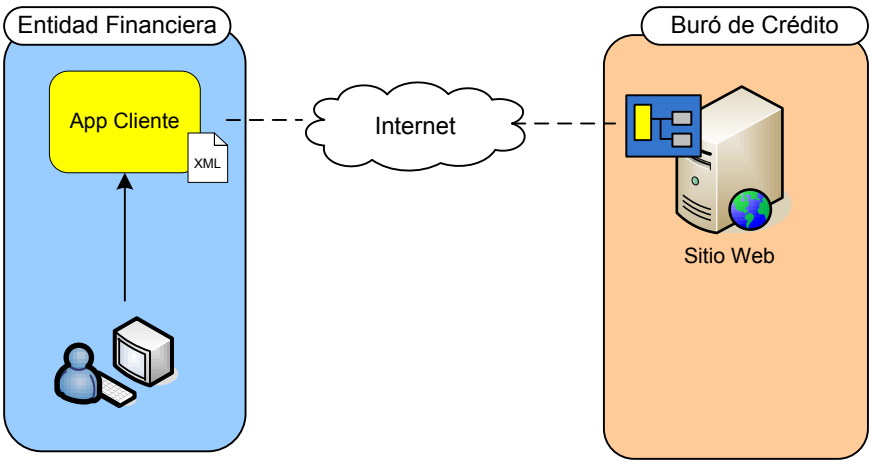


Figura 5.3 – Publicación de esquema de validación

Para la generación de los archivos de registros se basará en las especificaciones del formato INTF proporcionadas por Buró de Crédito, sin embargo, el proceso de persistencia y esquema de tablas de Buró de Crédito son completamente ajenos, por lo tanto, se deducen según las mismas especificaciones. Acotándonos sólo a la persistencia de los datos requeridos, ya que el propósito para con los datos se desconoce.

Se dispondrá de una aplicación-cliente para transferencia de archivos basadá en Java/Swing, que contendra las librerias con los componentes necesarios para realizar la extracción de información de la base de datos de la entidad financiera, la validación de los archivos de registros y el envío hacia el repositorio de archivos, como parte de la seguridad y rendimiento de la aplicación, el envió se realizará mediante el protocolo SFTP[15] hacia un repositorio asignado a las entidades financieras. Se propone que Buró de Crédito ponga a disposición dicha aplicación, facilitando así el cumplimiento de los requisitos para su contratación.

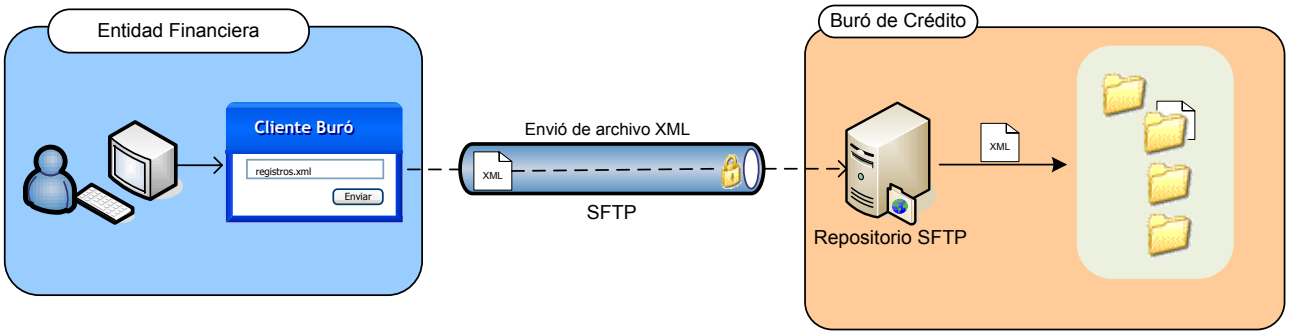


Figura 5.4 - Modelo de transferencia de archivos

Una vez transferido un archivo, la aplicación-cliente se encarga de la invocación del servicio web para la notificación del cumplimiento. Siendo ésta la última fase del proceso de envío, el servicio web es responsable de encolar la notificación para su posterior tratamiento por los componentes que procesan los archivos recibidos, almacenando los

[15] Secure File Transfer Protocol, (Protocolo Seguro de Transferencia de Archivos)

registros en la base de datos propia de Buró de Crédito, para su integración a los procesos internos de la institución, los cuales están fuera del alcance de éste proyecto.

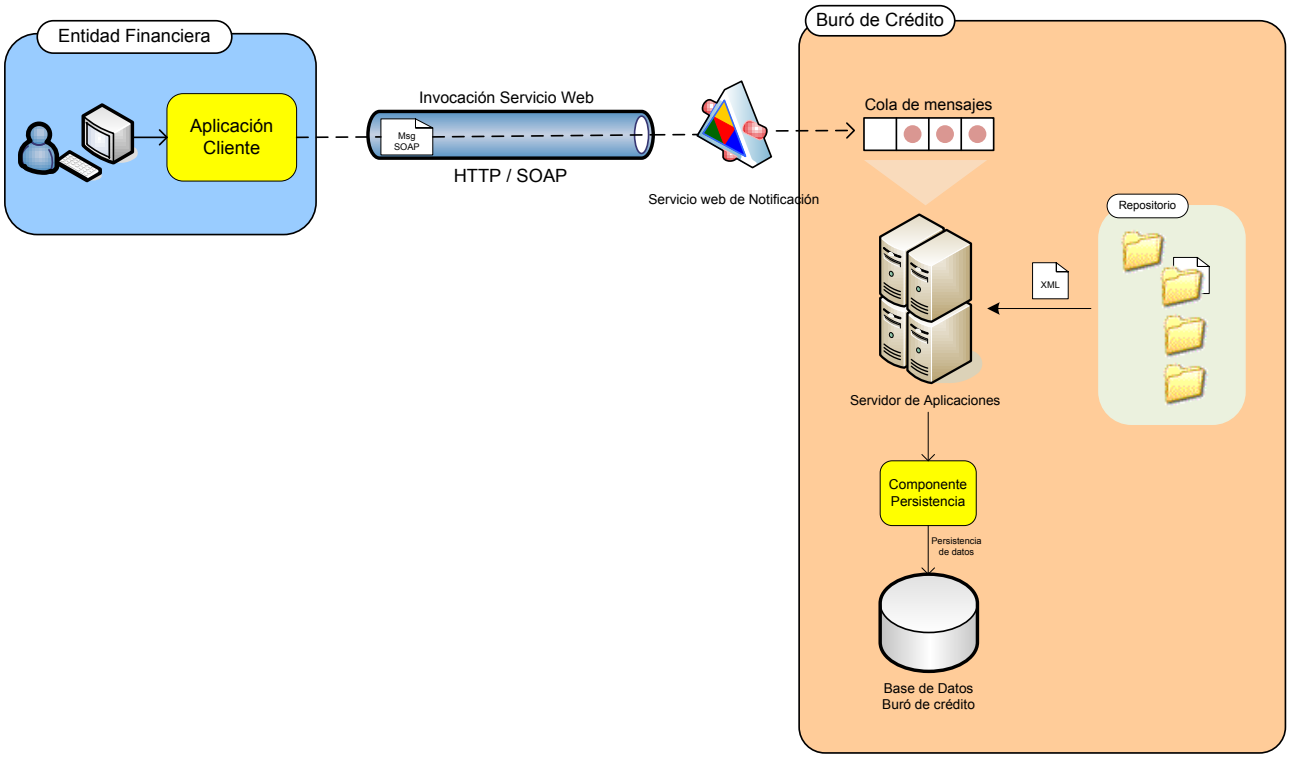
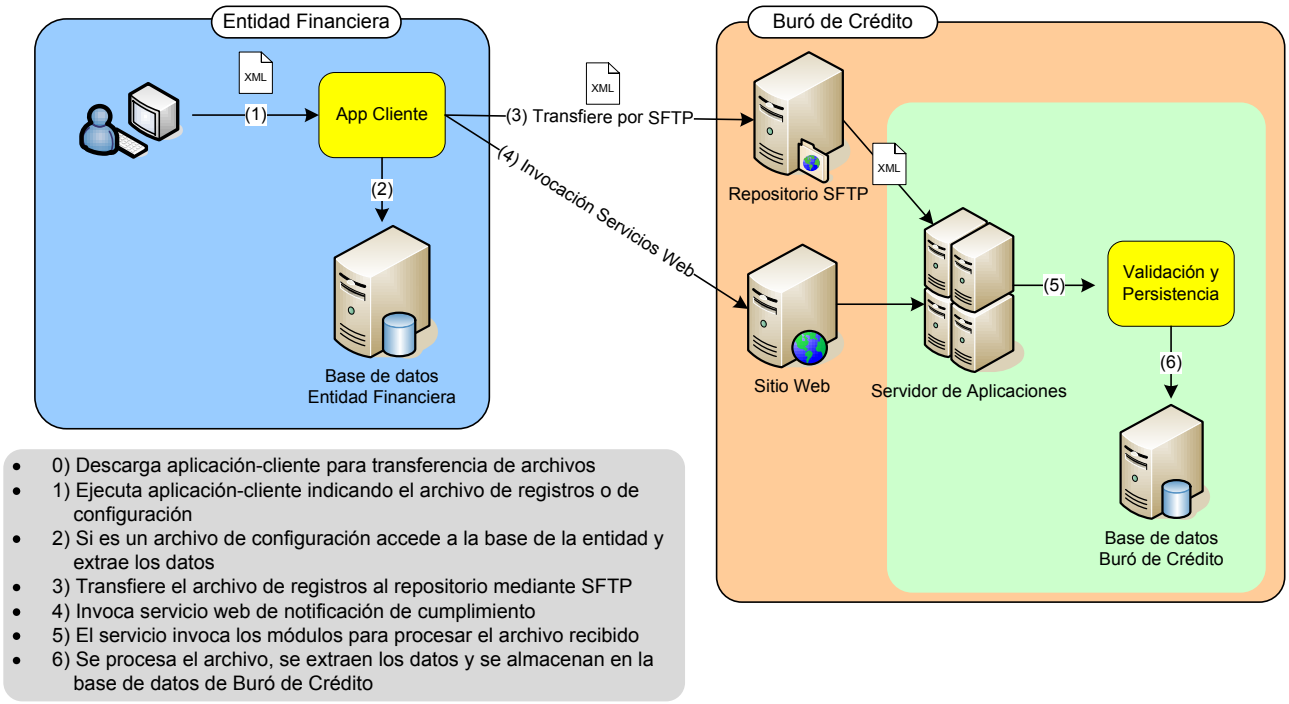


Figura 5.5 - Modelo de invocación del servicio web y administración de procesos

La figura 5.6 muestra a grandes rasgos los pasos que conforman el sistema de transferencia de información crediticia. En los siguientes apartados se describirán a más detalle los componentes y procesos integrados en el sistema.



- 0) Descarga aplicación-cliente para transferencia de archivos
- 1) Ejecuta aplicación-cliente indicando el archivo de registros o de configuración
- 2) Si es un archivo de configuración accede a la base de la entidad y extrae los datos
- 3) Transfiere el archivo de registros al repositorio mediante SFTP
- 4) Invoca servicio web de notificación de cumplimiento
- 5) El servicio invoca los módulos para procesar el archivo recibido
- 6) Se procesa el archivo, se extraen los datos y se almacenan en la base de datos de Buró de Crédito

Figura 5.6 – Arquitectura del Sistema

5.3. Alcances y limites

La propuesta está basada en las especificaciones del formato INTF[16], sin embargo, los requerimientos de Buró de Crédito contemplan la información crediticia de consumidores, tanto personas físicas, personas morales y créditos hipotecarios de una determinada Entidad Financiera.

Ya que el propósito de este proyecto es mostrar las ventajas de XML para estructurar los datos, sólo se contempla el desarrollo para personas físicas, ya que teniendo éstas bases, el desarrollo de personas morales e hipotecarias sigue los mismos lineamientos. Ya que se propone cambiar el formato INTF por XML, se diseñó un lenguaje de etiquetas equivalente a los requerimientos de Buró, dicho lenguaje se valida contra un Schema XML que impone la sintaxis y reglas de la información, de nuevo se destaca que el lenguaje fue diseñado para soportar solamente los datos de personas físicas.

De esta manera, las Entidades Financieras son libres de utilizar cualquier herramienta XML que se adapte a su plataforma de trabajo o motor de base de datos, para la extracción de la información y generación de los archivos de registros. Dentro del desarrollo de este proyecto se contempla un módulo de extracción de datos adaptándolos al lenguaje propuesto, capaz de configurarse mediante un lenguaje XML específico para el módulo, el cual está integrado en la aplicación-cliente.

Se contempla la coordinación del proceso de recepción de archivos, mediante una comunicación a través de un servicio web, donde Buró de Crédito facilitará los medios para su invocación.

Como se ha mencionado, Buró de Crédito solicita la información a diversas entidades financieras, por lo cual surge la necesidad de contemplar la concurrencia y manejo de transacciones, además, ya que el procesamiento de un documento puede tomar tiempo, se contempla el proceso principal de poblado de datos como un proceso asíncrono.

El esquema de tablas e información contenida, es información real de una entidad financiera, sobre la cual se adaptarán los componentes que conforman éste proyecto.

No se tocarán temas de seguridad a través del trabajo, solo se contempla el uso un protocolo seguro como lo es SFTP, ya que los detalles de instalación y configuración son parte de la infraestructura propia de las instituciones involucradas. Sin embargo, para posibles adaptaciones al sistema en el contexto de la seguridad de los datos, se podría sugerir el manejo de certificados digitales para autenticación de los actores, la encriptación de los archivos de registros mediante llaves asimétricas, invocación de los servicios web mediante HTTPS y manejo de permisos para la invocación de los servicios publicados.

Se contempla que Buró de Crédito cuenta con un modulo de administración de entidades financieras, donde se registran las nuevas entidades que requieran los servicios de Buró, sin embargo, éste trabajo se limita a refinar el proceso de transferencia de archivos, por lo que algunos aspectos de dicha administración no serán tratados o se darán por hecho.

[16] Consultar Apéndice C

5.4 Componentes

El sistema contempla una librería Core que contiene componentes para la extracción, validación y persistencia de los archivos de registros, dicha librería es compartida por ambas aplicaciones, tanto cliente como servidor.

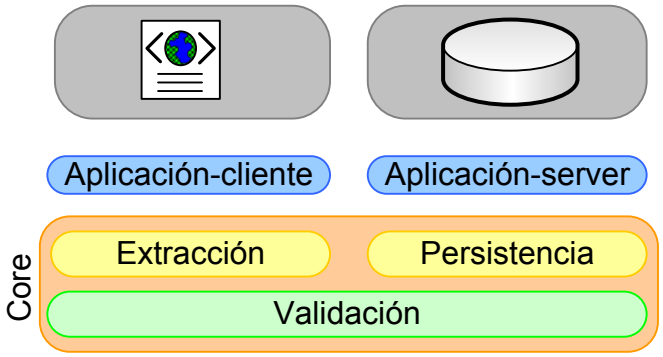


Figura 5.7 - Componentes

En las siguientes secciones se detallarán las capas de componentes que se muestran en la figura 5.7, indicando su uso y la relación.

5.4.1. Extracción

Para simplificar el uso del componente de extracción, se adaptó su configuración a través de un documento XML. Donde la sintaxis se explica en el apéndice Lenguaje de configuración[17]. Este componente cuenta con la clase `DataGenerator`[18] la cual contiene los métodos necesarios para la generación de archivo XML, el método principal `getDataXmlFromDB`, recibe como parámetro el archivo de configuración donde toma los datos necesarios para la extracción.

```
DataGenerator dataGenerator = new DataGenerator();
String dataFile = dataGenerator.getDataXmlFromDB(fileConfig);

System.out.println("Archivo de registros XML");
System.out.println(dataFile);
```

Código 5.1 - Código de Extracción

El componente de extracción es utilizado por la aplicación-cliente para envío de archivos de registros, la cual se describirá en los siguientes apartados.

[17] Consultar Apéndice A
[18] Se aplica el patrón Controller, especificado en las referencia [11, pp. 237]

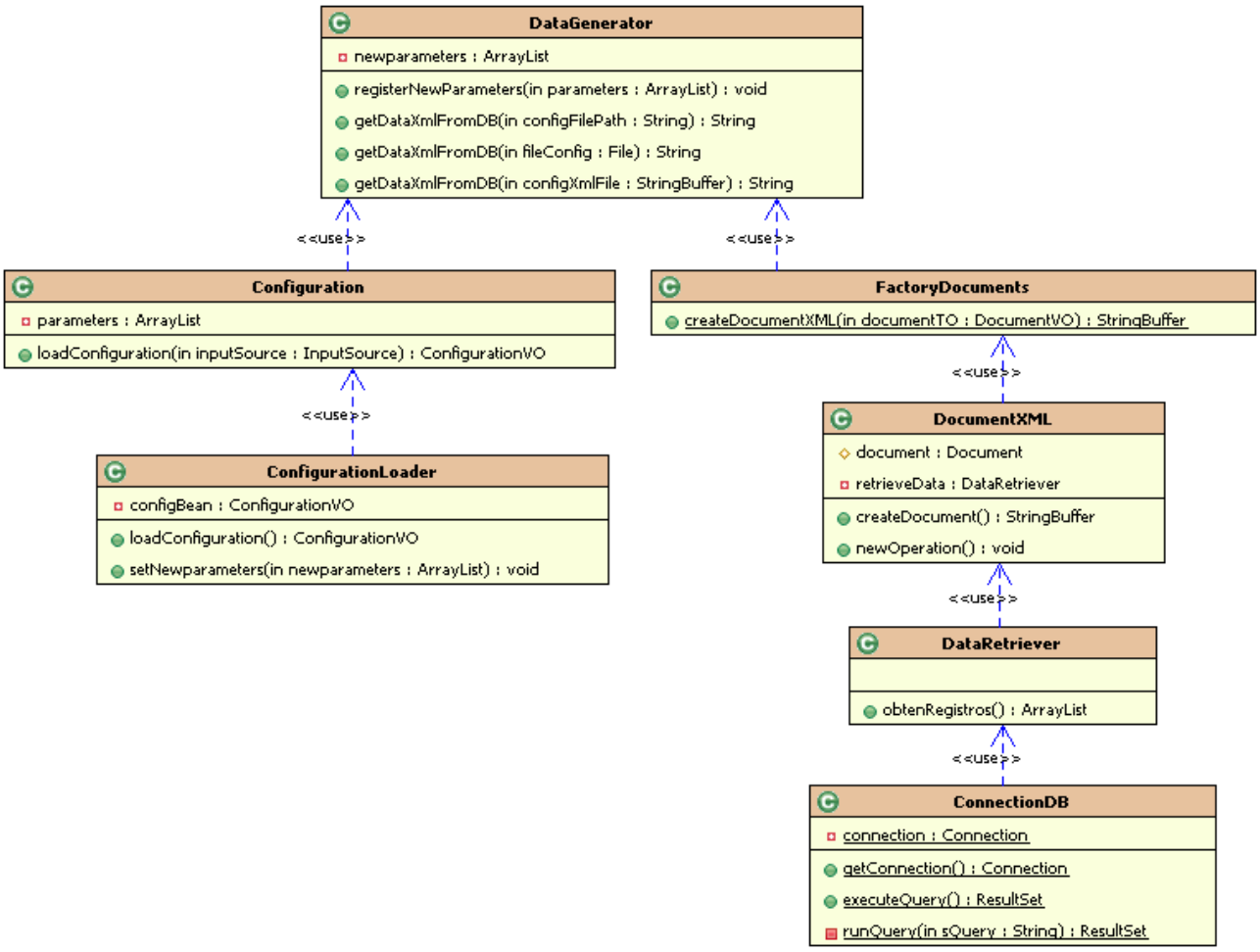


Figura 5.8 - Diagrama de clases de módulo de extracción

En la figura 5.8 se muestra un diagrama con las principales clases y métodos utilizados para el proceso de extracción.

- DataGenerator: coordina los componentes para la generación de documentos XML.
- Configuration: encapsula la funcionalidad del componente que contiene la configuración para la aplicación.
- ConfigurationLoader: extiende la funcionalidad de un lector XML basado en DOM para la obtención de argumentos del archivo de configuración, adaptando el mismo a una jerarquía de clases.
- FactoryDocuments: separa la funcionalidad para la creación de documentos, envolviendo la clase DocumentXML.
- DocumentXML: contiene la funcionalidad para la creación de documentos mediante la implementación de DOM, basándose en los objetos de configuración y la obtención de datos de la base de datos indicada.

- DataRetriever: obtiene la conexión a la base de datos configurada, realiza la consulta y la transfiere a una colección de objetos registros.

- ConnectionDB: componente encargado de realizar la conexión a la base de datos, además contiene métodos para realizar las consultas y el registro de parámetros.

5.4.2. Validación

La validación se lleva a cabo mediante el componente `Validator`, el cual realiza la verificación del archivo XML que se le indique, contra el Schema XML que tenga referenciado, invocando su método `validateDocument` que recibe como parámetro el flujo hacia el archivo de registros a validar.

La clase `Validator` contiene una propiedad `ValidatorHandler` que registra los errores ocurridos durante la validación del archivo XML, esta clase implementa la interfase `ErrorHandler` que pertenece al API SAX para el manejo de errores.

```
Validator validator = new Validator();
validator.validateDocument(dataFile);

ValidatorHandler handler = validator.getHandler();

if (handler.isValid()) {
      System.out.println("Archivo Valido");
} else {
      System.out.println("Archivo Invalido");
      System.out.println("Tipo : "+     handler.getTypeError());
      System.out.println("Error en : "+ handler.getLineNumber() +","+
                                         handler.getColumnNumber());
      System.out.println("Mensaje :"+   handler.getMessage());
}
```

Código 5.2 - Código de validación

La validación se realizará contra un Schema XML publicado en la siguiente ruta de la máquina local, la cual pertenece al módulo web de la aplicación, donde en la carpeta de contexto se encuentra el directorio "*config"* con los documentos de configuración para la aplicación:

http://localhost:8888/BuroCreditoServerWeb/config/buroschema.xsd

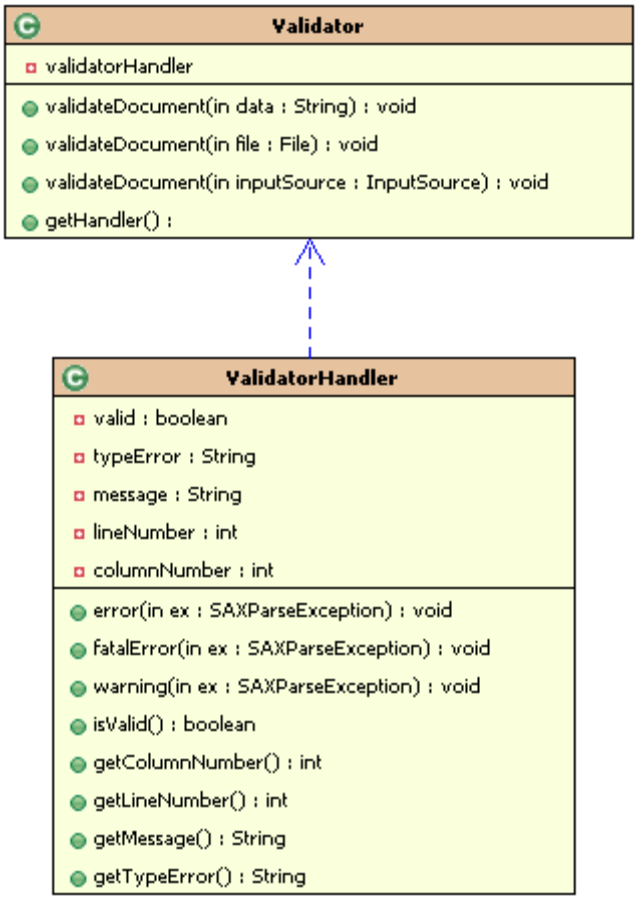


Figura 5.9 - Diagrama de clases de módulo de validación

En la figura 5.9 se muestra un diagrama con las principales clases y métodos utilizados para el proceso de validación.

- Validator: componente que implementa e inicializa propiedades para el lector XML de un documento con validación, registrando el componente para cachado de los errores.

- ValidatorHandler: implementación de un ErrorHandler de SAX, para el registro de errores durante el procesamiento de un archivo XML.

De igual forma que el proceso de extracción, la validación se ejecuta en la aplicación-cliente previo al envío del archivo de registros, evitando que se traten de persistir archivos erróneos o información que no cumpla con los formatos establecidos en los documentos publicados del Formato INTF.

5.4.3. Carga

Para la persistencia de los registros contenidos en los archivos XML, la librería Core contempla las clases `GenericDOMParser` y `GenericSAXParser` que extienden la funcionalidad de DOM y SAX, agregando métodos para la localización de elementos y utilerías para el tratamiento de los archivos en el sistema.

No se dio más funcionalidad a las clases contenidas en la librería Core para la persistencia de registros del negocio, ya que son utilizadas en la lectura de archivos XML para otros fines, manteniendo la responsabilidad de conocer y generar los registros del negocio al módulo correspondiente en la aplicación del servidor.

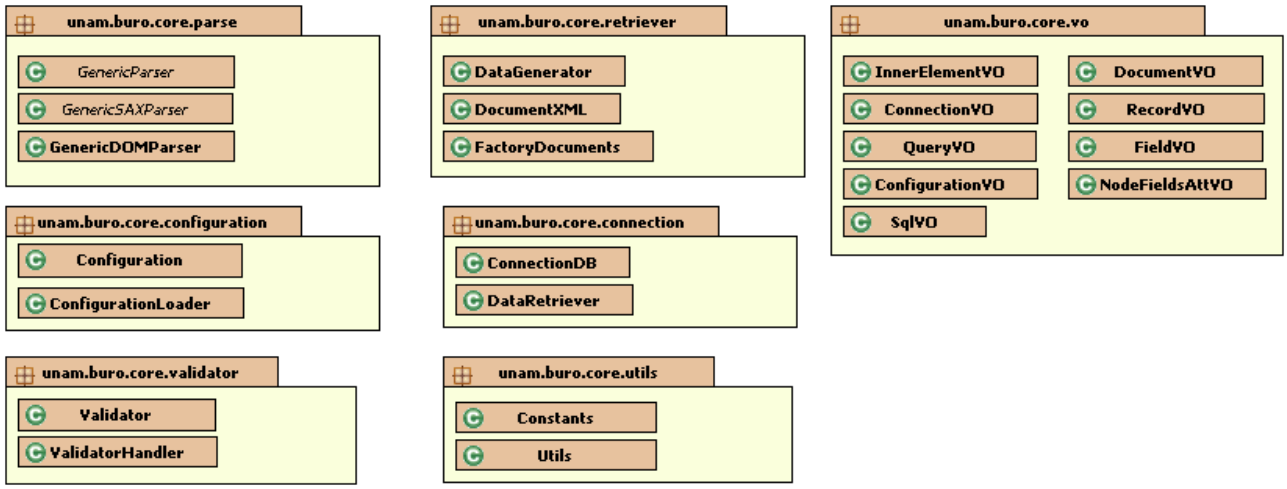


Figura 5.10 - Paquetes y clases de componentes Core

En la figura 5.10 se muestra los paquetes que conforman las librerías Core utilizadas por la aplicación-cliente y el proceso de persistencia de datos del lado del servidor.

5.5. Proceso de Extracción

En la figura 5.11 se muestra parte del esquema de base de datos que se utiliza actualmente por una Entidad Financiera determinada que contiene los datos para generar el archivo de registros. El sistema cuenta con diversos módulos y tablas de infraestructura, sin embargo, para fines de este proyecto se contemplan sólo las tablas para generar el archivo de personas físicas.

El sistema de la Entidad Financiera contempla cuatro principales tablas para personas físicas, las cuales se llenan a través de procesos[19] que recuperan y calculan los datos requeridos, dichos datos se encuentran almacenados en diversas tablas dentro de la misma base y los cálculos son muy complejos, apegados a las reglas del negocio, para

[19] Stored Procedures, Funciones, Triggers desarrollados en el lenguaje PL/SQL de la base de datos Oracle

simplificar la extracción se contemplaron las tablas mencionadas que serán las utilizadas para la aplicación.

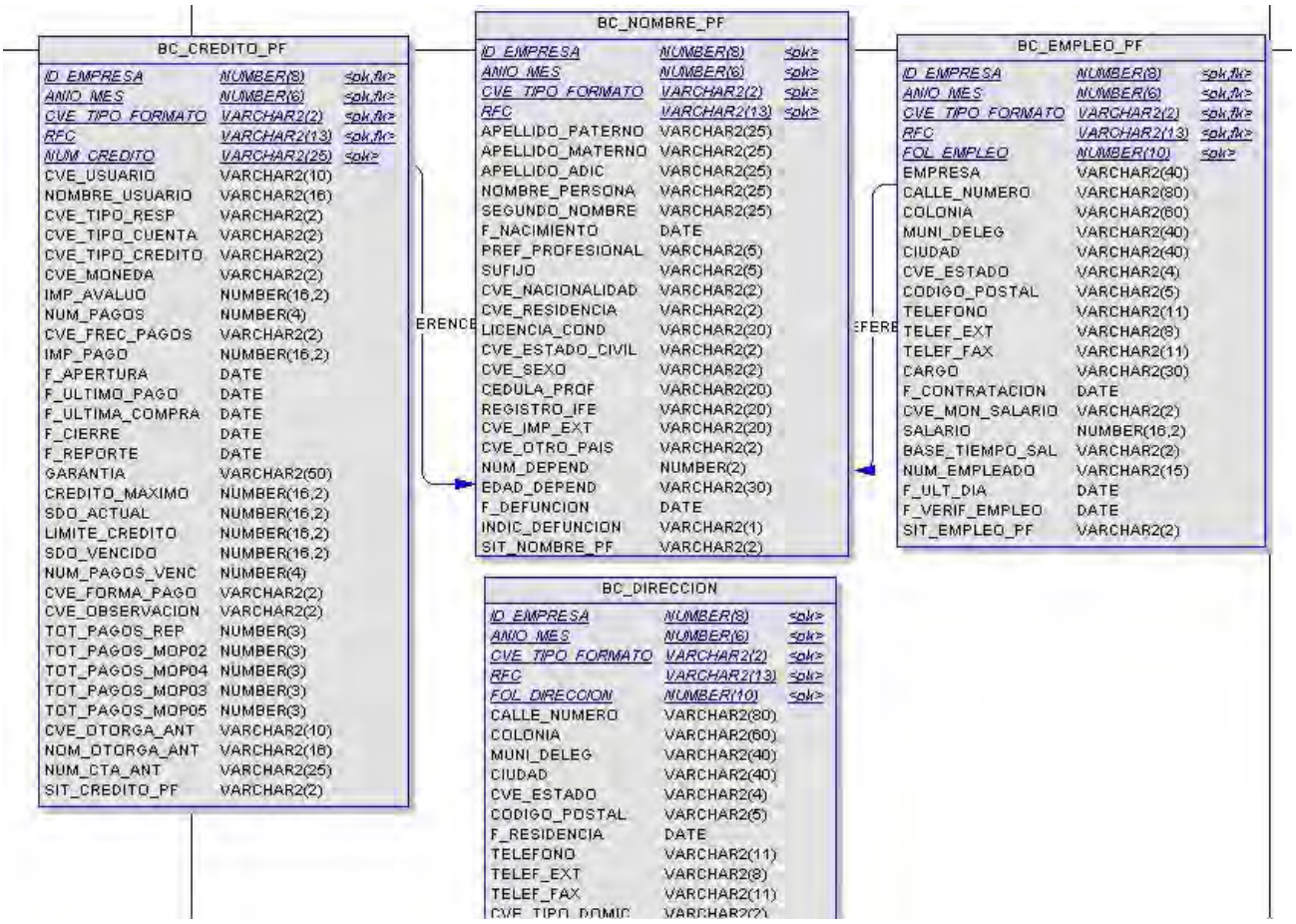


Figura 5.11 - Esquema de la base de datos de Entidad Financiera

La figura 5.11 muestra el esquema de tablas donde se almacenan los datos para la generación de archivo de registros.

- BC_NOMBRE_PF: contiene los registros de los consumidores, clientes de la entidad financiera bajo el régimen de Persona Física, contiene campos para datos personales además de procedencia la persona consumidor.

- BC_CREDITO_PF: almacena los datos crediticios de cada uno de las personas físicas, sus importes y fechas de movimientos, así como descriptores del tipo de cuenta.

- BC_EMPLEO_PF: contiene el domicilio del empleo por cada persona física, así como detalles sobre la contratación y contactos.

- BC_DIRECCION: almacena la información del domicilio fiscal y medios de contacto del consumidor, esta tabla se utiliza para personas físicas, personas morales y créditos hipotecarios.

En el ambiente de desarrollo se implemento la misma estructura de tablas a una base de datos sobre PostgreSQL destinada para la extracción, la información que contienen las tablas, es información real de la Entidad bajo estudio, poblada previamente en la base de datos.

Cada registro dentro del archivo generado en formato INTF, se distingue mediante la identificación de cada uno de los campos que lo componen, los cuales están numerados para distinguir el comienzo y fin de los registros. Para distinguir los campos, éstos deben de encontrarse ordenados y numerados además de indicar la longitud del valor.

Adaptando la información a un formato XML, la identificación de los registros, así como los campos y sus valores es simplificada, facilitando los procesos de persistencia de datos.

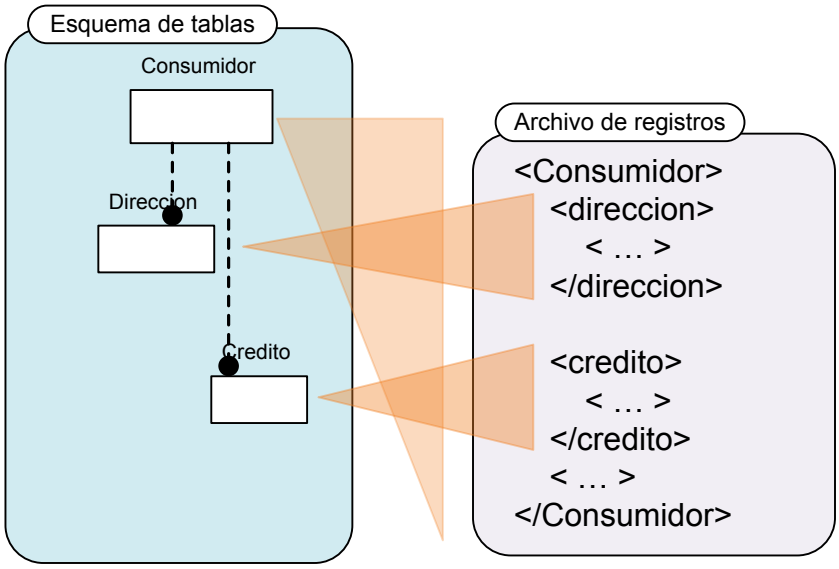


Figura 5.12 – Mapeo de tablas a XML

5.5.1. Archivo de Configuración para Extracción de datos

Una vez vista la relación entre las tablas de la base de datos, se genera el documento de configuración, indicando la consulta respectiva en la forma más conveniente.

El siguiente ejemplo de configuración mostradó por el código 5.1, esta basado en la consulta a la base de datos para extracción, solamente se muestra un fragmento del documento, en el apéndice A se puede consultar el documento completo, además de la sintaxis y explicación de cada elemento que lo componen.

```
<wsdbxml-config>

   <documents-xml>
      <doc-xml name="DOC_CONSUMIDOR" root="CONSUMIDORES">

        <query-fields record-name='CONSUMIDOR_BURO'>
            <sql>
              <select>
                SELECT NOM.APELLIDO_PATERNO AS CS_APATERNO,
                       NOM.APELLIDO_MATERNO AS CS_AMATERNO,
                       NOM.APELLIDO_ADIC    AS CS_AADICIONAL,
                      <inner_element name="DOMICILIO_BURO">
                       DIR.CALLE_NUMERO   AS DM_CALLE,
                       .. .. ..
                      </inner_element>
                     <inner_element name="EMPLEO_BURO">
                       EMP.ID_EMPRESA  AS EM_RAZON_SOCIAL,
                       EMP.CALLE    AS EM_CALLE,
                       .. .. ..
                      </inner_element>
                      <inner_element name="LINEA_CREDITO_BURO">
                       CRE.CVE_USUARIO      AS LC_CVE_USUARIO,
                       .. .. ..
                      </inner_element>
              </select>
              <from>
                FROM BC_NOMBRE_PF AS NOM
                     LEFT JOIN BC_EMPLEO_PF AS EMP
                     ON EMP.ID_EMPRESA = NOM.ID_EMPRESA
                      AND EMP.ANIO_MES = NOM.ANIO_MES
                      AND EMP.CVE_TIPO_FORMATO = NOM.CVE_TIPO_FORMATO
                      AND EMP.RFC = NOM.RFC
                      .. .. ..
              </from>
              <where>
                WHERE NOM.ID_EMPRESA = 2
                  AND NOM.ANIO_MES = '200502'
                  AND NOM.CVE_TIPO_FORMATO = 'PF'
              </where>
            </sql>

         </query-fields>
      .. .. ..
      </doc-xml>
   </documents-xml>
</wsdbxml-config>
```

Código 5.3 - Documento de Configuración para componente de extracción

El documento de configuración cuenta con elementos para seccionar una consulta previamente construida hacia la base de datos indicada, los nombres o alias de los campos serán los asignados a los elementos del archivo resultado, además se cuenta con el elemento *inner_element* que permite agrupar campos los cuales conformaran un sub-elemento indicado por el atributo *name*. Los archivos resultantes de esta configuración sólo tendrán dos niveles de animación lo cual es suficiente para los requerimientos y organización de los datos.

5.6. Proceso de Envío - Aplicación cliente

Para el proceso de transferencia, se propone que Buró de crédito ponga a disposición una aplicación dedicada a la validación y transferencia de los archivos de registros generados por las entidades financieras, con el fin de facilitar el cumplimiento del trámite.

Dicha aplicación incorpora el componente de extracción, en caso que la consulta hacia el esquema de base de datos de una determinada entidad financiera sea adaptable al documento de configuración.

Previo al envío, realiza la validación sobre el archivo de registros contra el esquema publicado en el sitio de Buró de crédito.

El envío se realiza a través del protocolo SFTP[20], hacia un repositorio destinado para las entidades, garantizando la confidencialidad de los datos durante la transferencia, mediante éste canal seguro.

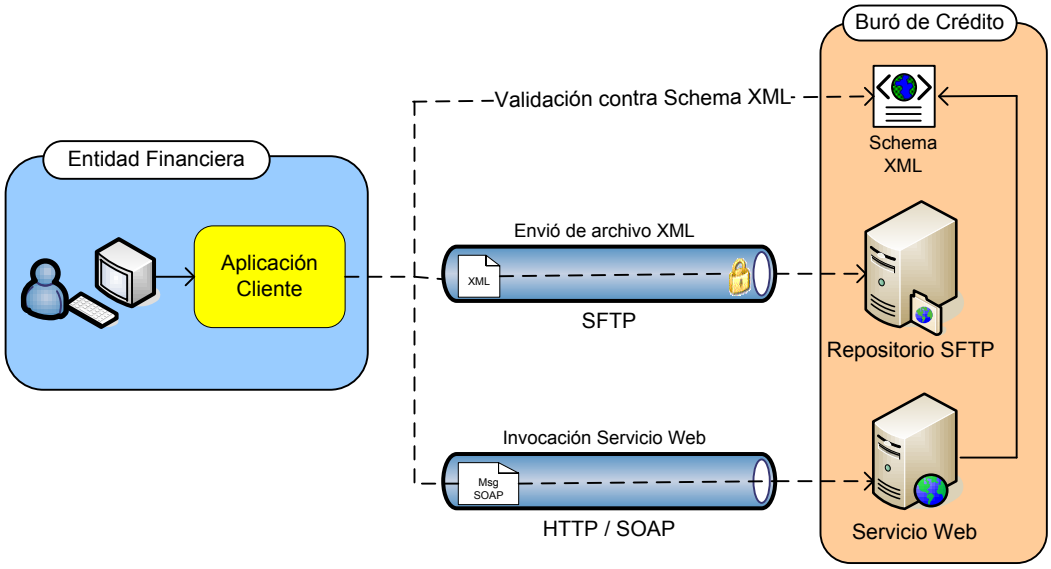


Figura 5.13 - Flujo de envío, validación y notificación

La aplicación-cliente coordina la comunicación con el servidor, mediante la invocación del servicio web para la notificación de la transferencia del archivo de registros. Este servicio web encola dicha notificación hacia el proceso de persistencia de los registros contenidos, dicho proceso se ejecuta de manera asíncrona por lo cual es posible que diversas entidades realicen el envío de manera simultánea.

[20] Véase http://es.wikipedia.org/wiki/Sftp

5.6.1 Uso de Aplicación Cliente de envío

Al ejecutar la aplicación-cliente que se muestra en la figura 5.8, el primer paso es indicar la clave de la entidad financiera que es asignada por Buró de crédito al dar de alta la entidad en su sistema de administración.

La aplicación-cliente presenta dos modos para leer el archivo de registros:

- el primero lee un archivo de registros XML generado con anterioridad mediante alguna herramienta de la propia entidad financiera
- el segundo modo procesa el archivo de configuración que cumple con el lenguaje de consultas proporcionado por Buró de Crédito[21], para la extracción de los registros en la base de datos de la entidad financiera y aplicar el formato a XML

Si el archivo procesado es de configuración, se realiza la mapeo del contenido a un componente de configuración que será utilizado para la extracción de los datos. Una vez procesado, se habilita el botón “Generar”, que iniciará el proceso de extracción.

La validación del lenguaje se lleva a cabo previo al proceso de envío, en caso de que no cumpla con el lenguaje establecido, se notificará mediante un mensaje en la pantalla de la aplicación.

La validación y envío del archivo de registros, que se lleva a cabo al presionar el botón de “Enviar”. Se inicia la validación, si el archivo es válido comienza el envió a través de SFTP hacia el repositorio asignado a las entidades financieras.

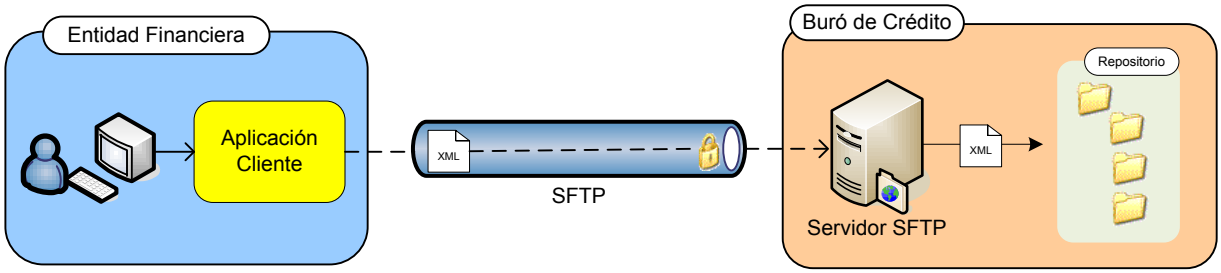


Figura 5.14 - Modelo de envío de archivos al repositorio

[21] Consultar Apéndice A

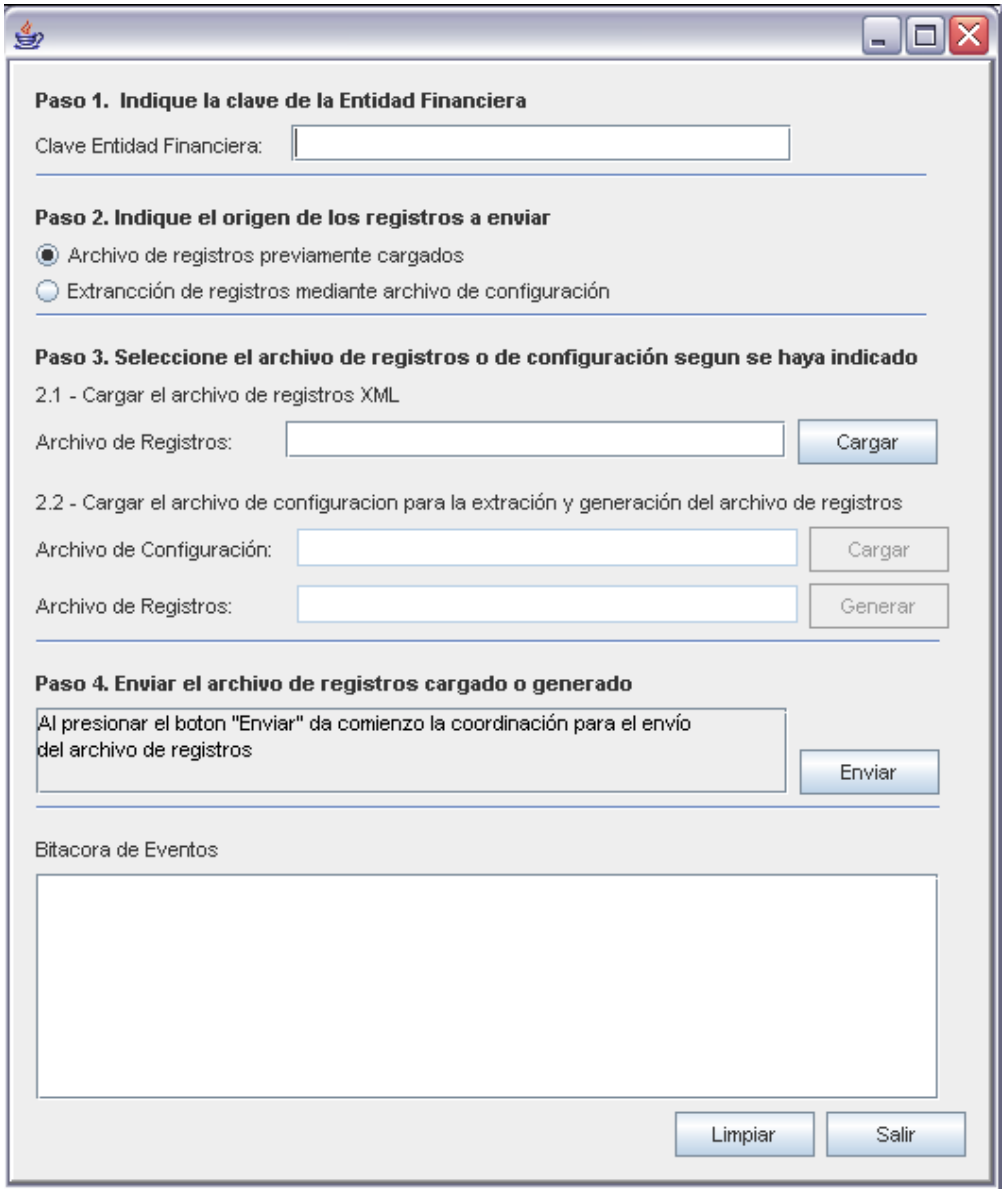


Figura 5.15 - Aplicación Cliente de Envío de Archivos de Registros XML

Al finalizar el flujo mencionado, la aplicación indica que es posible cerrarla con un mensaje en el control de eventos.

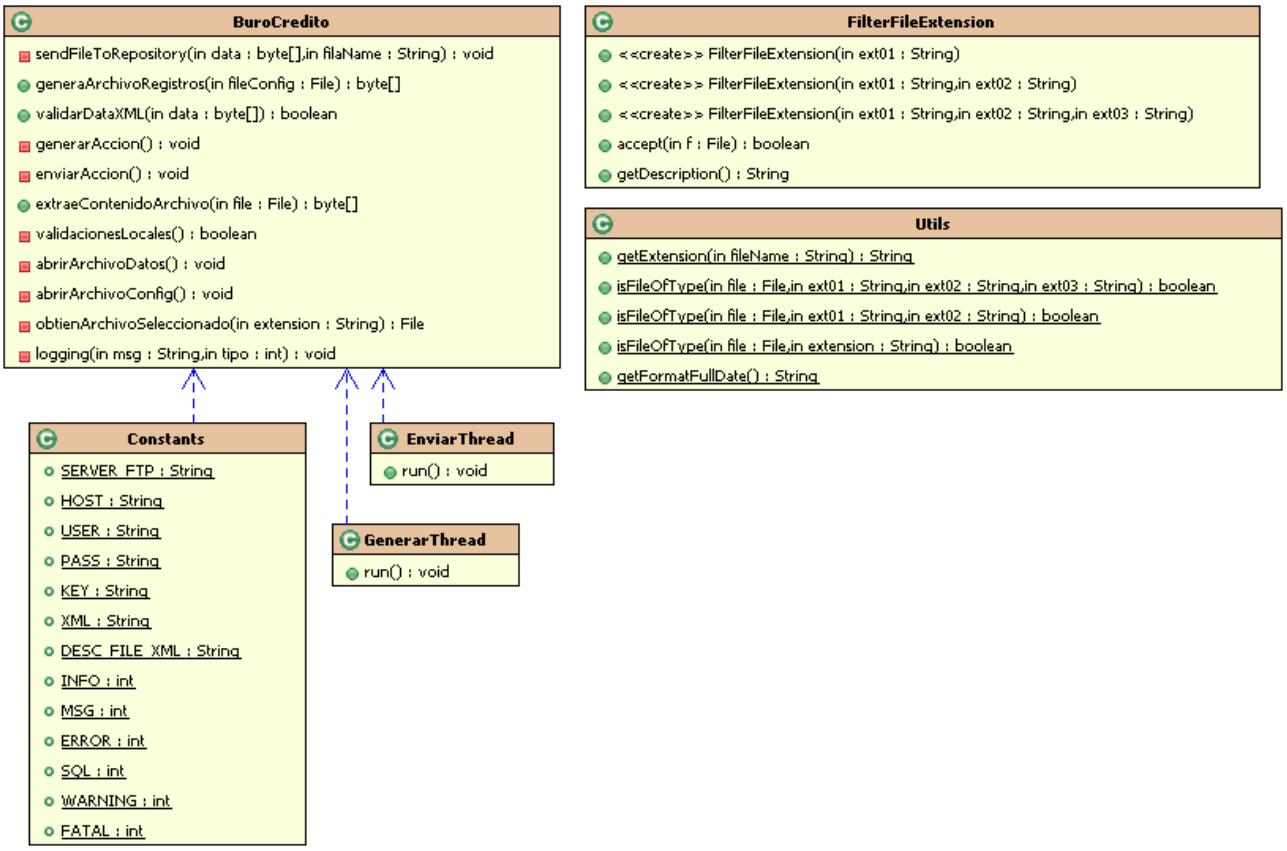


Figura 5.16 - Diagrama de clases de aplicación de envío de archivos

En la figura 5.16 se muestran las clases que conforman la aplicación-cliente para envío de archivos de registros.

- BuroCredito: construye la parte visual de la aplicación basada en Java Swing, contiene métodos correspondientes para la generación de los archivos, coordina el envío de archivos al repositorio y la notificación al servicio web para iniciar el procesamiento del archivo indicado.

- FilterFileExtension: clase para el filtrado de archivos a procesar, verifica las extensiones permitidas por la aplicación.

- Utils: contiene métodos de utilerías y validaciones

5.6.2. Estructura de Archivos Generados

En la figura 5.17 se muestra un archivo de registros XML, el cual esta vinculado al esquema para su validación de estructura, dicho Schema XML[22] se desarrollo para que contemplara las mismas condiciones que requiere buró de crédito en el formato INTF, con la facilidad de que éste estará publicado en el sitio web de Buró de Crédito, disponible para las validaciones pertinentes por cada entidad financiera.

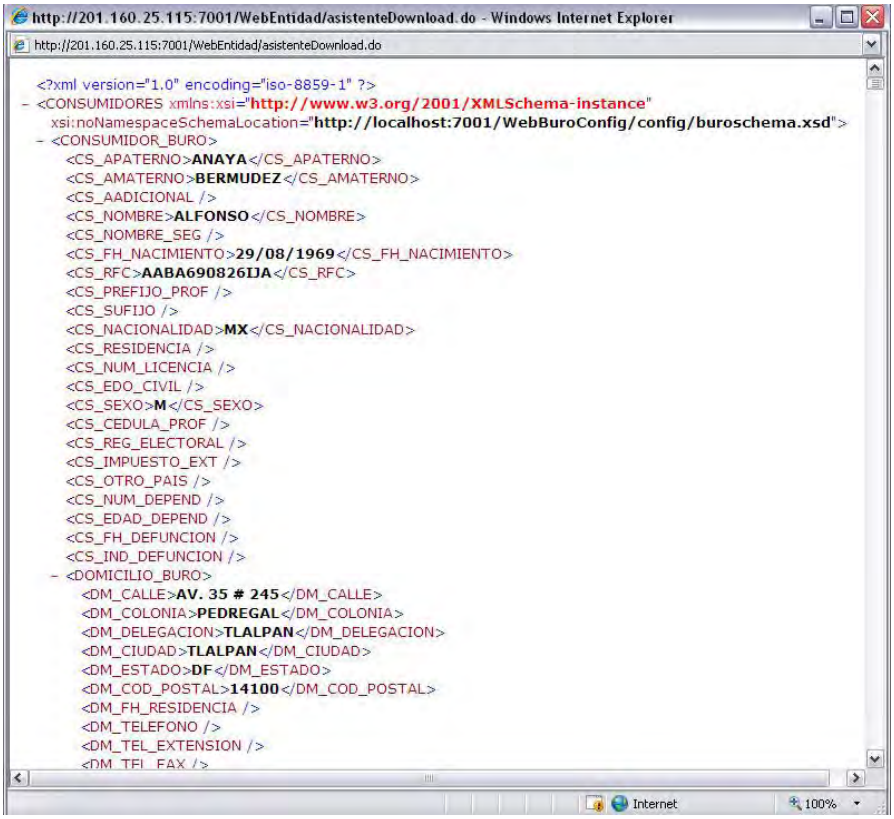


Figura 5.17 - Documento XML Generado por el Componente Cliente

La ventaja de utilizar un Schema XML, es romper con el ciclo de pruebas durante el envió del archivo de registros. Así cada Entidad Financiera puede validar los archivos generados antes de realizar su envío, cuando éstos cumplan con el esquema se garantiza la persistencia exitosa de los registros contenidos. Ahorrando tiempo en la corrección de procesos, regeneración y envió de los archivos.

[22] Véase Apéndice E

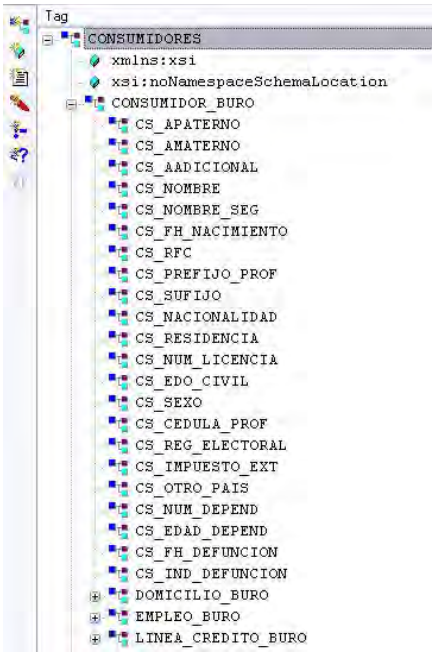


Figura 5.18 - Estructura de Árbol de Archivo de Registros XML

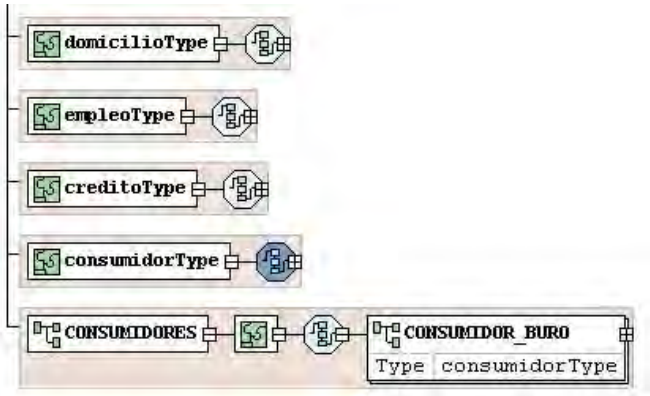


Figura 5.19 - Estructura principal del Schema XML de validación

La figura 5.19 muestra la estructura que valida el Schema XML para los archivos de registros, como ya se ha mencionado, el esquema esta basado para los registros de personas físicas con los elementos requeridos en el formato INFT.

Como parte de la librería Core, se encuentra un componente de validación que recibe un documento XML como argumento y realiza la validación contra el esquema que tiene referenciado, registrando los errores que detecte para su correspondiente corrección.

5.6.3 Servicio Web Cliente

La figura 5.20 muestra las clases generadas para el cliente del servicio web, mediante el cual se notifica el envío de archivo de registros por una entidad financiera.

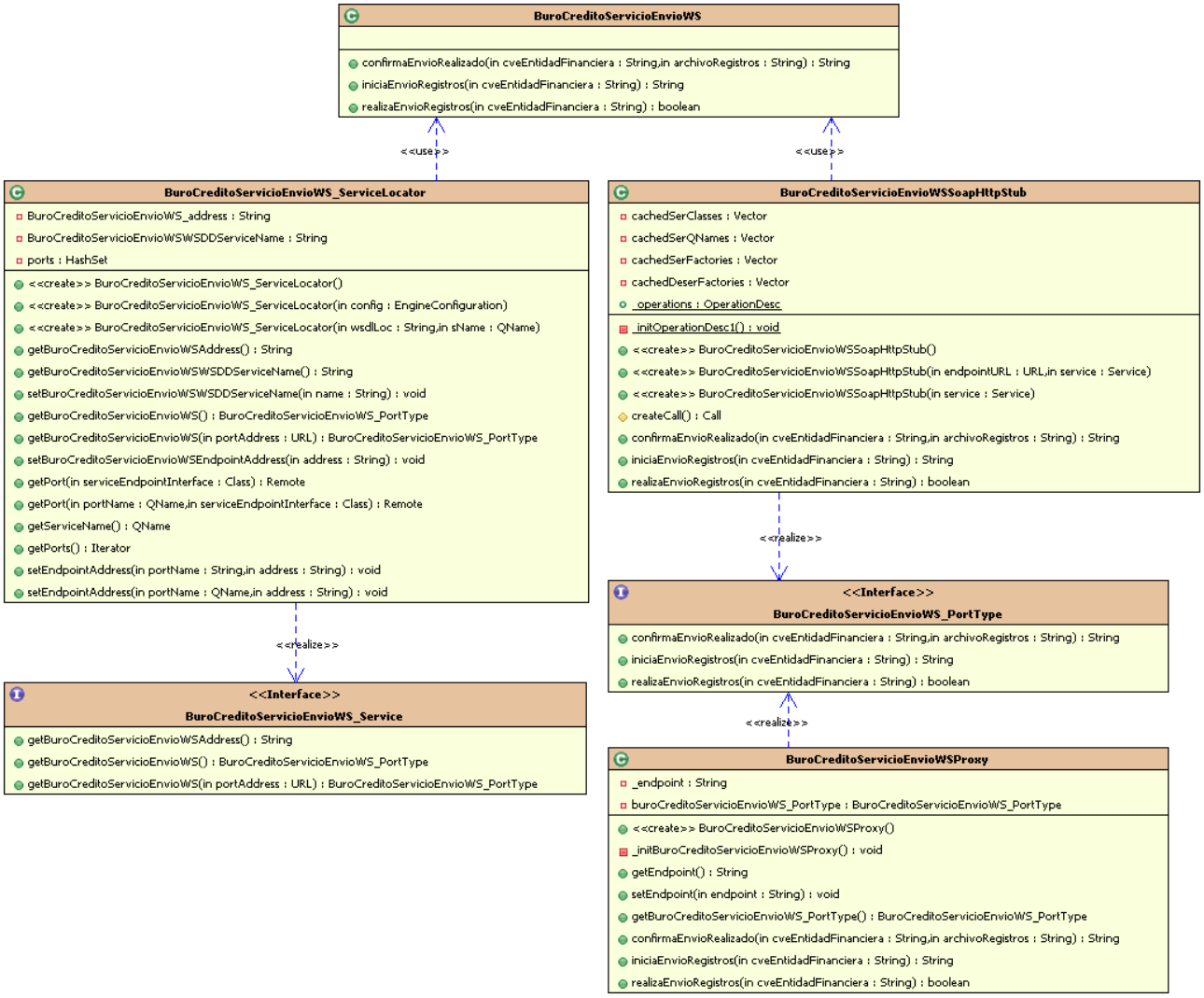


Figura 5.20 - Diagrama de clase de Cliente de Servicio Web

- BuroCreditoServicioEnvioWS: componente de envoltorio para la invocación de los métodos publicados por el servicio web

- BuroCreditoServicioEnvioWSSoapHttpStub: componente “stub” que representa la clase que contiene los métodos publicados como servicios web, realiza la comunicación con el servicio web a través del protocolo HTTP.

- BuroCreditoServicioEnvioWS_ServiceLocator: componente que implementa el patros Service Locator, encargado de la localización de los recursos remotos.

- BuroCreditoServicioEnvioWSProxy: Mantiene una referencia al objeto real, controla la creación y acceso a las operaciones del mismo.

Cuando la aplicación es desplegada, el servidor genera un descriptor (WSDL) del componente que es publicado como servicio web, dicho descriptor se muestra en el código 5.4.

5.7. Proceso de Carga de registros

Una vez recibido un archivo de registros XML en el repositorio SFTP, la aplicación-cliente realiza su correspondiente notificación mediante el servicio web publicado. Dicho servicio, recibe como argumentos la clave de la entidad financiera y el nombre del archivo de registros enviado.

Al ser invocado el servicio web, éste se encarga de enviar un mensaje a la cola de mensajes para el proceso de persistencia de datos, con los argumentos recibidos. Se optó por el uso de una cola de mensajes en el proceso de persistencia para facilitar el manejo de concurrencia en las solicitudes de procesamiento de los archivos de registros. Es decir, en dado caso que diversas entidades envíen sus archivos y realicen las notificaciones, al invocar el servicio, cada solicitud se encola a espera de su turno para ser procesado.

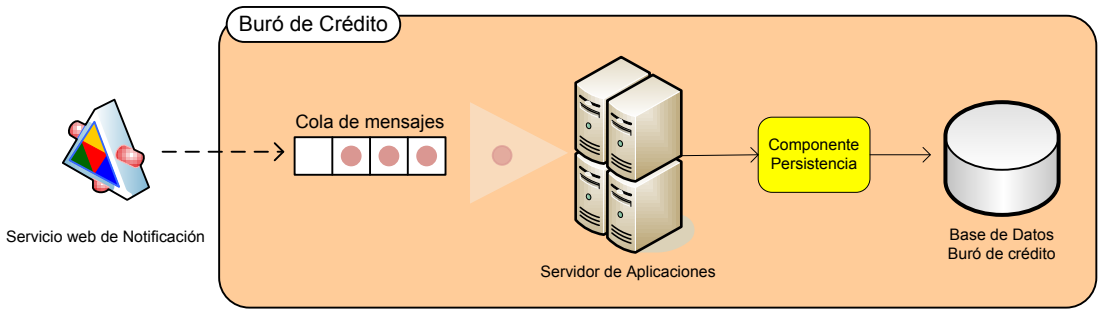


Figura 5.21 - Modelo de administración de procesos

Como una mejora posterior en el performance de la aplicación, se puede evaluar la implementación de un Cluster de computadoras para que realicen el procesamiento de los archivos de forma paralela, que diversos componentes consuman los mensajes de la cola de procesamiento.

Cuando llega el turno de un archivo de registros a ser procesado para el poblado de los datos en la base de datos de Buró de Crédito, se obtiene físicamente del repositorio de archivos y se extraen los registros mediante el procesamiento correspondiente, conteniendo los datos en una lista de objetos (POJO's)[23], la cual se envía a un componente de persistencia para su almacenamiento. Dado que ya se realizo una validación del archivo, previo a su envió por la aplicación-cliente, se debe confiar en un

[23] Plain Old Java Object

poblado de datos exitoso. Aunque es posible una segunda validación durante la lectura del archivo al extraer los registros en este proceso.

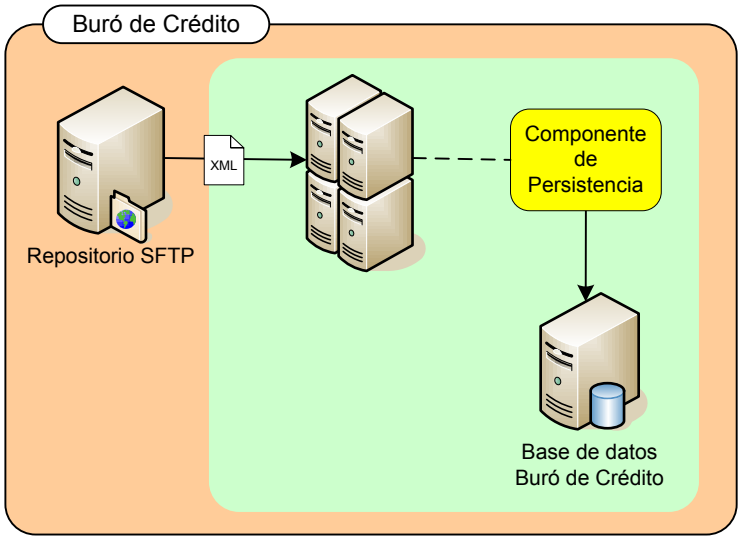


Figura 5.22 - Aplicación de lado del servidor.

5.7.1 Servicio Web para Notificación de Envío

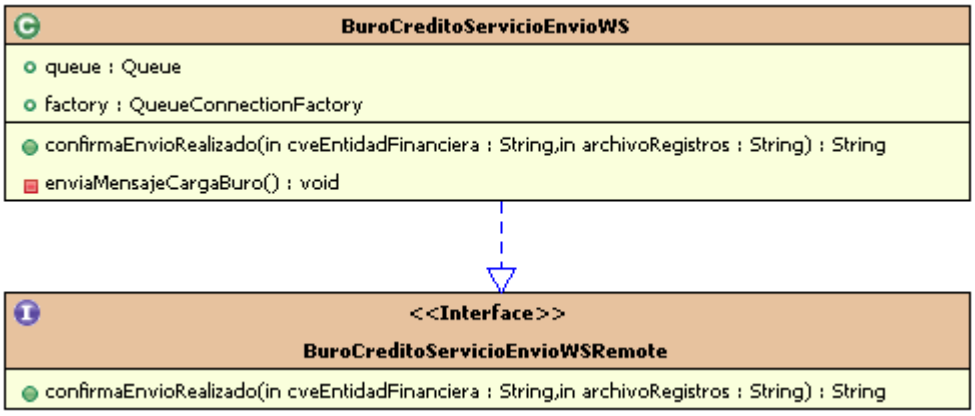


Figura 5.23 - Diagrama de clases de Servicio Web

La figura 5.23 muestra la clase publicada como servicios web y la interfase para acceso remoto, en caso de aplicación basadas en EJB's.

- BuroCreditoServicioEnvioWS: contiene el método publicado como servicio web , el cual encola la notificación en una cola de mensajes para su proceso por el componente BuroCreditoMDB.

- BuroCreditoServicioEnvioWSRemote: interfase para acceso remoto a la notificación de envío, para aplicación basadas en Java y EJB.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://unam.buro.server.ejb.session.ws/" name="BuroCreditoServicioEnvioWS"
targetNamespace="http://unam.buro.server.ejb.session.ws/">
    <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="confirmaEnvioRealizado" type="tns:confirmaEnvioRealizado"/>
    <complexType name="confirmaEnvioRealizado">
    <sequence>
    <element name="cveEntidadFinanciera" type="string" nillable="true"/>
    <element name="archivoRegistros" type="string" nillable="true"/>
    </sequence>
    </complexType>
    <element name="confirmaEnvioRealizadoResponse"
type="tns:confirmaEnvioRealizadoResponse"/>
    <complexType name="confirmaEnvioRealizadoResponse">
    <sequence><element name="return" type="string" nillable="true"/></sequence>
    </complexType>
    </schema>
    </types>
    <message name="BuroCreditoServicioEnvioWSPortType_confirmaEnvioRealizado">
        <part name="parameters" element="tns:confirmaEnvioRealizado"/>
    </message>
    <message name="BuroCreditoServicioEnvioWSPortType_confirmaEnvioRealizadoResponse">
        <part name="parameters" element="tns:confirmaEnvioRealizadoResponse"/>
    </message>
<portType name="BuroCreditoServicioEnvioWS">
<operation name="confirmaEnvioRealizado">
    <input message="tns:BuroCreditoServicioEnvioWSPortType_confirmaEnvioRealizado"/>
<output message="tns:BuroCreditoServicioEnvioWSPortType_confirmaEnvioRealizadoResponse"/>
</operation>
</portType>
<binding name="BuroCreditoServicioEnvioWSSoapHttp" type="tns:BuroCreditoServicioEnvioWS">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="confirmaEnvioRealizado">
  <soap:operation soapAction=""/>
  <input><soap:body use="literal"/></input>
  <output><soap:body use="literal"/></output>
  </operation>
  </binding>
  <service name="BuroCreditoServicioEnvioWS">
<port name="BuroCreditoServicioEnvioWS" binding="tns:BuroCreditoServicioEnvioWSSoapHttp">
<soap:address
location="http://localhost:8888/BuroCreditoServerEJB/BuroCreditoServicioEnvioWS"/>
</port>
</service>
</definitions>
```

Código 5.4 - Descriptor de Servicio Web (WSDL)

El código 5.4 muestra un fragmento del descriptor del servicio web (WSDL)[24], generado durante el despliegue de la aplicación por el servidor de aplicaciones, que a su vez se basa en las anotaciones incluidas en el componente BuroCreditoServicioEnvioWS que contiene los métodos publicados. El componente BuroCreditoServicioEnvioWS es un EJB de sesión que contiene anotaciones específicas para ser publicado como un servicio web por el servidor de aplicaciones durante el despliegue de la aplicación. Además implementa una interfase remota para su invocación desde aplicación basadas en Java mediante RMI[25]. El manejo de EJB's se detalla en el apéndice B, sin embargo, el código 5.5 muestra la declaración del método como servicio web mediante un EJB.

[24] Véase Capitulo IV - WSDL

[25] Remote Method Invocation (Invocación remota de métodos)

```
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;


@Stateless
@WebService(serviceName=Constants.WS_SERVICE_NAME,
portName=Constants.WS_PORT_NAME)
public class BuroCreditoServicioEnvioWS implements
BuroCreditoServicioEnvioWSRemote {

   @WebMethod
   public String confirmaEnvioRealizado(String cveEntidadFinanciera,
                                        String archivoRegistros) throws
Exception {

      .. .. ..
   }

   private void enviaMensajeCargaBuro(MensajeBuroVO vo) throws Exception {
    .. .. ..
   }
}
```

Código 5.5 - EJB de Session publicado como Servicio Web

5.7.2 Componentes del Proceso de Carga

En la figura 5.24 se muestran los componentes y los métodos relevantes que conforman el proceso de persistencia de datos.

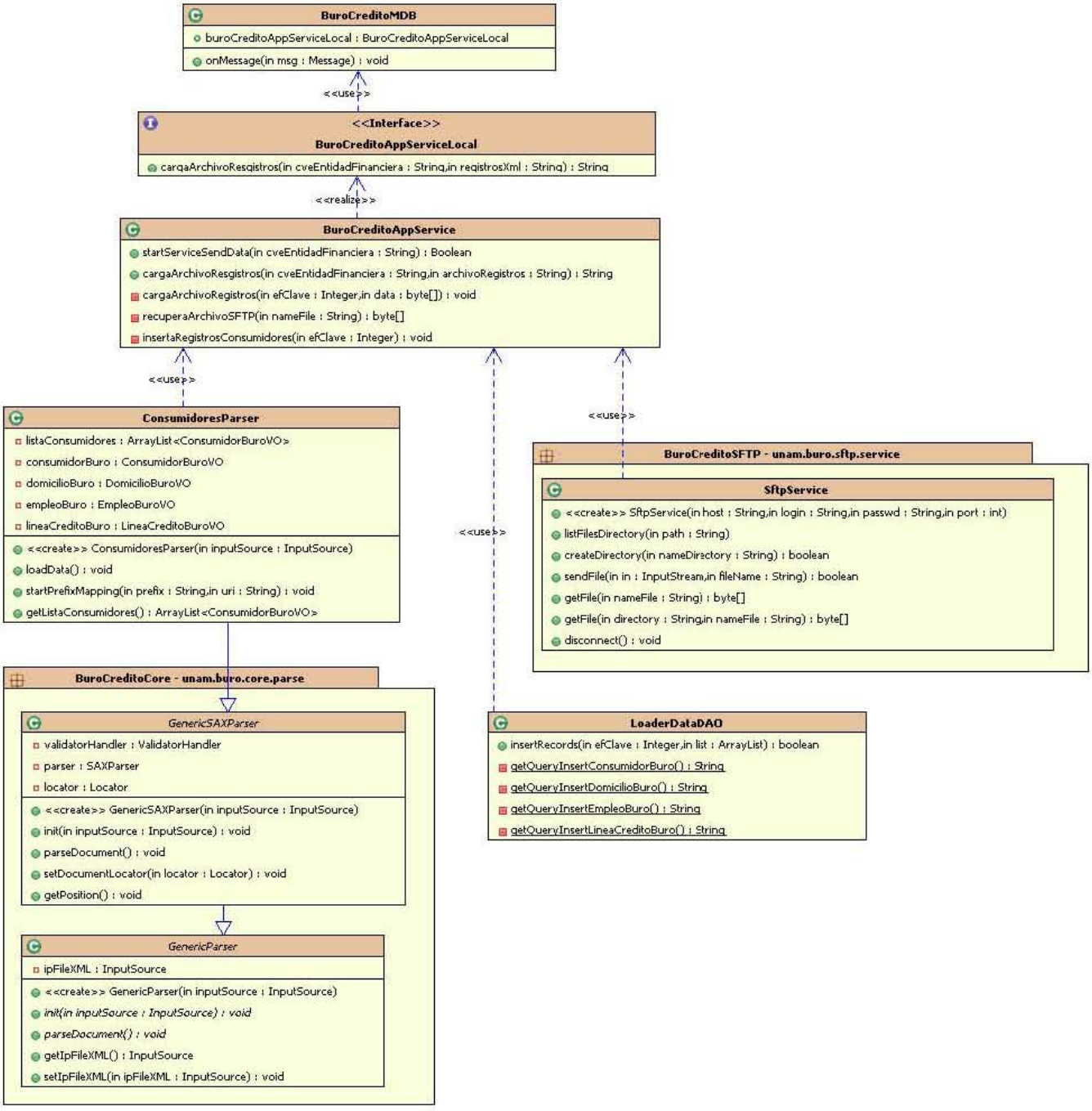


Figura 5.24 - Diagrama de clases de módulo de poblado de datos

- BuroCreditoMDB: componente basado en mensajes (Message Driven Bean), consume los mensajes encolados para su correspondiente procesamiento, el componente publicado como servicio web, encola las notificaciones de las entidades, con eso se controla la coalición de notificaciones, además es posible tener diversos componentes atendiendo la cola de mensajes para agilizar la carga de diversos archivos en paralelo.

- BuroCreditoAppService: contiene la lógica de negocio y requerimientos, siendo un EJB de session, el contenedor de EJB's del servidor de aplicaciones administra la concurrencia y el manejo de transacciones, además de crear un pool de instancias lo que da la posibilidad de atender diversas llamadas simultaneas sin que interfieran entre ellas.

- ConsumidoresParser: este componente extiende la funcionalidad de un lector SAX, mapea el archivo de registros hacia una colección de objetos que representan los consumidores con su registro de línea de crédito, empleo y domicilio.

- SftpService: componente que proporciona el servicio de conectividad hacia el repositorio del servidor SFTP, se utiliza para la obtención del archivo enviado por una determinada entidad financiera para su persistencia a la base de datos de Buró de Crédito.

- LoaderDataDAO: realiza el almacenamiento de los registros extraídos del archivo enviado, mediante una inserción en batch hacia la base de datos utilizando JDBC. Ya que el contenedor de EJB's crea un pool de instancias con cierto limite de tamaño, una carga masiva de registros utilizando componentes EJB's de entidad, haría necesaria la creación de instancias por cada registro, por lo que se decide implementar una inserción de los datos en *batch*, evitando consumir demasiados recursos.

Los componentes del proceso de persistencia están organizados de acuerdo a su rol dentro del proceso mismo, la figura 5.25 muestra los patrones y capas bajos los cuales esta diseñada la aplicación del servidor.

- Un actor externo al sistema inicia el proceso de persistencia invocando el servicio web.

- El servicio web al ser invocado encola el procesamiento del archivo enviando un mensaje hacia la cola de mensajes configurada en el servidor.

- El componente de mensaje atiende las notificaciones almacenadas en la cola de mensajes, de manera asíncrona inicia realmente el proceso de persistencia del archivo de registro, invocando al componente en la capa de servicios de la aplicación.

- Un componente que implementa el patrón Aplication Service, centraliza las reglas de negocio, representa la capa de negocio de la aplicación, coordina los diversos

componentes involucrados y delega responsabilidades a componentes de la misma capa o inferiores.

- Un componente Business Object, encapsula una determinada regla de negocio, separa la obtención de los datos de la lógica facilitando su reutilización.

- Los componentes para acceso de datos (Data Access Object) por lo general realizan consultas u operaciones hacia la base de datos a través de JDBC, en caso particular de la aplicación, se utiliza para el poblado de datos en batch de los registros de un archivo. Sólo a través de los DAO o Facade se tiene acceso a la capa de base de datos.

- Un Facade sirve de fachada para la administración de un componente de entidad asociado directamente a un registro de la base de datos, es decir, contiene los métodos de alta, baja, cambio y consulta de uno o varios registros de una determinada tabla de la base de datos, representados por componentes de entidad.

- Un componente de entidad (Entity) representa un registro de una tabla. Dentro de los EJB, estos componentes contiene la lógica de mapeo contra la tabla especificada.

- Persistencia: Capa de almacenamiento de datos

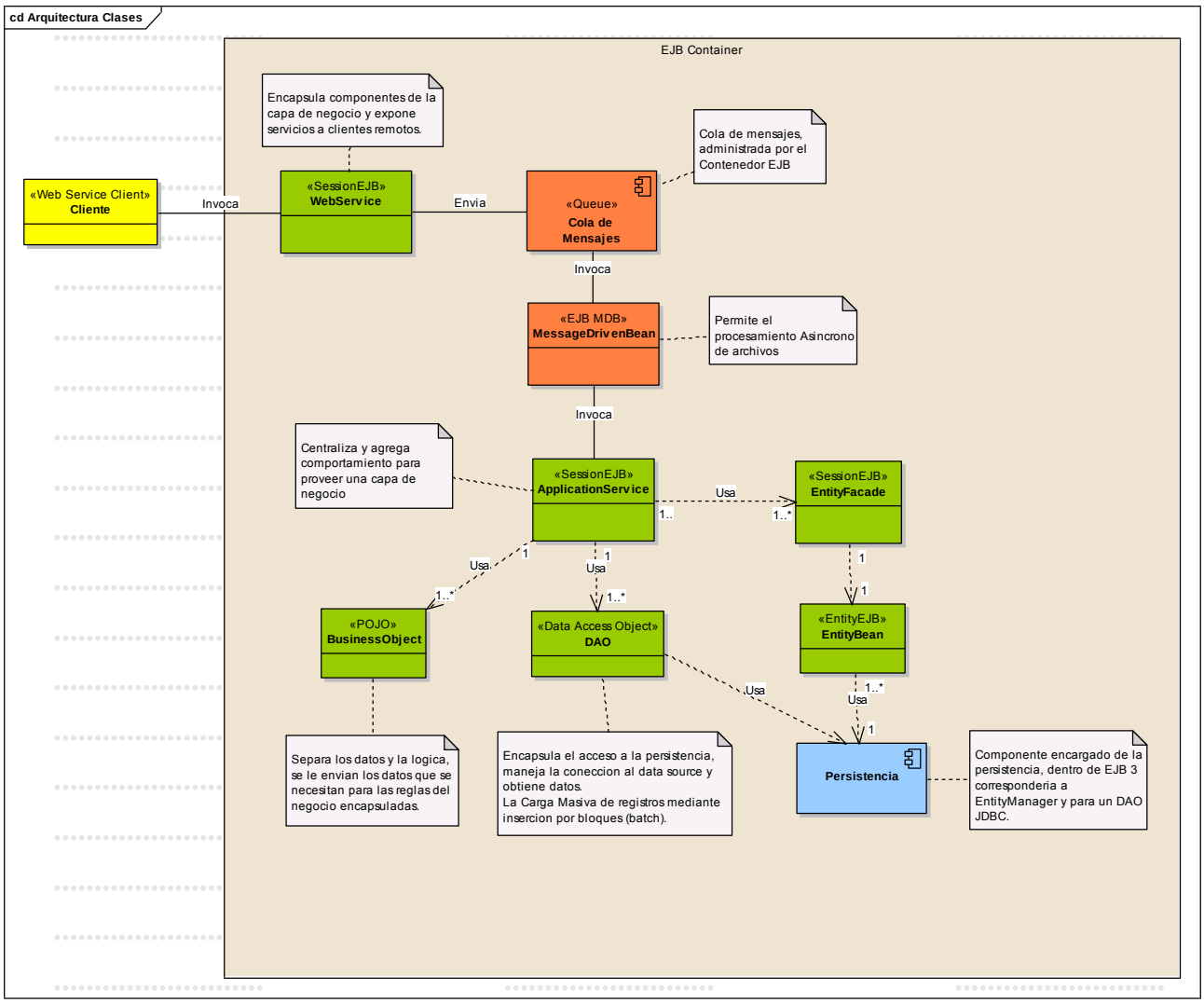


Figura 5.25 - Patrones aplicados en componentes del servidor

5.7.3 Modelo de Datos simulado para Buró de Crédito

La figura 5.26 muestra la estructura de base de datos propuesta para el poblado de los registros enviados, dicha estructura sólo refleja la jerarquía conceptual de las entidades, sin embargo, ya con los registros almacenados en colecciones de objetos, es relativamente sencillo adaptar la información hacia algún otro esquema de tablas según las reglas de negocio propias de Buró de Crédito, las cuales están fuera del alcance de este proyecto.

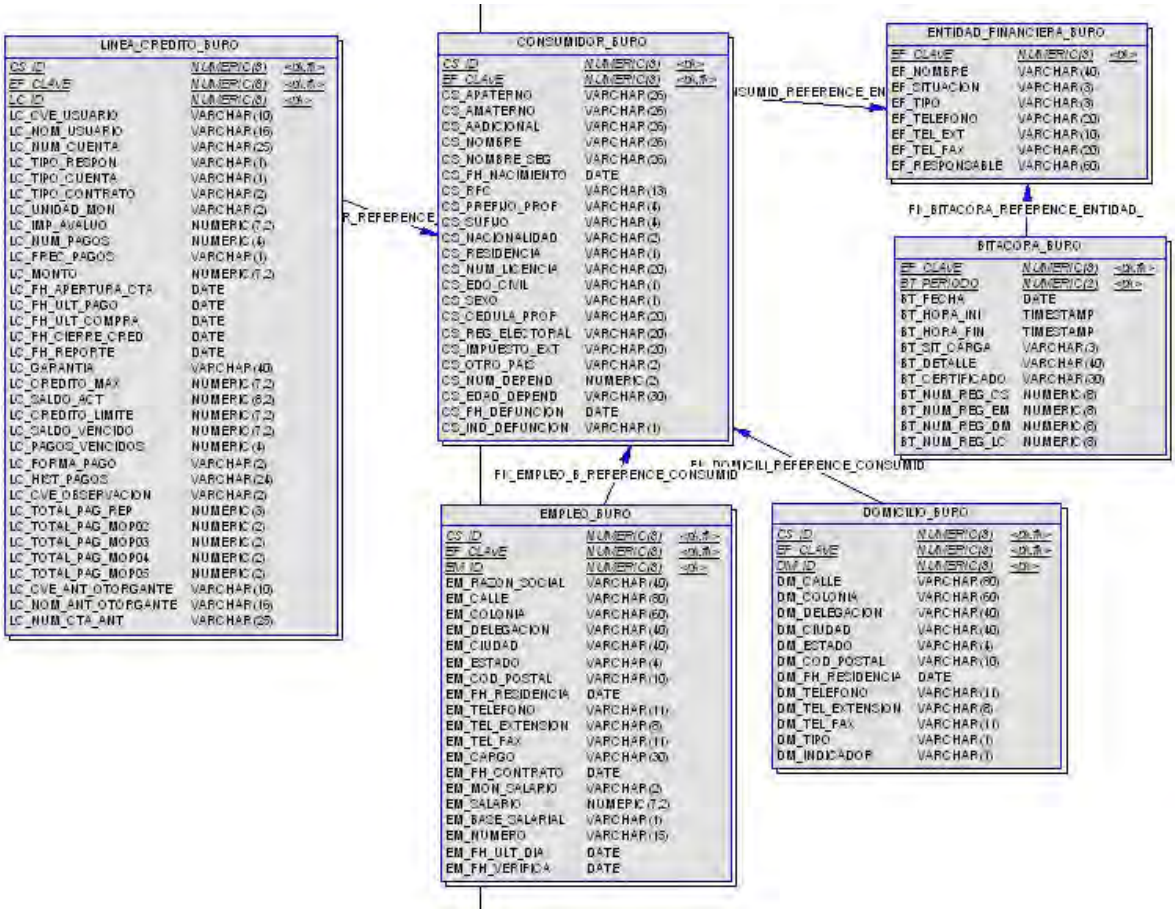


Figura 5.26 - Estructura de tablas de Base de Datos para persistencia de la información

- CONSUMIDOR_BURO: contiene los datos de los consumidores de cada entidad financiera registrada.

- LINEA_CREDITO_BURO: almacena los datos de la línea de crédito de cada consumidor de determinada entidad.

- EMPLEO_BURO: contiene la información de empleo de cada consumidor proporcionada por las entidades.

- DOMICILIO_BURO: almacena los datos sobre el domicilio de cada consumidor.

- ENTIDAD_FINANCIERA_BURO: catalogo de entidades financiera registradas en el sistema de buro para la administración de sus consumidores.

La figura 5.27 muestra el diagrama de secuencia que describe la interacción entre los componentes principales en el proceso de persistencia.

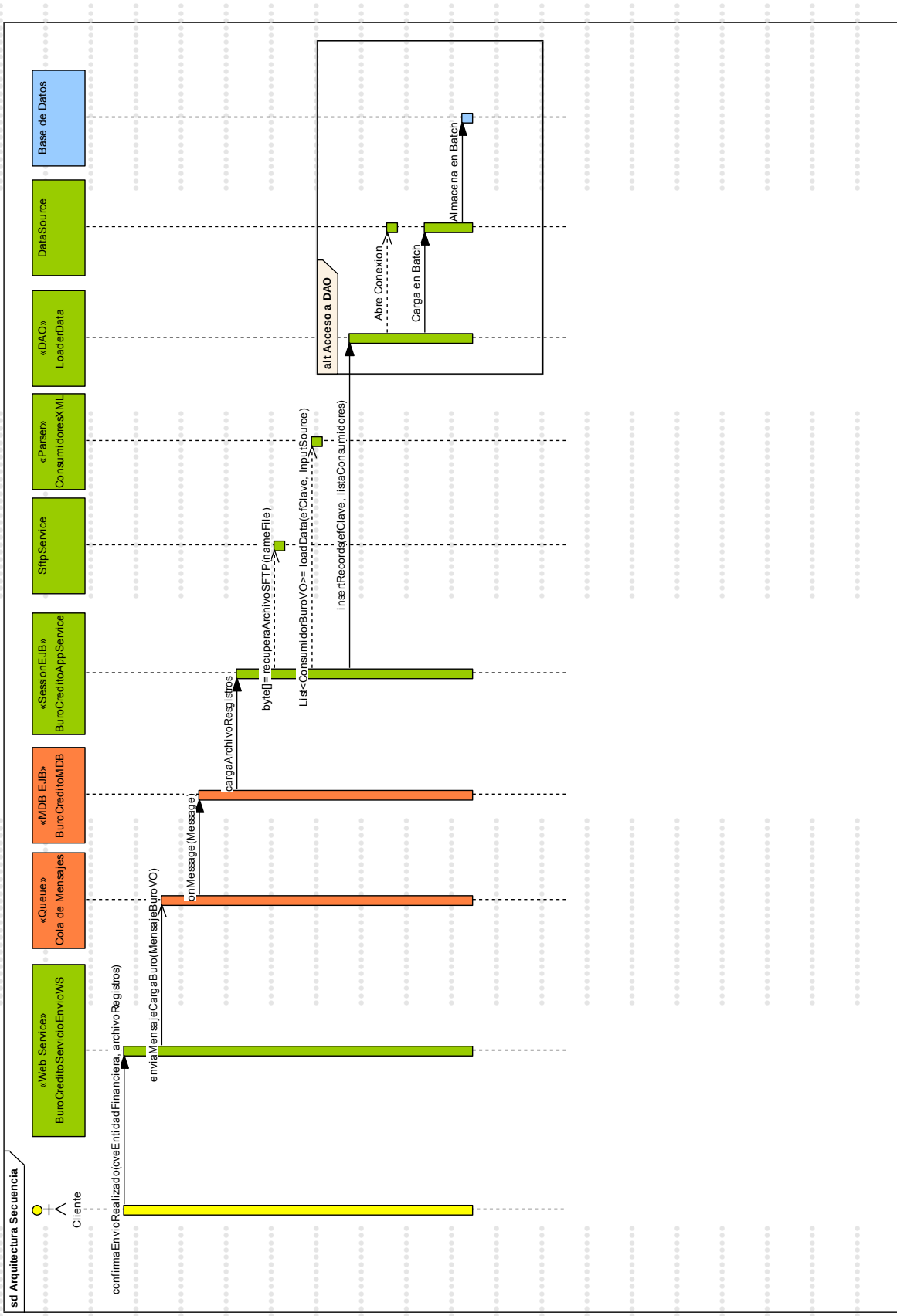


Figura 5.27 - Diagrama de Secuencia de Proceso de persistencia

Finalizado el proceso de persistencia sobre los registros contenidos en un archivo de movimientos enviado por una entidad financiera, se sugiere una notificación a dicha entidad, cerrando así el ciclo de transferencia y notificación entre entidades financieras y Buró de crédito.

5.8. Comparación entre el formato INTF y XML

Aspecto	Formato INTF	XML
Claridad de datos	La información se mezcla con datos del formato, lo que implica que a simple vista sea difícil reconocer los datos y en procesos automatizados aumente su complejidad	Mediante documentos centrados en datos, la revisión del la información es casi transparente a simple vista.
Procesamiento	El procesamiento de documentos se hace mediante la lectura de carácter por carácter, interpretando la posición y longitud de los datos, en caso de algún error durante el proceso toda el almacenamiento puede correr riesgo	Mediante las APIs' de DOM o SAX el procesamiento de documentos XML es muy sencillo brindando capacidades para la obtención de nodos y valores
Tratamiento de errores	Durante el procesamiento de un documento INTF, deben de considerarse el tratamiento de errores en la lectura de cada segmento y campo	DOM y SAX implementan un componente *handler* el cual indica el tipo y la posición donde se localizo el error durante la lectura del XML
Tamaño de documentos	Este formato genera documentos mas pequeños, ya que sólo es la información y los indicadores de posiciones y longitudes	Los documentos XML son más pesados ya que además de los caracteres de las etiquetas y sus nombres, cuentan con espacio para facilitar su lectura por usuarios humanos, siendo estos espacios innecesarios para su procesamiento, sin embargo, actualmente el tamaño de un archivo de texto tiene poca importancia a causa del gran avance de la tecnología. Adaptando la aplicación para el manejo de archivos comprimidos el tamaño de los archivos resultantes aumentan sólo un 20% contra archivos en formato INTF comprimidos.
Tiempo de procesamiento	El procesamiento de un archivo de texto línea por línea es mas rápido que el de un documento XML	DOM es más lento en el procesamiento de los documentos ya que carga el documento completo para crear un árbol de nodos, por otro lado SAX es mucho más veloz, ya que lee línea por línea un archivo, invocando eventos correspondientes a la localización de elementos.
Validaciones	La validación de estos documentos se realiza durante el procesamiento y se desarrolla manualmente. Dando así mayores posibilidades de errores no contemplados	XML brinda tecnologías para su validación antes de comenzar la lectura de un documento, como lo son Schemas XML y DTD
Facilidad de cambios	Los cambios el aplicaciones basadas en este formato son delicados y pueden llevar a errores, además ya que cada entidad cuenta con su propia aplicación el cambio varia para cada una	XML es flexible lo que facilita los cambios en su estructura y procesos de lectura, con una sencilla localización de cada componente

Tabla 5.1 - Comparación entre formatos

Conclusiones

Ahorro de recursos.

Las especificaciones de Buró de Crédito indican que cada Entidad Financiera es responsable del desarrollo de su propio sistema para la generación de los archivos de registros. Al basar el sistema en XML el tiempo de desarrollo se reduce considerablemente, dado que actualmente existe un gran número de herramientas que soportan XML, al igual que la mayoría de las bases de datos.

El proporcionar una aplicación para la validación y envío de los archivos, reduce el tiempo en el cumplimiento del trámite requerido. Además la funcionalidad de extracción en caso que la consulta a la base de datos se adapte al documento de configuración, disminuye la inversión de recursos en la generación de los archivos. Tomando en cuenta que el proceso se realiza por cada entidad del país, el beneficio es aún mayor.

Validación de datos.

La publicación del esquema para los archivos de registros XML, habilita la validación de los mismos antes de realizar el envío evitando posibles rechazos en caso de errores en los registros de consumidores. Garantizando una recepción de archivos de registros validos, lo cual disminuye la posibilidad de errores durante el almacenamiento de los mismos.

Seguridad.

La transferencia hacia un repositorio FTP a través de un canal seguro (SFTP), cubre las necesidades de una transferencia segura, además de un posible manejo de versiones de los archivo enviados.

Una propuesta para mejorar la seguridad de la información, es la utilización de algoritmos de encriptación mediante llaves simétricas y asimetricas, firmas y certificados digitales [26], con lo cual se otorga confidencialidad sobre la información transferida, autenticación de las entidades participantes en el envío y verificación de la integridad de los datos, invocación de los servicios web mediante HTTPS y manejo de permisos para la invocación de los servicios publicados.

Comunicación.

Los servicios web brindan la capacidad de interacción entre aplicaciones en diversas plataformas, lo que permite una libre comunicación entre las aplicaciones de envío (clientes) y la aplicación de procesamiento de archivos (servidor), mediante la implementación de un servicio web para la notificación de envío de archivos, que

[26] El trabajo no cubre el manejo de certificados digitales, para una posible implementación se puede consultar http://java.sun.com/docs/books/tutorial/security/index.html

comienza con la coordinación en el proceso de persistencia. Haciendo posible una automatización de las aplicaciones destinadas al reporte de la información.

Mantenimiento y flexibilidad a cambios.

La adaptación a XML de los archivos de registros y el proceso de persistencia, simplifica de igual manera el desarrollo y mantenimiento del sistema, dando la capacidad de expansión y flexibilidad, minimizando el costo y complejidad de los procesos de la aplicación. Así mismo fomenta la reutilización de los registros para otros fines según convenga a Buró de Crédito o a las mismas Entidades Financieras.

Tamaño

Los archivos en formato INTF son considerablemente más pequeños que un archivo XML con la misma cantidad de registros, ya que los caracteres que conforman las etiquetas, aumentan el peso del documento. Sin embargo, si se adapta la aplicación para soportar archivos comprimidos, los pesos entre un archivo de texto INFT y un archivo XML no difieren demasiado, lo que permite conservar el espacio necesario y aprovechar las características de XML.

	Archivo plano	Tamaño	Archivo comprimido	Tamaño
TXT	10020080501140516.txt	523,710 bytes	10020080501140516.rar	35,816 bytes
XML	10020080608215332.xml	1,083,500 bytes	10020080608215332.rar	43,184 bytes

A causa del tamaño de los archivos de registros se sacrifica un poco en el desempeño de la aplicación en comparación con el procesamiento de archivos de texto plano, pero considerando las capacidades que ofrece XML, las herramientas disponibles para su generación y procesamiento, la disminución en tiempo de desarrollo y mantenimiento, así como la flexibilidad y capacidad de cambio del sistema, dicho sacrificio resulta ser compensado.

Bases academicas

Para la realización del presente trabajo me fueron de gran utilidad los fundamentos y principios tecnológicos contemplados en las materias del plan de estudios de mi carrera, principalmente algunas de estas materias como lo son: bases de datos, programación avanzada, estructuras de datos, redes de cómputo, diseño e implantación de sistemas, sistemas operativos, por mencionar algunas.

Considero que las bases matemáticas fortalecen la capacidad de abstracción para comprender y apreciar los problemas desde diversos puntos de vista, dando como resultado diversas propuestas de solución.

Apéndices

Apéndice A. Lenguaje de configuración para el componente de envío

Se describirá el lenguaje diseñado para la configuración de la aplicación. Dicho lenguaje dará la capacidad de diseñar la estructura de datos que contendrá el documento resultante basado en la consulta indica.

La sintaxis del lenguaje esta definida por el DTD diseñado para tal, que será publicado por el mismo buró de crédito para la validación del documento de configuración, para éste ejemplo se publicó la definición del lenguaje en la ruta de la aplicación web del servidor.

http://localhost:7001/WebBuroConfig/config/wsxml-config.dtd

Elemento wsdbxml-config
Es el elemento raíz del documento, por el momento no contiene atributos.

Elemento connection
Indica la conexión a la base de datos configurada en el documento de configuración local.

Elemento local-config
El documento de configuración local debe de encontrarse en la ruta config\local-config.xml, sin embargo, en caso de localizarse en otro directorio, se indica mediante este elemento, que cuenta con los atributos:

- path: ruta del documento
- type: puede tener los valores url o local para indicar si el documento se encuentra publicado en una dirección de Internet o en un directorio local en caso de que la aplicación se corre localmente

Elementos documents-xml, doc-xml
El elemento documents-xml contiene la declaración de los documentos que se requieren en una invocación del servicio, para fines de este proyecto se configura un sólo documento, cada documento se describe mediante el elemento doc-xml el cual cuenta con los siguientes atributos:

- name: indica el nombre del documento generado
- root: nombre del elemento raíz del documento a generar

Elemento query-fields
Contiene la consulta SQL para recuperar los datos, contenida en el sub-elemento, cuenta con el atributo record-name cuyo valor indica el nombre de cada elemento por registro.

Sub-elementos sql, select, from, where
El elemento sql contiene elementos que indican la consulta a realizar y sus partes:

- select: contiene los campos a seleccionar de la consulta

```
<select>SELECT NOM.APELLIDO_PATERNO,
               NOM.APELLIDO_MATERNO,
               ……
 </select>
```

- from: porción de la consulta que indica las tablas a consultar

```
<from>FROM BC_NOMBRE_PF AS NOM</from>
```

- where: contiene las condiciones para filtrar los registros

```
<where>
        WHERE NOM.ID_EMPRESA = ?
          AND NOM.ANIO_MES = ?
          AND NOM.CVE_TIPO_FORMATO = ?
 </where>
```

Un aspecto importante del elemento select es que puede contener sub-elemento inner_element, que indican la estructura de anidamiento del documento a generar

Elemento inner_element
Esta contenido en el elemento select de la consulta, los campos que contenga serán contenidos por un elemento indicado por el atributo name de elemento.

```
<inner_element name="DOMICILIO_BURO">
     DIR.CALLE_NUMERO   AS DM_CALLE,
     DIR.COLONIA         AS DM_COLONIA,
     DIR.MUNI_DELEG      AS DM_DELEGACION,
     DIR.CIUDAD          AS DM_CIUDAD,
     DIR.CVE_ESTADO      AS DM_ESTADO,
     DIR.CODIGO_POSTAL   AS DM_COD_POSTAL,
     .. .. ..
</inner_element>
```

Elemento query-params
También pertenece a query-fields y contiene los parámetros para la consulta SQL, los cuales se asignan a la consulta en el orden declarados por el elemento qparam con el atributo value que contiene el valor del parámetro. Como se destaca en el ejemplo anterior, los parámetros se indican con el signo de interrogación en la sentencia SQL.

```
 <query-params>
   <qparam value="2"/>
   <qparam value="200502"/>
   <qparam value="PF"/>
 </query-params>
```

La figura A.1 muestra la estructura de árbol del documento de configuración.

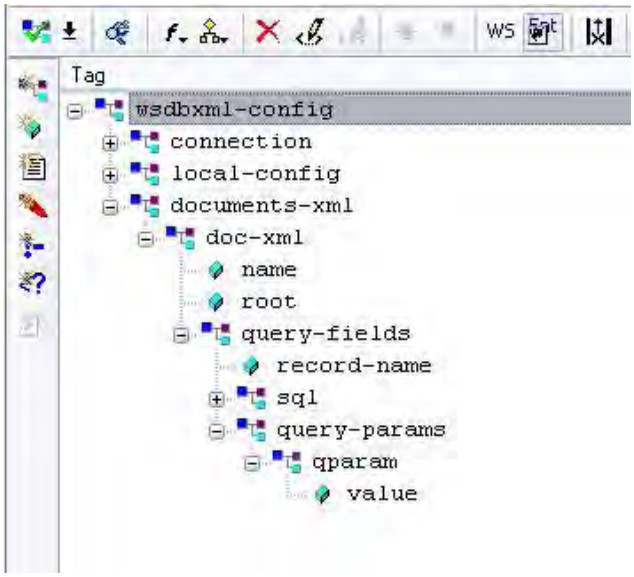


Figura A.1 - Estructura de árbol del documento de configuración

Dentro de las fuentes del proyecto se encuentra un archivo de configuración nombrado "*config\wsxml-config.xml*" donde se estructura la consulta a la base de datos, a través del lenguaje descrito anteriormente.

```
<!ELEMENT wsdbxml-config (connection, documents-xml)>

    <!ELEMENT connection (driver, url-db, usr, psw)>
       <!ELEMENT driver (#PCDATA)>
       <!ELEMENT url-db (#PCDATA)>
       <!ELEMENT usr (#PCDATA)>
       <!ELEMENT psw (#PCDATA)>

    <!ELEMENT documents-xml (doc-xml+)>
       <!ELEMENT doc-xml (query-fields+)>
            <!ATTLIST doc-xml name CDATA #REQUIRED>
            <!ATTLIST doc-xml root CDATA #IMPLIED>
            <!ATTLIST doc-xml version CDATA #IMPLIED>
            <!ATTLIST doc-xml standalone CDATA #IMPLIED>
            <!ATTLIST doc-xml encoding CDATA #IMPLIED>

            <!ELEMENT query-fields (sql, fields-att*, query-params?, query-
fields*)>
               <!ATTLIST query-fields root CDATA #IMPLIED>
               <!ATTLIST query-fields record-name CDATA #IMPLIED>

               <!--ELEMENT sql (select, from, where?)-->
               <!ELEMENT sql ANY>
                  <!--ELEMENT select (#PCDATA)-->
                  <!ELEMENT select ANY>
                     <!ELEMENT inner_element (#PCDATA)>
                        <!ATTLIST inner_element name CDATA #REQUIRED>
                  <!ELEMENT from (#PCDATA)>
                  <!ELEMENT where (#PCDATA)>

               <!ELEMENT fields-att (field)>
                  <!ELEMENT field (#PCDATA)>
                     <!ATTLIST field name CDATA #REQUIRED>
                     <!ATTLIST field att-list CDATA #REQUIRED>
                <!ELEMENT query-params (qparam+)>
                   <!ELEMENT qparam EMPTY>
                      <!ATTLIST qparam field CDATA #IMPLIED>
                      <!ATTLIST qparam value CDATA #IMPLIED>
```

Código A.1 – Definición del tipo de documento (DTD) para el lenguaje de extracción

Apéndice B. EJB

Los EJB's[27] (Enterprise JavaBeans) son componentes del lado de servidor, que son alojados en un contenedor de EJB's dentro de un servidor de aplicaciones. El contenedor se encarga del ciclo de vida de los EJB's, además de la administración de los recursos a los que acceden, y funcionalidades como:

- Seguridad
- Manejo de transacciones
- Concurrencia
- Rendimiento
- Persistencia
- Mensajería
- Manejo de estados
- Timer's (componentes que disparan un evento cada lapso de tiempo)
- Interceptores
- Invocación Remota
- Invocación como Servicios Web

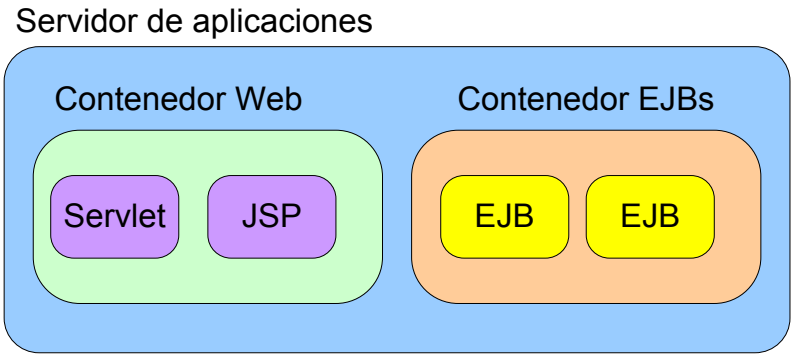


Figura B.1 – Contenedor de EJB's

Una de las nuevas características de Java a partir de la versión 1.5, es el manejo de anotaciones, lo que permite que el desarrollo de EJB's sea sumamente sencillo, ya que las clases destinadas a ser un EJB son simples POJO's (Plain Old Java Object, Objetos Java Planos) con cierto conjunto de anotaciones propias del paquete javax.ejb.

Figura B.2 – Un EJB es un componente administrado por el servidor.

[27] Se utilizó la versión EJB3 para el desarrollo de este trabajo.

Las especificaciones de EJB definen tres tipos:

Session Beans: forma parte de la capa de negocio, contiene los métodos que conforman las reglas del negocio, constan de dos tipos:

- Stateless: no mantienen el estado, sirven como contenedores de métodos, el contenedor se encarga de generar varias instancias contenidas en un pool, por lo que diversos clientes no obtiene la misma referencia al EJB, cuando ya no es utilizada se regresa al pool para su reutilización.

- Statefull: mantiene el estado de la conversación con el cliente, por lo que cada uno cuenta con su propia instancia del EJB. Conserva los valores de las propiedades a través de diversas llamadas de métodos.

La clase correspondiente a un session bean, utiliza la anotación javax.ejb.Stateless o javax.ejb.Statefull según sea el caso, las interfases que los complementan son:

- javax.ejb.Local: permite el acceso local a los métodos del EJB.

- javax.ejb.Remote: permite el acceso remoto a los métodos del EJB.

- javax.jws.WebService: permite el acceso como servicio web a los métodos del EJB.

```
@Stateless
public class NotificadorBean implements NotificadorLocal, NotificadorRemote,
NotificadorWeb {
  public String notificaEnvio(String nombreArchivo){
    . . .
  }
}

@Local
public interface NotificadorLocal {
   String notificaEnvio(String nombreArchivo);
}

@Remote
public interface NotificadorRemote {
   String notificaEnvio(String nombreArchivo);
}

@WebService(serviceName="nombreWS", portName="portWS")
public interface NotificadorWeb {
   String notificaEnvio(String nombreArchivo);
}
```

Código B.1 – Implementación de EJB y sus interfases

Message-driven Beans: al igual que los session bean pertenecen a la capa de negocios, son similares a los stateless, a diferencia que su invocación es a través de mensajes, por lo que corren de manera asíncrona y no tienen interacción directa con el cliente.

Entities Beans: pertenecen a la capa de integración, se ocupan de la persistencia de los datos, modelan el esquema relacional de la base de datos a una jerarquía de objetos

Para una comprensión más completa y profunda sobre el manejo de EJB's se sugiere la referencia [27].

Arquitectura en Capas

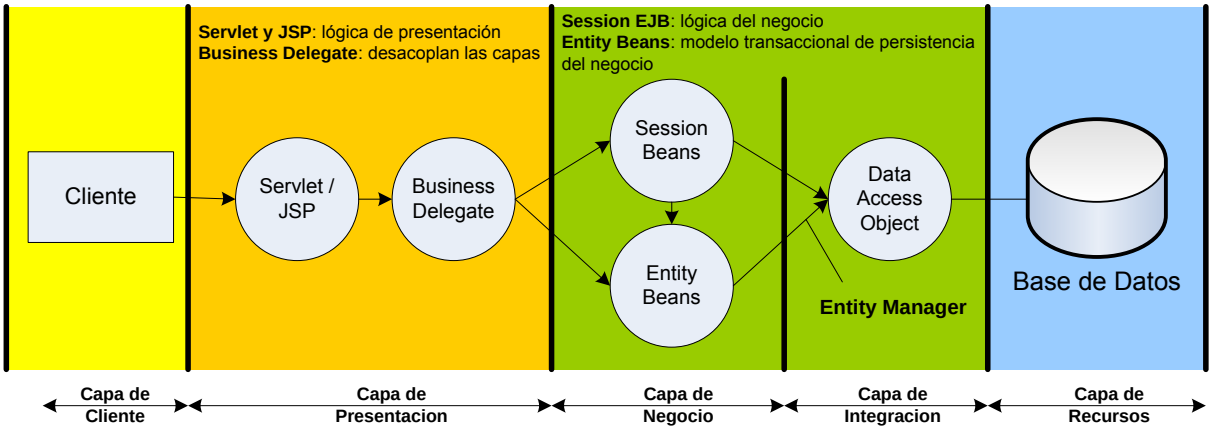


Figura B.3 – Arquitectura en capas

Apéndice C. Formato INTF[28]

Como ya se ha mencionado, buró de crédito proporciona un documento que describe la información requerida, además del formato a seguir. Se proporciona un fragmento de esté documento que describe el formato INFT que es utilizado actualmente por buró de crédito.

Fragmento de requerimientos

El Formato INTF describe la forma de convertir la información crediticia contenida en la Base de Datos de la Entidad Financiera o Empresa Comercial, para ser reportada a Buró de Crédito. *El personal del área de Sistemas de la Entidad será responsable de desarrollar la aplicación automatizada*.

El formato INTF está constituido por segmentos que contienen información homogénea. Por ejemplo el segmento de nombre contiene los datos personales del Cliente: nombre completo, fecha de nacimiento, Registro Federal de Contribuyente, etcétera. El referido a domicilio contiene todos los campos para identificar su dirección: calle y número, delegación o municipio, ciudad, estado y código postal.

La siguiente tabla muestra los segmentos que componen el Formato INTF, así como el orden en que deberán ser estructurados para ser reportados. Es importante observar que únicamente el Encabezado de INTF y el Segmento de Cierre son de longitud fija, el resto son de longitud variable.

El Segmento de Encabezado debe aparecer una sola vez al inicio del archivo y por cada línea de crédito deben considerarse los segmentos de Nombre, Dirección, Empleo y Cuenta.

Etiqueta del Segmento	Nombre del Segmento	Criterios de Validación	Número Máximo de Ocurrencias	Longitud Máxima
INTF	Encabezado	Requerido	1	150 (Fijo)
PN	Segmento de Nombre	Requerido	1	375
PA	Segmento de Dirección	Requerido (opcionales)	1 3	326
PE	Segmento de Empleo	Cuando esté disponible	2	461
TL	Segmento de Cuenta	Requerido	1	436
TR	Segmento de Cierre	Cuando esté disponible	1	253 (Fijo)

Tabla D.1 Segmentos de documento

Las tablas en el Formato INTF describen cada segmento de información, los campos correspondientes y sus características para reportar el historial crediticio del Cliente.

[28] La descripción completa del formato de encuentra en la documentación del proyecto

A continuación se describen los encabezados que aparecen en las tablas analíticas de contenido por segmento:

Concepto	Significado
Posición	Posición de un campo, dentro de un segmento de longitud fija.
Etiqueta del Campo	Es un "string" de dos posiciones que identifica a cada campo en los segmentos de longitud variable.
Longitud del campo	En los campos de longitud variable, después de la etiqueta deben aparecer dos bytes numéricos, indicando el tamaño real de la información a reportar. En los campos de longitud fija es el número exacto de posiciones que debe tener el campo.
Nombre del campo	El nombre de cada campo.
Criterios de Validación	Los campos pueden ser de las siguientes clases: • **Requerido.** Es obligatorio enviar esta información, de lo contrario el registro será rechazado y no podrá ser almacenado en la base de datos de producción. Se solicitará su corrección por parte del Otorgante. • **Opcional para reportar la información si está disponible.** Es recomendable que el Usuario reporte toda la información disponible en su Sistema. Cuando no se cuenta con algún dato es apropiado pasar a la etiqueta siguiente y omitir por completo la etiqueta, longitud y campo. • **Requerido con otros campos**. La información del campo se valida conjuntamente con otro dato contenido en el registro del Cliente, por lo tanto, no se debe enviar solo.
Tipo	Identifica el tipo de carácter que puede aparecer en cada campo. Los valores son los siguientes: • **A: Alfabético**. Acepta sólo letras en los campos. La "Ñ" puede ser representada por el signo de número (#) o la letra "N". Los caracteres alfabéticos deben ser reportados en mayúsculas y sin acentos • **N: Numérico**. Acepta sólo números. • **A/N: Alfanumérico.** La etiqueta acepta letras y números.
Formato	Identifica si el campo es de longitud fija o variable. Los valores en esta columna son: • **F: Longitud Fija**. El campo contendrá un número específico de bytes. • **V: Longitud Variable**. El total de bytes enviados en el campo es variable, sin embargo no excederá el número máximo de bytes establecido en el formato.
Comentarios	Contiene observaciones o instrucciones específicas para reportar el campo. Además de los valores permitidos o requeridos.

Tabla D.2 Campos de Segmentos

Se muestra sólo un fragmento de la tabla para el Segmento de Nombre (PN) que contiene los datos del consumidor:

Etiqueta	Nombre del campo	Tipo	Formato	Longitud	Criterios de Validación	Comentarios
PN	**Apellido Paterno**	A	V	26	***Requerido*** Reportar el apellido completo.	……
00	**Apellido Materno**	A	V	26	*Opcional* Reportar la información si está disponible.	……

01	**Apellido Adicional**	A	V	26	*Opcional* Reportar la información si está disponible.	.....
02	**Primer Nombre**	A	V	26	***Requerido***	
03	**Segundo Nombre**	A	V	26	*Opcional* Reportar la información si está disponible.	
04	**Fecha de Nacimiento**	N	F	8	*Opcional* Reportar la información si está disponible.	

Tabla D.3 Fragmento del Segmento de Nombre (PN)

Cabe destacar que existen tablas de los segmentos para la generación de un archivo con la información de personas físicas y otras tablas para personas morales.

Formato de Entrega

Para dar formato a un segmento de longitud fija simplemente ingrese los datos indicados en cada tabla de descripción, respetando las posiciones que aparecen en ésta. Los segmentos de longitud variable pueden contener ambos tipos de longitudes.

Para crear el formato:

1. Proporcionar al inicio la etiqueta de dos bytes para identificación del campo (referirse a la columna de Etiqueta del campo en cada tabla).
2. Ingresar la longitud (en dos bytes) indicada para la cantidad de posiciones del campo.
3. Agregar el valor del campo.

Por ejemplo: **MARTINEZ** aparecería como **PN08**MARTINEZ, donde:

- **PN** se refiere a la etiqueta del segmento y ocupa dos bytes
- **08** es la longitud (indicado en dos bytes) que ocupará el campo.
- **MARTINEZ** es el valor

Ejemplo de un registro INTF:

```
INTF10ZZ99990001BANCO LATINO 01310819980000000000

PN06MENDEZ0008GONZALEZ0208ANTUANET0513MEGA510503RE3PA26PICO
DE VERAPAZ 435 PISO 50122JARDINES EN LA
MONTAÑA0207TLALPAN0306MEXICO0402DF0505142100TL02TL0110ZZ999900
010212BANCO
LATINO0406123456050101060101070202CL0802MX1001211101M1204106213081
1111996140806081998210516732220419792403180250112602029903FIN
```

Referencia bibliográfica

[1] Michael Morrison. XML al descubierto. Prentice Hall. Madrid, 2000, 1ª edición.

[2] Heather Williamson. XML: Manual de Referencia. McGraw-Hill, Osborne Media. Madrid, 2001, 1ª edición.

[3] SGML. http://www.serv-inf.deusto.es/abaitua/konzeptu/sgml

[4] Elliotte Rusty Harold. Processing XML with Java: A guide to SAX, DOM, JDOM, JAXP and TrAX. Addison- Wesley, USA, 2003, 1a edición.

[5] Brian Benz, John R. Durant. XML Programming Bible. Wiley Publishing, Inc. USA, 2003, 1a edición.

[6] Thomas A. Powell. HTML 4: Manual de Referencia. McGraw-Hill, Osborne Media, Madrid 2001, 1ª edición.

[7] H. M. Deitel, P. J. Deitel. Java: How to program. Deitle & Associates, Inc. USA, 1999, 3a edición.

[8] Herbert Schildt. Java 2: Manual de Referencia. McGraw-Hill, Osborne Media, Madrid 2001, 4ª edición.

[9] Phil Hanna. JSP: Manual de Referencia. McGraw-Hill, Osborne Media, Madrid 2002, 1ª edición.

[10] Budi Kurniawan.Java for the Web with Servlets, JSP, and EJB. New Riders, USA 2002, 1a edición.

[11] Craig Larman. UML y Patrones, Una introducción al analisís y diseño orientado a objetos y al proceso unificado. Pearson, Prentice Hall, España, 2003, 3ª edición.

[12] Abraham Silberschatz, Henry F. Korth, S. Sudarshan. Fundamentos de Bases de Datos. McGraw-Hill, Osborne Media, Madrid 1998, 3ª edición.

[13] XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm

[14] DTD to Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm

[15] XML. http://www.dei.uc.edu.py/tai2000/xml/biblio.htm

[16] Jim Melton, Andrew Eisenberg. SQL y Java. Alfaomega, México 2002, 1a edición.

[17] XML and Databases. http://www.timebase.com.au/index.cfm?method=MALTXML.article2#classifications

[18] XML Database Products. http://www.rpbourret.com/xml/XMLDatabaseProds.htm

[19] Thomas Erl. Service-Oriented Arquitecture. Prentice Hall, USA.

[20] XML Schemas. http://www.w3schools.com/schema/schema_intro.asp

[21] Estrategias para Transferencia de Datos. http://www.rpbourret.com/xml/DataTransfer.htm

[22] Mapeo de DTD a Bases de Datos.http://www.rpbourret.com/xml/DTDToDatabase.htm

[23] Richard Monson – Haefel. J2EE Web Services. Pearson Education

[24] JAXR. http://www.javapassion.com/webservices/JAXR_speakernoted.pdf#search='JAXR'

[25] Java EE 5. http://java.sun.com/javaee/5/docs/tutorial/doc/

[26] Ed Romn, Scout W. Ambler, Tyler Jewell. Mastering Enterprise JavaBeans. Wiley Computer Publishing, EUA 2002, 2a edición.

[27] Debu Panda, Reza Rahman, Derek Lane. EJB3 In Action. Manning, EUA Julio 2007, 1ª edición.