UNIVERSIDAD NACIONAL AUTONÓMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
A C A T L A N

"DESARROLLO DE PROTOTIPO DE UN WEB SERVICE PARA ESTABLECER INTEROPERABILIDAD ENTRE APLICACIONES HETEROGÉNEAS EN UNA INSTITUCIÓN BANCARIA".

SEMINARIO TALLER EXTRACURRICULAR

QUE PARA OBTENER EL TÍTULO DE LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

P R E S E N T A :

JORGE ALVAREZ LÓPEZ

ASESOR:
LIC. CARLOS ALBERTO RANGEL ROJAS

NAUCALPAN, EDO DE MÉXICO
SEPTIEMBRE 2005

Quiero agradecer muy especialmente a mis padres,
quienes me han apoyado,
no solo en mi vida estudiantil,
sino a lo largo de toda mi existencia.

A mi esposa
por todo su apoyo y comprensión,
fundamentales para mi desarrollo profesional.

A los profesores que me dedicaron tiempo,
a mi asesor por sus acertadas observaciones.

Finalmente, por mi fuente de inspiración, **Mis Hijos.**

GRACIAS!!

Contenido

Objetivos *4*
Introducción *5*

Introducción a Web Services *7*
1.1 Aplicaciones WEB 8
1.2 Descripción de los Web Services. 10
1.3 Arquitectura de Web Services. 13
1.4 Descubrimiento UDDI. 16
1.5 WSDL. 17
1.6 Codificación: XML. 18
1.7 Transporte 20

Planteamiento del problema *24*
2.1 Problema 25
2.2 Consecuencias de manejo de información inadecuado. 26
2.3 Requerimientos 27
2.4 Préstamos 29
2.5 Créditos 33
2.6 Propuesta de Solución 35

Diseño *37*
3.1 Procesos actuales. 38
3.2 Especificaciones para llevar a cabo el seguimiento del flujo de un préstamo 40
3.3 Diseño arquitectónico del componente. 43
3.4 Diseño modular. 46
3.5 Alcances del nuevo componente. 51

3.6 Descripción de datos utilizados. 53
3.7 Aplicación cliente. 54

Desarrollo del prototipo *56*
4.1 Prototipo 57
4.2 Metodología. 57
4.3 Desarrollo del componente. 59
4.4 Desarrollo de las clases 61
4.5 Funciones y métodos 63
4.6 Intercambio de datos 66
4.7 Mapeo de tipos complejos 67
4.8 Interfase cliente 69

Implementación *73*
5.1 Equipo 74
5.2 Base de datos. 75
5.3 Puesta en marcha 76
5.4 Pruebas del prototipo 77
5.5 Documentos 79

Conclusiones *80*
Glosario *80*
Bibliografía *90*
Referencias bibliográficas *90*
Referencias Web *91*

Objetivos

Objetivo General

Comunicar aplicaciones heterogéneas por medio de un componente Web Service

Por medio de un cornponente Web Service, establecer una comunicación directa entre aplicaciones desarrolladas en diferentes lenguajes y en diferentes plataformas utilizando protocolos estándares. Esto conlleva a que los usuarios del componente sean las aplicaciones y no los usuarios, y de esta manera evitar procesos duplicados y reducir el uso directo de la intervención humana.

Objetivos específicos

- Mostrar la importancia de utilizar la tecnología denominada Web Services.
- Unificar procesos en el control de la Administración dentro de una institución bancaria.
- Comunicar aplicaciones heterogéneas.

Introducción

Cuando alguien usa Internet, las dos tareas más comunes son navegar por la Web y enviar o recibir correo electrónico.

Sin embargo, como el uso de Internet va creciendo cada vez más, en consecuencia, nuevas tecnologías y aplicaciones son liberadas, las cuales, tienen el potencial de cambiar la manera de su uso. El problema con Internet, hasta ahora, es que para utilizarlo, es necesaria la presencia de un ser humano. Por el otro lado, existen los *Web Services*, los cuales están construidos para ser leídos e interpretados por programas computacionales, no por humanos.

De modo, que el conjunto de *Web Services* en Internet es una World Wide Web paralela, de carácter no humano, sino cibernético. Es como pensar que los ordenadores ya hablan solos a través de Internet.

Actualmente los *Web Services* están siendo utilizados como ayuda a grandes y pequeñas entidades para obtener mas de sus recursos tecnológicos, permitiendo la integración de diversas aplicaciones de software, desde programas de escritorio, hasta amplios sistemas empresariales, no sólo útiles para operaciones diarias, sino especialmente útiles en integración de sistemas post integración o adquisición.

Web Services, como indica su propio nombre, son servicios ofrecidos vía Web. Representan una solución con mucho futuro para la integración de aplicaciones en Internet, es por ello, que son el principal enfoque en este trabajo.

El presente trabajo está estructurado de la siguiente manera:

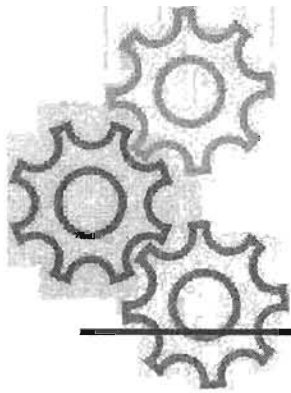
Capítulo 1 Introducción a *Web Services*. En este capítulo se presenta un panorama general de lo que son los *Web Services*, como interactuar con ellos y cuales son sus principales características, así como sus componentes.

Capítulo 2 Planteamiento del problema. En este capítulo, se plantea y se analiza la problemática de contar con sistemas hechos en diferentes lenguajes y diferentes plataformas dentro de una institución bancaria, se definen los requerimientos para la realización de un componente que de solución a la problemática planteada y se definen los conceptos bancarios utilizados en este trabajo.

Capítulo 3 Diseño. En este capítulo se lleva el diseño de la arquitectura y los procesos, con ayuda de diagramas de caso de uso para tener una vista general, y con diagramas de secuencia para conocer aspectos más a detalle. De igual manera se diseña una aplicación cliente que utilice los servicios ofrecidos por el componente *Web Service*.

Capítulo 4 Desarrollo del prototipo. En este capítulo se hace el desarrollo del componente como un prototipo con sus principales funciones y métodos, mostrando líneas de código en las partes clave del desarrollo.

Capítulo 5 Implementación. En este capítulo se hace la implementación y evaluación del prototipo desarrollado, mencionando el comportamiento general que se obtuvo durante el proceso de evaluación.



1 Introducción a Web Services

Objetivos

- Analizar de manera general cómo trabajan los *Web Services*, para qué sirven, y su utilización práctica.
- Describir los procesos necesarios para la implementación de un *Web Service*.

Este capítulo nos introduce a la nueva tecnología basada en los *Web Services* para tener un panorama más amplio de los conceptos que se manejarán durante el trabajo.

1.1 Aplicaciones WEB

En un principio, el uso de Internet perseguía objetivos publicitarios o el suministro de funcionalidades adicionales que complementaran la actividad principal de una empresa, sin embargo, hoy en día la base de muchos negocios reside en la disponibilidad de los servicios ofrecidos a través de Internet.

La Web en sus orígenes fue pensada como un medio para desplegar información de manera estática en los servidores, la cual es accedida a través de una consulta hecha por un navegador valiéndose del protocolo HTTP (Hyper Text Transfer Protocol; es el protocolo de transferencia de texto más utilizado en internet). Actualmente se maneja el mismo concepto en la comunicación de peticiones cliente-servidor (navegador - webserver) sólo que no necesariamente el resultado de la comunicación debe provenir de la carga de una página estática, esta puede ser el resultado de la ejecución en el servidor remoto de alguna aplicación. Esto último no es en si una aplicación Web, pero se acerca al concepto.

En esencia, una aplicación Web usa un sitio Web como entrada (front-end) a una aplicación típica. Si no existe la lógica del negocio en el servidor, el sistema no puede ser llamado aplicación Web. Bajo este concepto las aplicaciones Web no sólo se encargan de desplegar información, sino que también, deben contener una lógica asociada que permita apoyar algún proceso propio del negocio para el cual fue diseñada.

Tecnologías para el desarrollo de aplicaciones web.

Para el desarrollo de aplicaciones Web se han generado múltiples tecnologías entre ellas se encuentran: CGI, Fast CGI, Páginas Dinámicas, Servlets, *Web Services*.

Common Gateway Interfase (CGI) fue la primera técnica utilizada para que el contenido de las páginas Web se generara de manera

dinámica, es común encontrar en los diferentes servidores Web el módulo que soporta la ejecución de CGIs. De manera resumida se puede decir que el CGI es un mecanismo de comunicación entre el servidor Web y una aplicación en otra computadora, esta aplicación puede estar desarrollada en casi cualquier lenguaje, sólo debe cumplir la condición de ser soportado por el servidor http, es común encontrar que la mayoría de las aplicaciones CGIs se encuentren desarrolladas con el lenguaje PERL.

Este mecanismo tiene limitaciones que evita su uso a gran escala, la más conocida es en cuanto a rendimiento, ya que por cada petición que se realice en el servidor se crea un nuevo proceso, lo cual tiene un costo muy alto en lo que a recursos del sistema se refiere.

Fast - CGI. Esta es una solución similar al CGI mencionado anteriormente, propone la creación de un solo proceso persistente por cada programa FastCGI, en lugar de ejecutar un proceso por cada solicitud del cliente. Es una solución viable pero también tiene inconvenientes de proliferación de procesos en el caso de peticiones concurrentes.

Páginas dinámicas. Con la aparición de esta tecnología se entra a una nueva forma de trabajo, la cual esta orientada al trabajo del diseñador Web, quien no necesariamente conoce de lenguajes de programación. Este nuevo enfoque consiste en insertar pequeños fragmentos de lógica de programación en la estructura HTML de la página, al contrario de lo que se hacia en los CGIs, que era en el lenguaje de programación utilizar sentencias de impresión para generar salidas HTML. En este sentido se conocen diferentes alternativas, entre ellas podemos mencionar PHP, ASP, JSP, entre otros.

Servlets. El servlet podemos considerarlo como una evolución de los CGIs desarrollada por SUN Microsystems como parte de la tecnología JAVA. De forma general consiste en la ejecución de aplicaciones Java en el motor de servlets (Servlet engine), el cual hace parte del servidor Web. Algo que lo hace ventajoso con respecto a los CGIs, es que por cada petición de usuario no se crea un proceso, sino un hilo, el cual es mucho más económico para el sistema. Esta tecnología es parte de la arquitectura propuesta por SUN en su plataforma J2EE (Java 2 Enterprise Edition).

Web Services. La arquitectura de *Web Services* plantea algo más que una técnica para el desarrollo de aplicaciones Web, representa un modelo de computación distribuida para Internet basado en XML

(eXtensible Markup Language). Bajo este concepto ya no sólo se trata la comunicación usuario - aplicación, sino que de manera adicional se maneja la interacción aplicación - aplicación.

1.2 Descripción de los Web Services

En un *Web Service*, una aplicación de negocio envía una petición vía HTTP a un servicio situado en una URL. El servicio recibe la petición, la procesa y devuelve una respuesta también sobre HTTP.

La idea es sencilla pero requiere:

- Un protocolo de intercambio de mensajes petición/respuesta sobre HTTP.
- Una forma de que clientes y proveedores puedan interactuar a través de los mensajes, es decir, un lenguaje de especificación de interfases.

Se ha optado por utilizar SOAP (Simple Objet Access Protocol) como protocolo de intercambio de mensajes. Es un protocolo sencillo basado en XML y estandarizado por el W3C (The World Wide Web Consortium).

El lenguaje de especificación de interfases utilizado en *Web Services* es WSDL (Web Services Description Language). WSDL permite especificar en XML las operaciones y tipos de datos de un *Web Service*. Así, aunque el cliente y el servidor estén escritos en lenguajes distintos (por tanto, con sintaxis y tipos de datos diferentes) pueden interactuar al utilizar un lenguaje neutral para comunicarse.

Una petición de un servicio Web constaría de los siguientes pasos:

- En el cliente se elabora una petición de una operación con unos parámetros.
- La petición se transforma a formato XML utilizando WSDL.
- La petición transformada se envía vía HTTP utilizando SOAP.
- El servidor de Web Services recibe la petición.
- El servidor determina qué operación debe realizarse y transforma los parámetros de formato XML a su representación

correspondiente en el lenguaje utilizado para implementar el servidor.

- El servidor invoca la operación con los parámetros enviados, elabora una respuesta y se la envía al cliente de la misma manera.

También existe una especie de listado telefónico en el que se publicitan los *Web Services*. El UDDI (Universal, Description, Discovery and Integration) es una iniciativa de varias empresas (IBM, Microsoft, entre otros) que ofrece un servicio gratuito para registrar y buscar *Web Services*. Cada *Web Service* se registra dando, entre otras cosas, su nombre, su punto de acceso y una descripción del servicio.

Su implementación

Para implementar *Web Services* existen varias APIs (comerciales y gratuitas) para los lenguajes más usuales. Las APIs no son estándares pero esto no afecta a la interoperabilidad ya que la interoperabilidad es posible porque los protocolos (SOAP, WSDL, UDDI) están estandarizados.

Podemos distinguir dos tipos de APIs:

- APIs de programación basadas en mensajes: tanto el receptor como el emisor disponen de un proveedor de mensajería. Permiten mensajes síncronos y asíncronos, enviar a más de un receptor y calidad de servicio. Son APIs muy potentes pero complejas.
- APIs de programación basadas en RPCs[1]: en el caso de un lenguaje orientado a objetos, este tipo de APIs permiten definir interfaces (en WSDL) cuyos métodos se pueden invocar remotamente (similar a CORBA). No es tan potente como las APIs basadas en mensajes pero es mucho más sencilla de usar.

Sus APIs

- Dentro de la tecnología Microsoft, los *Web Services* forman parte de .NET, y en cuanto a Java, existen muchas APIs de distintos fabricantes (Iona, Inprise, Oracle, IBM, Sun, entre otros) y también gratuitas (Apache).

[1] (Remote Procedure Call) Llamada a Procedimiento Remoto

- Sun está estandarizando el API Java para *Web Services*. Consta de varias APIs que forman parte de J2EE:
- *JAXM*: Api de Java para Mensajes XML.
- *JAX-RPC*: Api de Java para XML basado en RPC.
- *JAXR*: Api de Java API para registros XML.

Requisitos de un *Web Service*

Para que un componente, solución o aplicación pueda ser considerado como un *Web Service*, debe cumplir los siguientes requisitos:

- *Interoperabilidad*: Debe dar servicio remoto y permitir su utilización por clientes de otras plataformas.
- *Amigabilidad con Internet:* Debe poder funcionar para soportar clientes que accedan a los servicios remotos desde Internet.
- *Interfaces con la misma definición de tipos de dato:* No debe haber ambigüedad acerca del tipo de dato enviado y recibido desde un servicio remoto, los tipos de datos definidos en el servicio remoto deben corresponder con los de la mayoría de los lenguajes de programación procedimentales.
- *Posibilidad de aprovechar los estándares de Internet existentes:* La implementación del servicio remoto debe aprovechar estándares de Internet existentes tanto como sea posible, y evitar reinventar soluciones a problemas que ya se han resuelto. Una solución construida sobre un estándar de Internet ampliamente adoptado puede aprovechar conjuntos de herramientas y productos existentes creados para dicha tecnología.
- *Soporte para cualquier lenguaje:* La solución no debería ligarse a un lenguaje de programación particular, Java RMI, por ejemplo, esta ligada completamente a lenguaje Java. Sería muy difícil invocar funcionalidad de un objeto Java remoto desde Visual Basic o PERL. Un cliente debería ser capaz de implementar un nuevo *Web Service* existente independientemente del lenguaje de programación en el que se halla escrito el cliente.

- *Soporte para cualquier infraestructura de componente distribuida:* No debe estar fuertemente ligada a una infraestructura de componentes en particular. De hecho, no se debería requerir el comprar, instalar o mantener una infraestructura de objetos distribuidos, solo construir un nuevo servicio remoto al utilizar un servicio existente. Los protocolos subyacentes deben proporcionar un nivel base de comunicación entre infraestructura de objeto distribuidos existentes tales como DCOM y CORBA.

Los *Web Services* se definen como la nueva generación de componentes de software, independientes de plataforma, que permite a las aplicaciones compartir datos vía Internet, fácilmente publicadas, localizadas e invocadas mediante protocolos Web estándar, como XML, SOAP, UDDI o WSDL.

1.3 Arquitectura de Web Services

Un *Web Service* se registra en un repositorio de servicios, el cliente busca en el repositorio el servicio que necesita y luego lo invoca. De manera más detallada, la arquitectura de los *Web Services* es una meta-arquitectura que permite que ciertos servicios de red sean dinámicamente descritos, publicados, descubiertos e invocados en un ambiente de cómputo distribuido.

Los Web Services son aplicaciones auto-contenidas y modulares que pueden ser:

- Descritas mediante un lenguaje de descripción de servicio, como el lenguaje WSDL (Web Service Description Language) .
- Publicadas al someter las descripciones y políticas de uso en algún registro bien conocido, utilizando el método de registro UDDI (Universal Description, Discovery and Integration).
- Encontradas al enviar peticiones al registro y recibir detalles de ligamiento (binding) del servicio que se ajusta a los parámetros de la búsqueda.

- Asociadas al utilizar la información contenida en la descripción del servicio para crear una instancia de servicio disponible o proxy.
- Invocadas sobre la red al utilizar la información contenida en los detalles de ligamento de la descripción del servicio.
- Compuestas con otros servicios para integrar servicios y aplicaciones nuevas.

Como descripción más real del funcionamiento de los *Web Services*, tenemos que los *Web Services* son construidos y luego descritos por medio de WSDL y registrados bajo el estándar UDDI, el cliente busca en el registro UDDI (como si fuese un motor de búsqueda al estilo de Google) y obtiene el descriptor WSDL del servicio que necesita, lo invoca haciendo uso de SOAP el cual es utilizado para comunicar la petición entre los diferentes componentes del servidor que aloja el *Web Service*, para entregar una respuesta utilizando nuevamente SOAP. En la figura 1.3.1 "Arquitectura de Web Services" se muestra gráficamente la interrelación de los componentes que conforman el *Web Service.*

La arquitectura de los *Web Services* no hace referencia a un lenguaje de programación, se da libertad a los desarrolladores para utilizar la plataforma y el lenguaje que deseen para implementarlos. Actualmente se pueden construir *Web Services* en múltiples lenguajes de programación, dentro de las posibilidades que existen para el desarrollo utilizando plataformas de Sofware Libre o de código abierto son:

- Axis: implementación de SOAP en el lenguaje Java.
- SOAP::Lite: Implementación de SOAP para el lenguaje PERL
- SOAP-Python: Implementación de SOAP para el lenguaje Python
- NuSOAP: Implementación de SOAP para el lenguaje PHP
- SOAP4R: Implementación de SOAP para el lenguaje Ruby

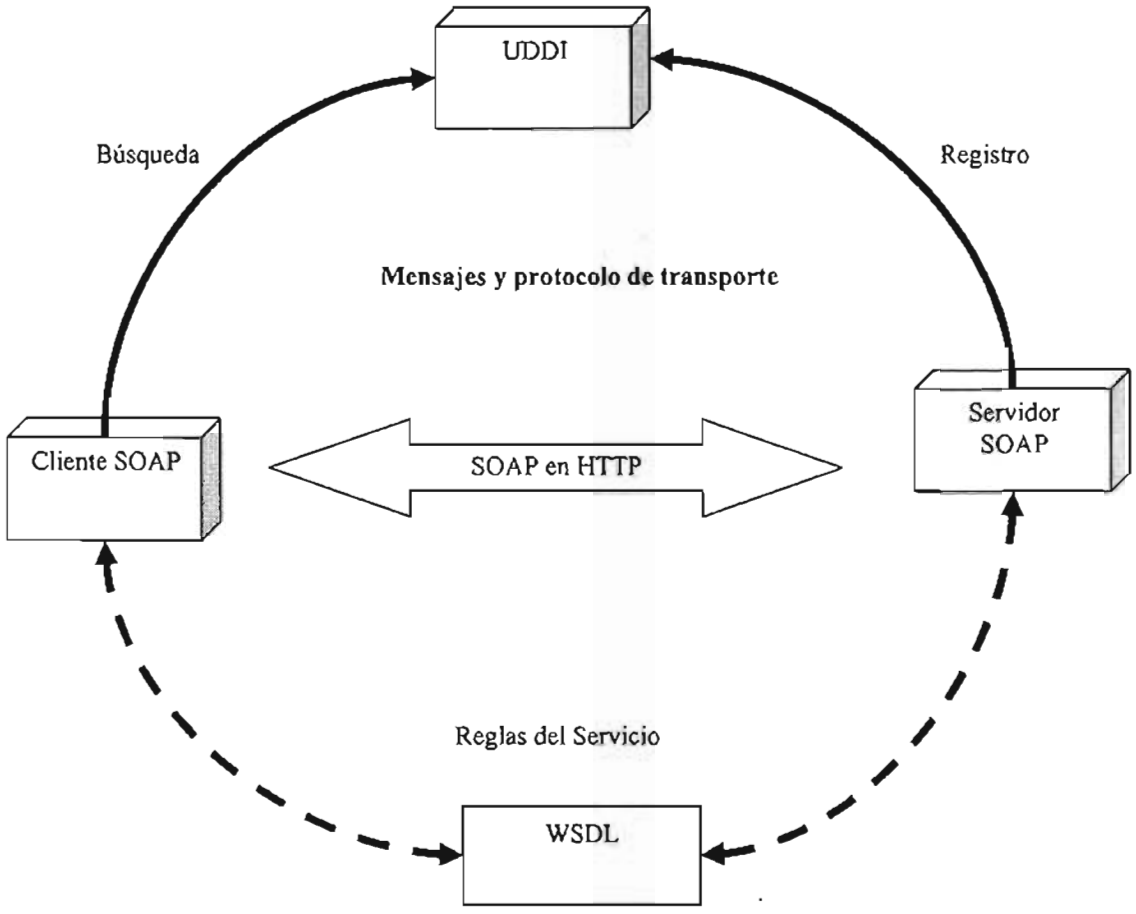


Figura 1.3.1 "Arquitectura de Web Services"

Protocolos y estándares utilizados en los Web Services

Se han definido una serie de estándares para el desarrollo de los Web Services, los principales son:

- UDDI (Universal, Description, Discovery, and Integration): (Descubrimiento, Descripción e Integración Universal)
- WSDL (Web Service Description Language): (Lenguaje de Descripción de Servicios Web)
- XML eXtensible Markup Language. XML: (Lenguaje de Marcado eXtensible)

- SOAP (Simple Object Access Protocol): (Protocolo Simple de Acceso a Objetos)

1.4 Descubrimiento UDDI

La aplicación cliente que necesita acceder a la funcionalidad que expone un *Web Service*, requiere una forma de resolver la ubicación del servicio remoto. Esto se logra mediante un proceso llamado normalmente, *descubrimiento* (discovery).

UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen. UDDI contiene información sobre las interfases técnicas de los servicios de una empresa. A través de un conjunto de llamadas a API XML basadas en SOAP, se puede interactuar con UDDI tanto en tiempo de diseño como de ejecución para descubrir datos técnicos de los servicios que permitan invocarlos y utilizarlos. De este modo, UDDI sirve como infraestructura para una colección de software basado en *Web Services*.

Varias empresas, incluidas Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP y unas trescientas más[2], unieron sus esfuerzos para desarrollar una especificación basada en estándares abiertos y tecnologías no propietarias, que permitiera resolver los retos anteriores. El resultado, cuya versión beta se lanzó en diciembre de 2000 y estaba en producción en mayo de 2001, fue un registro empresarial global alojado por varios nodos de operadores en el que los usuarios podían realizar búsquedas y publicaciones sin coste alguno.

A partir de la creación de esta infraestructura para *Web Services*, los datos sobre estos servicios se pueden encontrar de forma sistemática y confiable en una capacidad universal totalmente independiente de proveedores. Se pueden llevar a cabo búsquedas categóricas precisas utilizando sistemas de identificación y taxonómicos extensibles. La integración de UDDI en tiempo de ejecución se puede incorporar a las aplicaciones. Como resultado, se fomenta el desarrollo de un entorno de software de *Web Services*.

[2]Para obtener un listado actualizado, consulte http:///www.UDDI.org/Community

1.5 WSDL

Una vez que se tiene un *Web Service* funcionando, el cliente necesita suficiente información para interactuar adecuadamente con el mismo. La descripción de un *Web Service* implica datos con una estructura predefinida sobre la interfaz que intenta utilizar la aplicación cliente, así como documentación escrita del *Web Service* incluyendo un ejemplo de uso.

Para describir cómo interacciona un cliente con el *Web Service* es necesario definir un esquema para los mensajes que se intercambiarán entre el cliente y el servidor. El esquema debe contener una definición para los mensajes de petición y repuesta de los métodos que se pueden ejecutar. Uno de los principales objetivos es que los desarrolladores no tengan que investigar en las definiciones del esquema. Es por eso que requiere describir el servicio de forma que una herramienta pueda descifrarlo y crear un Proxy por el cliente.

Además de la información que proporciona el esquema, el cliente debe conocer los patrones de intercambio de mensajes para invocar de manera correcta los métodos que expone el *Web Service,* evitando así, posibles ambigüedades.

Una descripción formal de los patrones de mensaje resulta aún más importante en el caso de *Web Services* más complejos. Algunos servicios podrían aceptar una petición pero no enviar la respuesta correspondiente devuelta al cliente.

El lenguaje de descripción de Web Services (WSDL, Web Service Description Language) es un dialecto basado en XML sobre el esquema que describe un *Web Service*. Un documento WSDL proporciona la información necesaria al cliente para interaccionar con el *Web Service*. WSDL es extensible y se puede utilizar para describir, prácticamente, cualquier servicio de red.

Dado que los protocolos de comunicaciones y los formatos de mensajes están estandarizados en la comunidad del Web, cada día aumenta la posibilidad e importancia de describir las comunicaciones de forma estructurada. WSDL afronta esta necesidad definiendo una gramática XML, que describe los servicios de red como colecciones de puntos finales de comunicación capaces de intercambiar mensajes. Las definiciones de

servicio de WSDL proporcionan documentación para sistemas distribuidos, y sirven como fórmula para automatizar los detalles que toman parte en la comunicación entre aplicaciones.

1.6 Codificación: XML

XML es un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.

XML fue creado al amparo del Word Wide Web Consortium (W3C) organismo que vela por el desarrollo de WWW.

Su desarrollo se comenzó en 1996 y la primera versión salió a la luz el 10 de febrero de 1998. La primera definición que apareció fue: "Sistema para definir, validar y compartir formatos de documentos en la Web." [XML, URL]

Objetivos De XML

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fácilmente creables.

Principales características de XML

A continuación se presentan las principales características que conforman a XML.

- Es una arquitectura más abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en internet/intranet por medio de un validador de documentos (parser).
- Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML (los RDF Resource Description FrameWork), en comparación a los atributos de la etiqueta del HTML.
- Integración de los datos de las fuentes más dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje nos permitirá agrupar una variedad amplia de aplicaciones, desde páginas Web hasta bases de datos.
- Gestión y manipulación de los datos desde el propio cliente Web.
- Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido Web en XML consigue que la estructura de la información resulte más accesible.
- Permite búsquedas personalizables y subjetivas para robots y agentes inteligentes. También conlleva a que los clientes Web puedan ser más autónomos para desarrollar tareas que actualmente se ejecutan en el servidor.
- Comportamiento más estable y actualizable de las aplicaciones Web, incluyendo enlaces bidireccionales y almacenados de forma externa (El epígrafe "404 file not found" puede desaparecer).
- Permite denominación independiente de la ubicación, enlaces bidireccionales, enlaces que pueden especificarse y gestionarse desde fuera del documento, hiperenlaces múltiples, enlaces

agrupados, atributos para los enlaces, etc. Creado a través del Lenguaje de enlaces extensible (XLL).

- Facilita la exportación a otros formatos de publicación (papel, Web, cd-rom, etc.). El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

XML y Los Web Services

XML se utiliza para los Web Services por lo siguiente:

- *Es un estándar abierto;* es reconocido mundialmente, ya que muchas compañías tecnológicas integran en su software compatibilidad con dicho lenguaje. Esto quiere decir que la gran mayoría de software de escritorio, de sistema operativo, aplicaciones móviles, etc., permiten la compatibilidad con XML esto lo hace muy potente a la hora de permitir la comunicación entre distintas plataformas de software y hardware (este es el sentido final de los *Web Services*).

- *Simplicidad de sintaxis;* esto quiere decir que es muy fácil de escribir código en XML, y la representación de los datos es entendible por casi cualquier ser humano. Esto lo hace muy flexible a la hora de reprensar datos de cualquier especie, basta con contar con cualquier editor de texto y aprender instrucciones básicas para estar en condiciones de escribir código XML, el cual será soportado o entendido por cualquier aplicación que pueda leer documentos XML. El hecho de que XML sea tan fácil de codificar y de entender lo hace el lenguaje ideal para utilizarlo en los *Web Services.*

- *Independencia del protocolo de transporte;* XML es un lenguaje de marcado de texto, no necesita de ningún protocolo de trasporte especial, sólo necesita de un protocolo que pueda transferir texto o documentos simples. Existen muchos protocolos con estas características como lo son HTTP y SMTP por nombrar algunos.

1.7 Transporte

Una vez que se ha dado formato al mensaje y se han serializado los datos en el cuerpo del mensaje, se deben transferir entre el cliente y el servidor utilizando algún protocolo de transporte.

Los protocolos:

Existen dos maneras de invocar los *Web Services*: XML-RPC y SOAP. Cuando se comienza a programar un *Web Service*, es necesario decidir qué protocolo usar, ya que estos protocolos son incompatibles. De modo que si se propaga un *Web Service* con XML-RPC, no podrá ser invocado desde un lenguaje de programación que trabaje con SOAP, como por ejemplo .Net de Microsoft.

Tanto SOAP (Protocolo de acceso a objetos simple, Simple Object Access Protocol) como XML-RPC son lenguajes de mensajería basada en XML, estandarizados por el consorcio W3C, que especifican todas las reglas necesarias para ubicar *Web Services* XML, integrarlos en aplicaciones y establecer la comunicación entre ellos.

La diferencia entre SOAP y XML-RPC es su complejidad. XML-RPC está diseñado para ser sencillo. SOAP por el contrario está creado con idea de dar un soporte completo y minucioso de todo tipo de *Web Services*. Aprender XML-RPC es más sencillo, por lo que un programador novato en este campo, puede conseguir resultados satisfactorios con poco esfuerzo. Con SOAP no pasa esto, pero a cambio, se dispone de más potencia. Por ejemplo, con XML-RPC no se puede elegir el conjunto de caracteres para el envío de datos, en cambio con SOAP, se puede elegir entre varios formatos como US-ASCII, UTF-8 y UTF-16.

SOAP (Simple Object Access Protocol)

Es un protocolo para intercambio de información en un descentralizado y distribuido ambiente de desarrollo. Está basado en XML, consta de tres partes: una envoltura que describe que contiene el mensaje y como procesarlo, una serie de reglas codificadas para expresar instancias de tipos de datos de aplicaciones definidas, y una convención para representar llamadas y respuestas remotas. SOAP tiene la potencialidad de poder ser usado en combinación con una amplia variedad de protocolos, siendo la base de transmisión, el protocolo HTTP.

Una de las razones principales es que SOAP ha recibido un increíble apoyo por parte de la industria. SOAP es el primer protocolo de su tipo que

ha sido aceptado prácticamente por todas las grandes compañías de software del mundo. Algunas de las grandes Compañías que soportan SOAP son Microsoft, IBM, SUN Microsystems, SAP y Ariba.

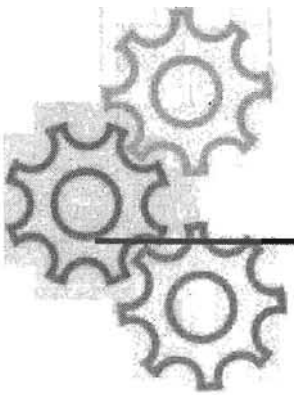
Algunas de las Ventajas de SOAP son:

- No está asociado con ningún lenguaje: los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista pero los desarrolladores responsables de mantener antiguas aflicciones heredadas podrían no poder hacer esta elección sobre el lenguaje de programación que utilizan. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación.
- No se encuentra asociado a ningún protocolo de transporte: La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- No está atado a ninguna infraestructura de objeto distribuido.
- Aprovecha los estándares existentes en la industria: Los principales contribuyentes a la especificación SOAP, evitaron intencionadamente reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes en lugar de utilizar su propio sistema de tipo que ya están definidas en la especificación esquema de XML. Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.
- Permite la interoperabilidad entre múltiples entornos: SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dichos estándares pueden comunicarse mediante mensajes SOAP con aplicaciones que se ejecuten en otras plataformas. Por ejemplo, una aplicación de escritorio que se ejecute en una PC puede

comunicarse con una aplicación del back-end ejecutándose en un mainframe capaz de enviar y recibir XML sobre HTTP.

- La anatomía de un mensaje de SOAP es simple, SOAP proporciona un mecanismo estándar de empaquetar un mensaje. Un mensaje SOAP se compone de un sobre que contiene el cuerpo del mensaje y cualquier información de cabecera que se utiliza para describir el mensaje.

Es importante observar que los Web Services pueden ser la tecnología más prometedora dentro de los sistemas distribuidos por las características estudiadas en este capítulo.



2 Planteamiento del problema

Objetivos

- Analizar el problema existente.
- Definir requerimientos.
- Introducir al lector a los términos bancarios utilizados en este trabajo.

En este capítulo, se plantea la problemática existente en una institución bancaria con sucursales ubicadas en Europa y en América, algunas de ellas en México, debido a que sus aplicaciones no fueron diseñadas para convivir directamente con otras aplicaciones de la misma institución desarrolladas en otras plataformas y en otros lenguajes.

2.1 Problema

En la actualidad, debido al avance tecnológico dentro del área de sistemas, el tiempo de vida del software es corto. Aún cuando el software haya sido desarrollado a la medida del problema, siempre surgen nuevas necesidades que hacen que estos vayan perdiendo su funcionalidad, situación que ocurrió en cierta institución bancaria, en los departamentos de préstamos y créditos.

El software con el que cuentan estos departamentos, tenía como objetivo controlar cada una de sus áreas, solamente dentro de su localidad, pero con el paso del tiempo, ha surgido la necesidad de comunicar estas aplicaciones para compartir información que arroje reportes diarios, semanales y mensuales. Estos reportes son a nivel Institución, además del departamental, entonces se requiere recibir datos no sólo de las sucursales de México, sino de las que se encuentran en toda Europa y América. En un principio se optó por establecer, como pronta solución, un intercambio de información por medio de archivos de texto, ya que las aplicaciones están desarrolladas en diferentes lenguajes de programación y bajo diferentes plataformas. Esto es, que los usuarios de préstamos recopilaran los datos solicitados por el departamento de crédito, y enviarlos por e-mail por medio de archivo de texto. Los usuarios del departamento de crédito envían también sus datos de la misma forma.

Es importante señalar que por este medio, la información no es segura, ya que puede ser alterada por cualquier usuario que así lo desee. Por tal motivo, después de procesados los reportes diarios, se emiten los reportes comparativos para revisar que la información enviada y recibida sea concordante. Esta situación ya no es conveniente, pues se utilizan más recursos de los necesarios para hacer el ajuste en caso de requerirse, y averiguar por qué la información no cuadra si fuera el caso.

Otro punto desfavorable es que la información, generalmente queda fuertemente ligada al programa con que fue generada, y es así como se pierde mucho tiempo en pasar la información de Sybase a Excel, de Excel a HTML, y de Oracle a SQL server. Es importante tener presente que esa información está cómo quiso verla la persona que la generó.

Los bancos enfrentan numerosos desafíos, incluyendo la necesidad de aumentar la eficiencia operativa, mejorar y expandir las ofertas de producto, mejorar el nivel de servicio y las relaciones con el cliente, bajar costos, aumentar la productividad, seguridad en procesos y asegurar el cumplimiento de reglamentaciones.

Los equipos de desarrollo de la institución se han enfrentado a severos problemas: como flexibilidad en modificaciones, incrementos, y conectar aplicaciones heterogéneas que cumplan con los requerimientos.

2.2 Consecuencias de manejo de información inadecuado

- No hay integridad en la información, algunas veces la información está alterada o dispar.
- Pérdida de tiempo en revisar la información procesada, por no tener un mecanismo seguro de procesar la información.
- No se tiene la información en el tiempo que se requiere.
- Exceso de tiempos muertos en los diferentes departamentos.
- Duplicidad en procesos, muchos procesos que ya existen se vuelven a programar.
- Envíos lentos de información, debido a que la comunicación es por medio de archivos planos de texto.
- No interoperan las aplicaciones, es decir, no se comunican entre ellas directamente, necesitan del trabajo humano para hacer sus tareas.
- No está bien coordinado el personal para la entrega y recepción de archivos.

2.3 Requerimientos

Especificación de requerimientos del usuario

Funcionales.

- Contar con una interfase que pueda emitir reportes de ambos departamentos (Créditos y Préstamos)
- Generación de solicitud para renovar préstamos y créditos, así como de nuevos.
- Generar reportes de los diferentes tipos de operaciones que se procesan en cada departamento, con diferentes criterios y con la opción de seleccionar los campos que se necesita consultar.
- Control de los pagos por vencer, con envío automático por E-mail a los clientes que tengan pagos próximos a vencer.
- Tener la opción de accesar al sistema desde cualquier máquina que cuente con un browser y conexión a nternet.
- Respuesta rápida en los procesos.
- Seguir utilizando los sistemas existentes en ambos departamentos (Créditos y préstamos).

No Funcionales.

- Tener disponibilidad de interacción total entre aplicaciones en todo momento. Es decir, que las aplicaciones puedan hacer peticiones a otras en cualquier momento y bajo cualquier circunstancia con una pronta respuesta.
- Contar con una interfaz robusta que garantize la comunicación directa entre componentes de software. Eliminar terceras intervenciones que faciliten interpretaciones erróneas de la información que se procesa.
- Contar con información autocontenida de cada componente listo para su operación. De tal manera que cada componente describa con detalle los metodos, funciones y propiedades con las que cuenta.
- Seguridad entre aplicaciones y usuarios. Controlando accessos y malos manejos de la información.
- Veracidad y confiabilidad de información por medio de comunicación directa entre componentes de software.
- Minimizar tiempos de procesos.
- Eliminar duplicidad de procesos.
- Seguridad.

Dentro de la institución se analizaron los procesos de dos áreas que trabajan de manera separada, para integrar los servicios ofrecidos por sus aplicaciones en el *Web Service*.

A continuación se analiza la manera de manejar los créditos y préstamos para comenzar a resolver la problemática iniciando con estas dos áreas, y lograr poco a poco que todas las aplicaciones de la institución tengan una interoperabilidad directa.

2.4 Préstamos

Un préstamo no es más que un sistema de financiación en el cual desde un primer momento queda fijada la cuantía y plazo de devolución del capital prestado. Por otra parte, es una financiación que desde un primer momento está vinculada a su finalidad. Así cuando se solicita un préstamo se hace para crear un negocio, ampliar una empresa, ir de vacaciones, etc., en definitiva, para acceder a un bien o servicio concreto.

Un préstamo es un sistema de financiación que permite a una persona física o jurídica adquirir un determinado bien o servicio, financiando una parte de su coste a medio y largo plazo.

Un préstamo cuenta con las siguientes características:

Plazo

Viene prefijado desde el momento de su contratación siendo el marco temporal más habitual aquel que comienza en los dos años como mínimo y los treinta años como máximo. Esta gran amplitud de los plazos va vinculada a las garantías del mismo así un préstamo a dos años suele tener una garantía personal mientras que un préstamo a treinta años suele ser de garantía hipotecaria.

Garantías

La contratación de un préstamo está sujeta a la aportación de una serie de garantías, la cuantificación de dichas garantías suele superar el 200 % del principal solicitado. Estas garantías pueden ser realse, personales ó pignoraticias[3].

Reales; estas garantías son las exigidas en operaciones de muy larga duración, habitualmente más de diez años, siendo la garantía un inmueble que se encuentra directamente vinculado al préstamo y que responde de su pago. Así si el préstamo no es atendido lo primero que hará la entidad financiera para cobrar la deuda será embargar dicho inmueble.

[3]Pignoraticio: es, según el Diccionario usual de la Real Academia (1899) "perteneciente o relativo a pignorar" y pignorar se presenta como sinónimo de "empeñar".

Personales; se habla de garantías personales cuando no existe ningún inmueble hipotecado a favor de la entidad y por tanto esta concede el préstamo basándose en la solvencia real y moral del prestatario[4]. Así la entidad a través de la capacidad de generación de ingresos del prestatario, tiene clara su capacidad para hacer frente a las cuotas y por su historial, tiene claro que hará todo lo posible para hacer frente a dichas cuotas. Esto no quiere decir que como consecuencia del impago de un préstamo de garantía personal la entidad no pueda ir contra los bienes del prestatario, de hecho sea cual sea el tipo de garantía de la operación, el prestatario responderá de ella con todo su patrimonio presente y futuro.

Pignoraticias; se trata de utilizar como garantía del préstamo un capital depositado en la misma entidad financiera que concede el préstamo.

Costes y Liquidaciones

Los costes de un préstamo están agrupados en función del momento de la vida del préstamo en que se encuentre. Así se tiene:

Comisión apertura y gastos de formalización, son todas las comisiones y gastos añadidos, fedatario público, tasaciones, etc. a los que el prestatario ha de hacer frente al contratar la operación.

Intereses, durante la vida normal del préstamo tan sólo se deberá de hacer frente a los intereses devengados en cada período de liquidación sin tener que hacer frente a otro tipo de gastos o comisiones.

Gastos de modificaciones, si el prestatario realiza amortizaciones[5] anticipadas, cancela anticipadamente el préstamo o modifica sus características, deberá hacer frente a comisiones y gastos, aunque muchas de estas comisiones son negociables y en muchas ocasiones no se liquidan.

En cuanto a las liquidaciones, todos los préstamos liquidan con una frecuencia predeterminada en el momento de la contratación, mensual, trimestral, cuatrimestral o anual, una cuota compuesta habitualmente de intereses y amortización. Así en cada cuota se liquidarán los intereses

[4]Prestatario: Persona titular de un préstamo, que asume todas las obligaciones y adquiere todos los derechos del contrato que firma con la entidad financiera prestamista.

[5]Amortización: Pago total o parcial que se realiza para la devolución de un préstamo. Además, se entiende por amortización la cantidad del capital que se va devolviendo. Por tanto, a medida que va pasando el tiempo, de cada cuota se irá amortizando más capital y se irán reduciendo los intereses.

generados por el capital pendiente hasta dicha fecha, mas una parte de dicho capital.

Modalidades

Los préstamos se pueden dividir en dos grandes modalidades o tipos basados en las garantías más habituales.

Préstamos personales, son operaciones de préstamo que habitualmente tienen siete años como plazo máximo y cuyas garantías son personales.

Hipotecarios, son operaciones a largo plazo, entre los diez y los treinta años con principales elevados y que están garantizados a través de una hipoteca sobre un determinado inmueble.

Tipos de interés y divisa

En cuanto al tipo de interés los préstamos, se pueden contratar tanto a interés fijo como variable, si bien la cuantía del tipo de interés aplicado está vinculada a la garantía de la operación. Así, entre mayores son las garantías más bajo será el tipo de interés aplicado, de hecho, los préstamos hipotecarios suelen tener tipos de interés sustancialmente inferiores a los préstamos personales.

Por otra parte, en muchas ocasiones, es determinante el plazo de la operación para que el tipo de interés sea fijo o variable. Así, cualquier préstamo se podrá contratar a interés variable, mientras que si el plazo del préstamo es superior a los 15 años es casi seguro que no se podrá contratar a interés fijo. Evidentemente cuando se contrata una operación a interés fijo, tanto la institución bancaria, como el prestatario, corre el riesgo de equivocarse y por tanto cuanto mayor es dicho plazo mayor es el riesgo y menor la posibilidad de cubrir dicho riesgo por lo que las entidades prefieren no asumirlo.

En cuanto a la divisa de un préstamo esta puede ser cualquiera con cotización oficial, siendo el tipo de interés de referencia para el préstamo el tipo de interés de la divisa.

Notas a tener en cuenta

El préstamo es una operación que por su propia definición es inflexible, sus características se mantienen constantes desde el momento de su contratación. Este hecho unido a que sus plazos suelen ser medios o largos, hace que se deban negociar con especial cuidado ya que si se realiza una mala negociación, se sufrirá durante un largo período de tiempo.

Es el producto ideal para financiar la adquisición de bienes duraderos ya que se puede acomodar la financiación a la duración del bien permitiendo asumir su adquisición con facilidad.

En ocasiones puede ser útil utilizar un préstamo para conseguir recursos financieros destinados a la financiación de desfases de tesorería. Así en el caso de una empresa, puede utilizar un préstamo para no tener que acudir al descuento comercial, o anticipo sobre recibo bancario, productos de financiación de circulante más caros que un préstamo.

Normalmente las entidades financieras no prestarán el 100 % del precio del bien que se va a adquirir, la filosofía es que un préstamo es una financiación del capital que falta para adquirir el bien y no la financiación del bien en su totalidad. En este último caso la entidad entiende que más que un prestarnos dinero, está alquilando el bien.

En el caso de un préstamo hipotecario, cuando después de haber impagado se produce la ejecución de la garantía y por tanto la subasta del inmueble hipotecado, puede ser que el precio de adjudicación sea inferior al importe de la deuda reclamada por la entidad financiera, y por tanto, no sólo se pierda dicho inmueble, sino que la entidad continúe actuando contra el resto del patrimonio del titular.

Titularidad

Dado que este tipo de operaciones se solicita tanto por personas físicas como jurídicas es muy habitual encontrarse con un préstamo cuya titularidad la ostenta una empresa y a la vez está avalado por una persona física. También es habitual que el préstamo tenga una titularidad pluripersonal. Así una agrupación de empresas o personas pueden ser titulares, solidaria o mancomunadamente, de una operación de préstamo. Este hecho es muy habitual en los préstamos hipotecarios para la adquisición de vivienda donde lo más normal es que los adquirientes sean pareja y ambos firmen la titularidad del inmueble y de la hipoteca.

2.5 Créditos

La Póliza de Crédito es un instrumento de financiación muy interesante para la empresa, pues su flexibilidad es total, con lo que la empresa paga por la financiación que realmente precisa.

Concepto

Una póliza de crédito conceptualmente no es otra cosa que una cuenta corriente que permite disponer de una cantidad de dinero a discreción, utilizando para ello los instrumentos de movilización de fondos habituales en una cuenta corriente.

Por otra parte, se debe tener en cuenta que en una póliza de crédito, se puede realizar tanto disposiciones como imposiciones de tal modo que a priori no tendría porqué tener vencimiento, sin embargo lo habitual es que una póliza de crédito si tenga un vencimiento. En este sentido, existen las pólizas de campaña, con vencimiento a seis meses hasta las pólizas de crédito con renovación tácita y vencimiento a los tres o cinco años de su formalización.

Instrumentación

Mediante un número de cuenta del que se pueda disponer con cualquier instrumento de movilización de fondos, cheque, pagaré, tarjeta, transferencia, permitiendo además la domiciliación de efectos comerciales o recibos periódicos.

Operatoria y Costes de utilización

Una vez formalizada la operación, es firmada ante fedatario público la póliza de garantías y abierta la cuenta de crédito, su utilización es idéntica a la de una cuenta corriente, con la salvedad de que el importe disponible con el que cuenta es el límite pactado en la póliza de garantías. Una vez abierta la cuenta de crédito, simplemente se realizan disposiciones e imposiciones como si de una cuenta tradicional se tratara.

Al igual que una cuenta corriente, periódicamente se realizan liquidaciones de intereses y gastos.

Como ya se ha mencionado, el período de tiempo durante el que se puede utilizar la póliza suele ser limitado, de unos meses a varios años, esto hace que a su vencimiento, si se tiene saldo dispuesto, se deba cubrir inmediatamente dicho saldo, bien mediante la renovación de la póliza, utilizando fondos propios o refinanciando el saldo dispuesto mediante un préstamo. Este último caso es muy habitual cuando se produce una mala gestión de la póliza utilizándose el saldo disponible para la financiación de activos fijos, hecho que habitualmente hace que desde dicho momento hasta su vencimiento, se encuentre dispuesta y sea muy difícil cancelar. Puesto que durante toda la vida de la póliza no se tiene obligación de cubrir el saldo dispuesto, las entidades financieras son vigilantes con respecto a su utilización, de tal modo que si detectan que esta se está utilizando para otros fines que cubrir los desfases de tesorería suelen imponer a su vencimiento la cancelación total o parcial del crédito.

Tipo de interés y garantías

A diferencia del descuento comercial o el anticipo sobre el recibo bancario, el análisis realizado por la entidad cuando se le plantea una póliza de crédito, es igual o mayor magnitud que cuando se le plantea un préstamo. Esto se debe a la falta de vencimientos periódicos ya que muy bien podríamos disponer de todo el crédito el primer día y no atender ninguna de las liquidaciones de intereses hasta su vencimiento ni saldarla en dicha fecha con lo que automáticamente la entidad debería iniciar el proceso de ejecución de las garantías.

Ventajas

Se trata del instrumento de financiación de circulante más flexible del mercado. Puesto que la póliza de crédito admite tanto disposiciones como imposiciones realmente pagamos la financiación que necesitamos en cada momento.

Dado que operativamente funciona igual que una cuenta corriente, la realización de disposiciones es tan simple como la realización de un reintegro o la firma de un cheque o pagaré sin tener que realizar gestión añadida alguna.

Bien gestionada es el producto de financiación de circulante más barato del mercado.

Si se plantea adecuadamente la negociación y el objetivo de su tenencia, es la cobertura de los desfases de tesorería, se puede, negociando una póliza plurianual, solucionar necesidades de financiación de circulante en una única gestión para varios ejercicios.

Inconvenientes

La discrecionalidad de las disposiciones y la no obligación de realizar imposición alguna durante su período de vigencia, hace que las entidades exijan un nivel de garantías muy superior al necesario para negociar una póliza de descuento de efectos del mismo importe.

Es un instrumento que se ha de utilizar adecuadamente, esto es, no por disponer de dicho saldo lo podemos aplicar a lo que sea. Su razón y máxima utilidad se encuentra cuando se utiliza para cubrir desfases de tesorería, en caso contrario es más que probable que se encuentre con un problema cuando llegue su vencimiento.

2.6 Propuesta de Solución

Como solución al problema de esta institución, se propone una nueva tecnología llamada *Web Services*, que ha emergido como un mecanismo de integración de sistemas, rompiendo la barrera del lenguaje de programación y plataforma. Brindando además flexibilidad en modificaciones futuras.

Parte de la solución es desarrollar un componente que enlace las aplicaciones existentes dentro de la institución, sin importar en que lenguaje de programación estén desarrolladas, y accesando a una nueva base de datos para almacenar los registros que se vayan generando en las bases de datos de cada departamento. Otro aspecto a considerar es hacer las respectivas modificaciones a las aplicaciones actuales, para que puedan interactuar directamente con el componente.

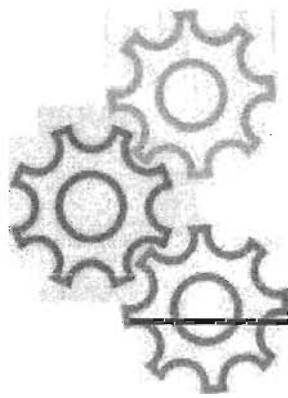
Los *Web Services* permiten que las aplicaciones compartan información y que además invoquen funciones de otras aplicaciones independientemente de cómo se hayan creado las aplicaciones, cuál sea el sistema operativo o la plataforma en que se ejecutan y cuáles los dispositivos utilizados para obtener acceso a ellas. Aunque los *Web*

Services son independientes entre sí, pueden vincularse y formar un grupo de colaboración para realizar una tarea determinada.

Los protocolos que soportan los servicios Web se comunican normalmente por el puerto 80, y basándose en HTTP, métodos GET y PUT. Esto hace que se pueda acceder a ellos al igual que lo se hace en una página Web. La diferencia entre una página Web y un *Web Service*, es que la página la visita cualquier individuo interesado, mientras que el servicio sólo lo visitan programas que lo requieren.

Los usuarios que utilizarán este nuevo componente serán principalmente los desarrolladores de aplicaciones, analistas y administradores de bases de datos. Ya que son las aplicaciones desarrolladas por estas personas las que tendrán contacto directo con el componente.

Ya conociendo la problemática existente en la institución, se puede comenzar con el modelo del prototipo sugerido.



3 Diseño

Objetivos

- Diseñar el componente en base a los requerimientos.
- Diseñar una aplicación cliente que utilice el los servicios ofrecidos por el componente Web Service

En este capítulo se hace el modelado de las diferentes interfaces que compondrán el componente *Web Service*, el objetivo es proporcionar a los desarrolladores un panorama más completo de cómo se requiere que funcione la aplicación y como establecer la conexión entre las aplicaciones de préstamos y créditos, así como de una aplicación cliente que utilice este servicio y que lo pruebe. Se hace el modelo del lado del cliente y del lado del servidor.

3.1 Procesos actuales

Cabe mencionar que actualmente se utilizan dos sistemas independientes para el control de préstamos, créditos, y manejo de reportes, situación que se pretende cambiar con este proyecto. Tomando en cuenta las políticas y procedimientos de las dos áreas (créditos y préstamos), se fueron depurando los procesos duplicados o que simplemente ya no aplican bajo las circunstancias actuales.

Cabe resaltar que la información de las bases de datos actual no es consistente, ya que el sistema de créditos guardaba en un principio las fechas como tipo char, y después se modificó para que las guardara como tipo date. Parte del proyecto consistió en depurar las actuales bases de datos, con el fin de tener todos los datos consistentes.

En la figura 3.1.1 "Diagrama de procesos actuales" se muestra el modelo de ambos sistemas (créditos y préstamos), en el que se puede observar como se manejan los dos sistemas completamente separados uno del otro, pero que conllevan procesos que podría ser el mismo para ambos. El usuario de cada departamento es el que da inicio al proceso, los cuales muestran condiciones semejantes pero separadas físicamente.

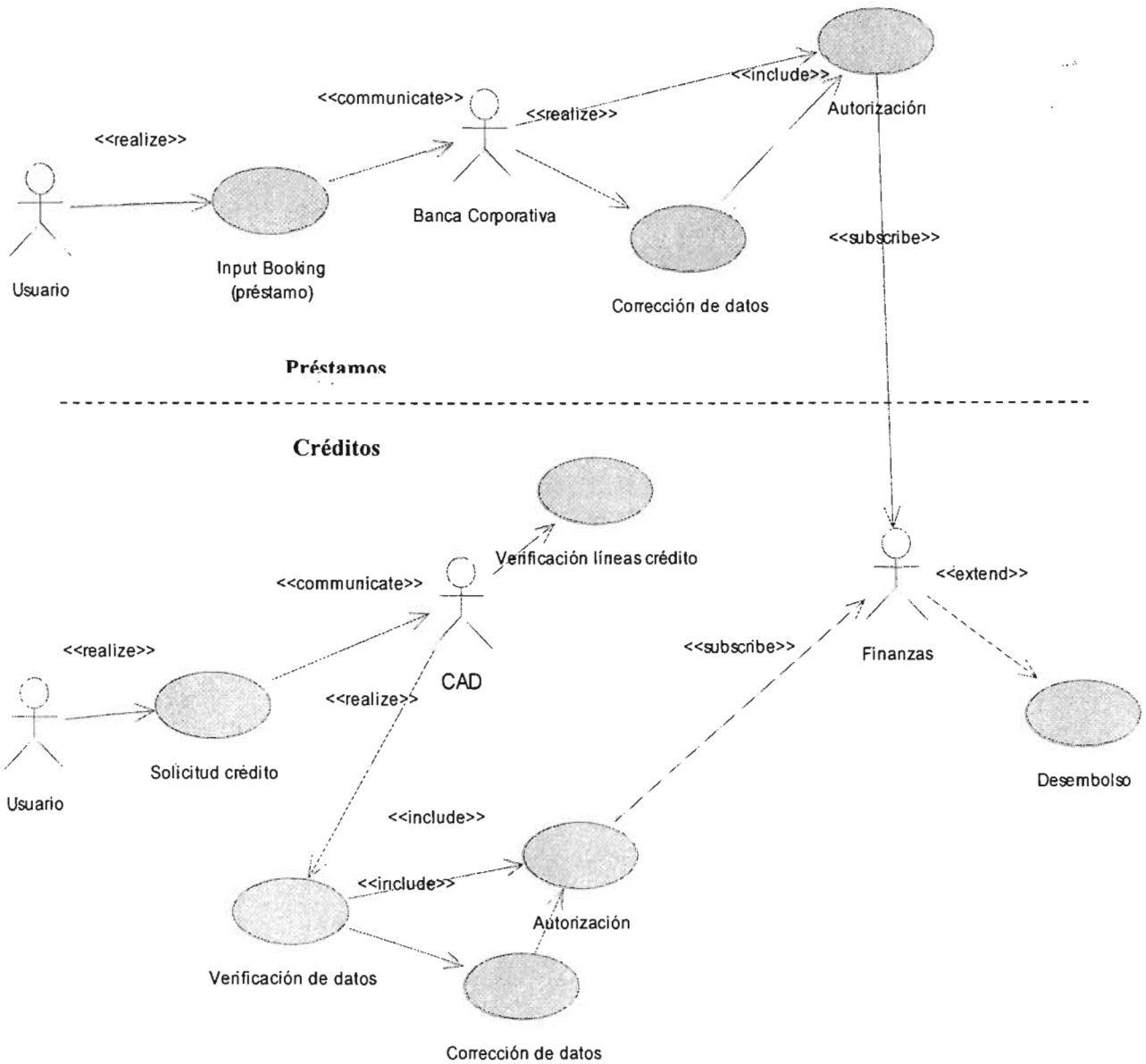


Figura 3.1.1 "Diagrama de procesos actuales"

3.2 Especificaciones para llevar a cabo el seguimiento del flujo de un préstamo

Es necesario hacer una descripción de las especificaciones para llevar el seguimiento del flujo de un préstamo, que es muy parecido al de un crédito. Esto, para conocer como se lleva actualmente el control de un préstamo, ya que dicho control, será ahora manejado por el componente *Web Service*.

- El flujo inicia con la captura de la información del préstamo ("Booking Template")
- Esta información deberá ser capturada por los usuarios del departamento de Banca Corporativa en el nuevo Sistema de Préstamos.
- Una vez que se ha guardado la información; el préstamo, deberá ser visto por el usuario RM (Relationship Manager), al que este relacionado el préstamo. También podrá ser visto por un usuario que tenga privilegios de BackUp del RM.
- Este usuario podrá determinar si el préstamo continúa su flujo, o es cancelado para que se corrija cualquier dato erróneo en la captura.
- Cuando hay error en algún dato de la captura, el RM tendrá la obligación de capturar una nota, en la que se especifique el problema que ha detectado, para que el usuario de Banca Corporativa que capturó tenga conocimiento y lo corrija.
- Si no hubo problemas cuando el RM firme electrónicamente, el área de CAD (Credit Adminitration Document) podrá ver el préstamo para que revise los datos.
- Este usuario podrá determinar si el préstamo continúa su flujo, o es cancelado para que se corrija cualquier dato erróneo en la captura e inicie otra vez el flujo.
- Cuando hay error en algún dato de la captura, el usuario responsable en el área de CAD, tendrá la obligación de capturar una nota en la que se especifique el problema que ha detectado,

para que el usuario de Banca Corporativa que capturó se entere y lo corrija.

- Si no hubo problemas cuando el usuario responsable del área de CAD lo revisa, entonces firma electrónicamente y el área de operaciones podrá ver el préstamo para que revise los datos.
- Este usuario podrá determinar si el préstamo continúa su flujo, o es cancelado para que se corrija cualquier dato erróneo en la captura e inicie nuevamente el flujo.
- Cuando hay error en algún dato de la captura, el usuario de operaciones tendrá la obligación de capturar una nota en la que se especifique el problema que ha detectado, para que el usuario de Banca Corporativa que capturo sepa y lo corrija.
- Si no hubo problemas cuando el usuario del área de operaciones lo revisa, entonces firma electrónicamente y se genera la interfaz a formato de pagos para que se emita el pago al cliente.
- Si durante el proceso de formato de pagos deciden cancelarlo, este deberá regresar un estado y una nota que especifique el problema al sistema de Préstamos.
- Si el proceso en formato de pagos es exitoso, (se emite el pago) este deberá regresar un estado al sistema de préstamos para que los usuarios puedan identificarlo.

La figura 3.1.2 "Flujo de datos de préstamos" muestra lo mencionado anteriormente.

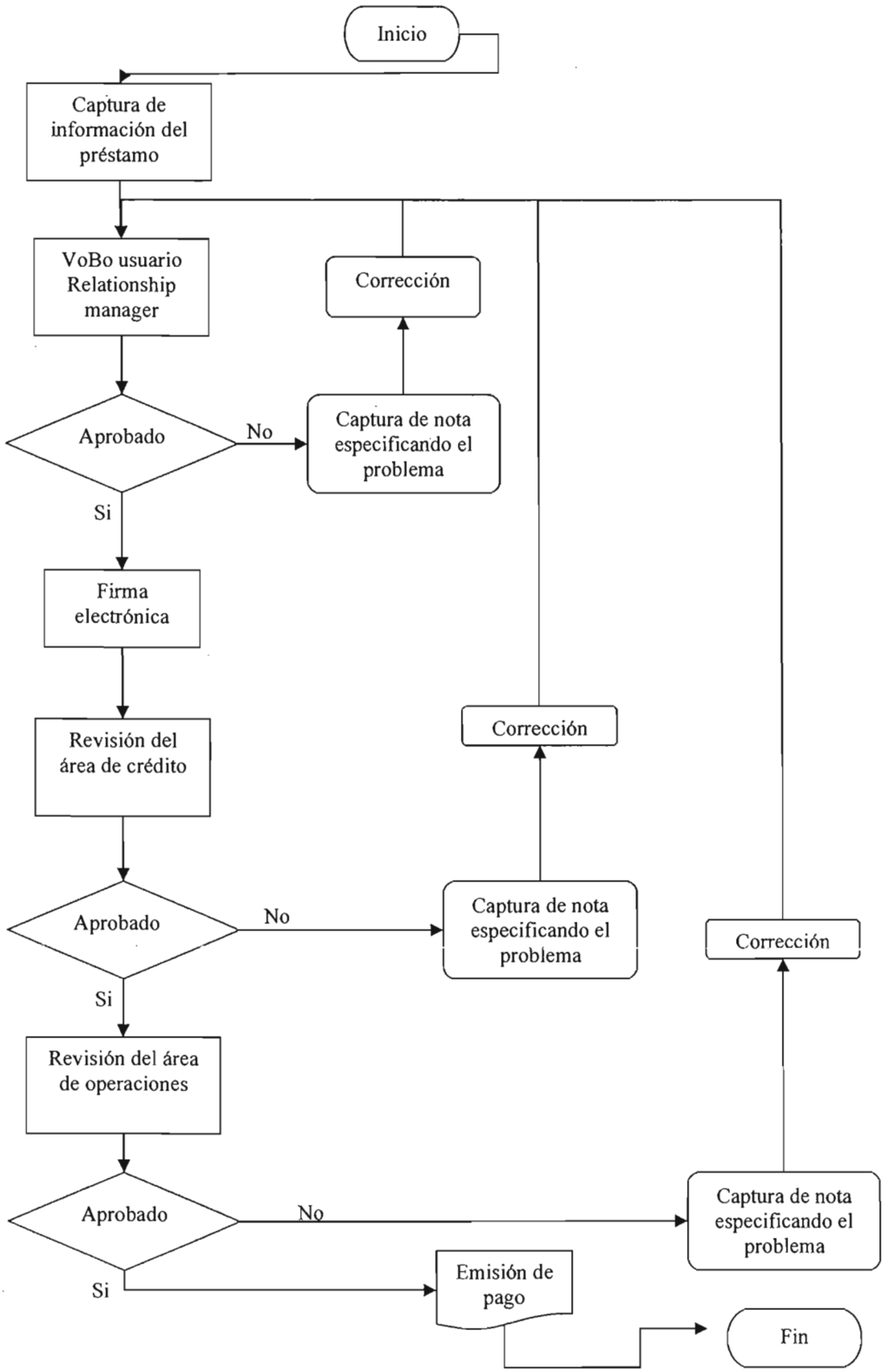


Figura 3.1.2 "Flujo de datos de préstamos"

3.3 Diseño arquitectónico del componente

El diseño arquitectónico del componente, es decir, la relación entre cada uno de los elementos estructurales del sistema que cumpla con los requerimientos de usuarios y de sistema quedará de la siguiente manera.

El componente contará con varios servicios que pondrá disponibles a las aplicaciones. Algunos de ellos son:

- Procesar solicitudes de créditos y préstamos.
- Generación de pagos.
- Envío automático de E-mail.
- Mantenimiento de catálogos.
- Generación de reportes.

Se creará un entorno con la tecnología que provee java *Web Services* para hacer que las aplicaciones ya existentes de crédito y préstamos trabajen sincronizadamente y se guarden nuevos registros en una base de datos común, además de los registros ya almacenados en sus respectivas bases de datos.

Es necesario hacer algunas modificaciones a las aplicaciones existentes para que se puedan comunicar con el componente.

Es importante contar con una aplicación cliente para acceder a los reportes solicitados por ambas áreas (créditos y préstamos), y verificar que las aplicaciones están trabajando correctamente. Esta aplicación leerá los registros de la nueva base de datos con información de ambos sistemas.

Esta aplicación cliente se desarrollará utilizando JSP (Java Server Pages), y servlets, creando páginas dinámicas y toda la lógica del negocio será realizada del lado del servidor.

La aplicación cliente será invocada desde un navegador en la máquina que vaya a utilizar el componente, el componente estará situado del lado del servidor de aplicación al igual que la interfase gráfica, y la base de datos estará en un servidor de base de datos.

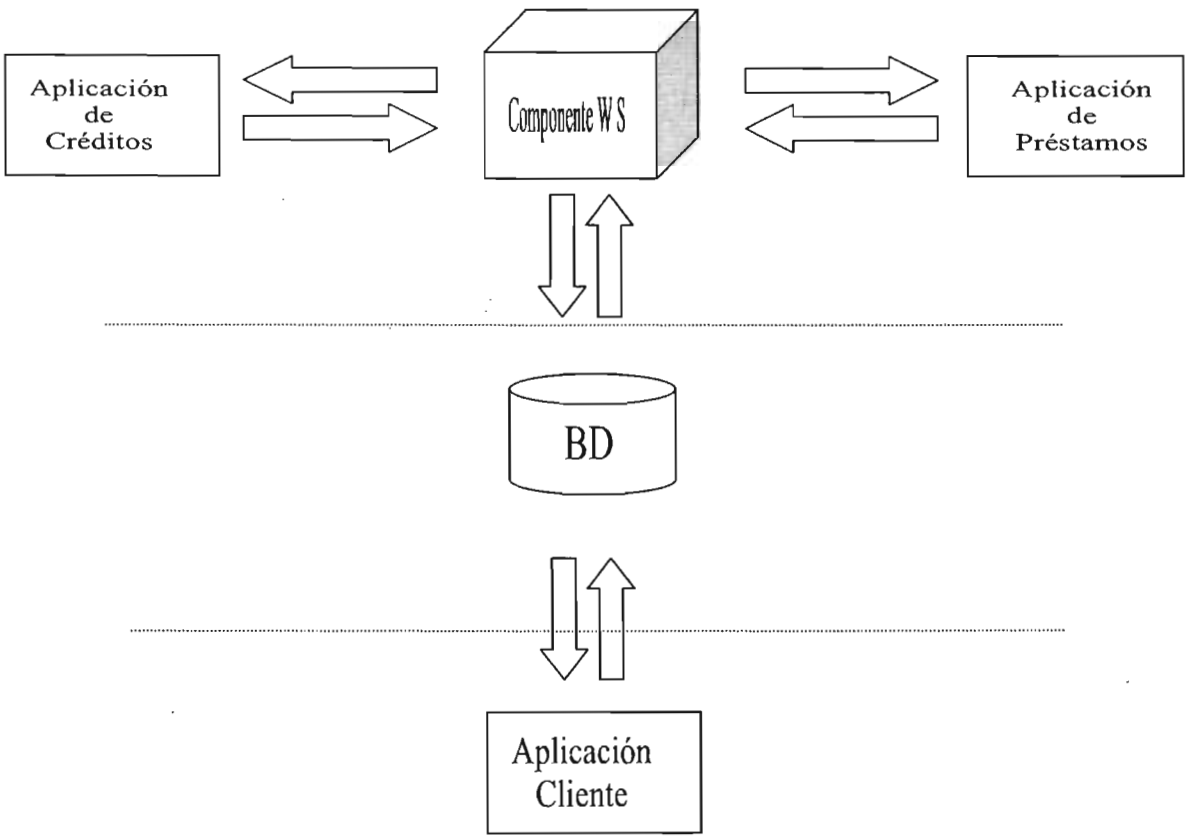


Figura 3.2.1 "Arquitectura general"

La figura 3.2.1 "Arquitectura general" muestra la arquitectura del nuevo componente desde un punto de vista global dividido en tres capas, componente y aplicaciones actuales, base de datos y aplicación cliente.

La figura 3.2.2 "Diseño arquitectónico" muestra más detalladamente la función de cada objeto dentro de la aplicación.

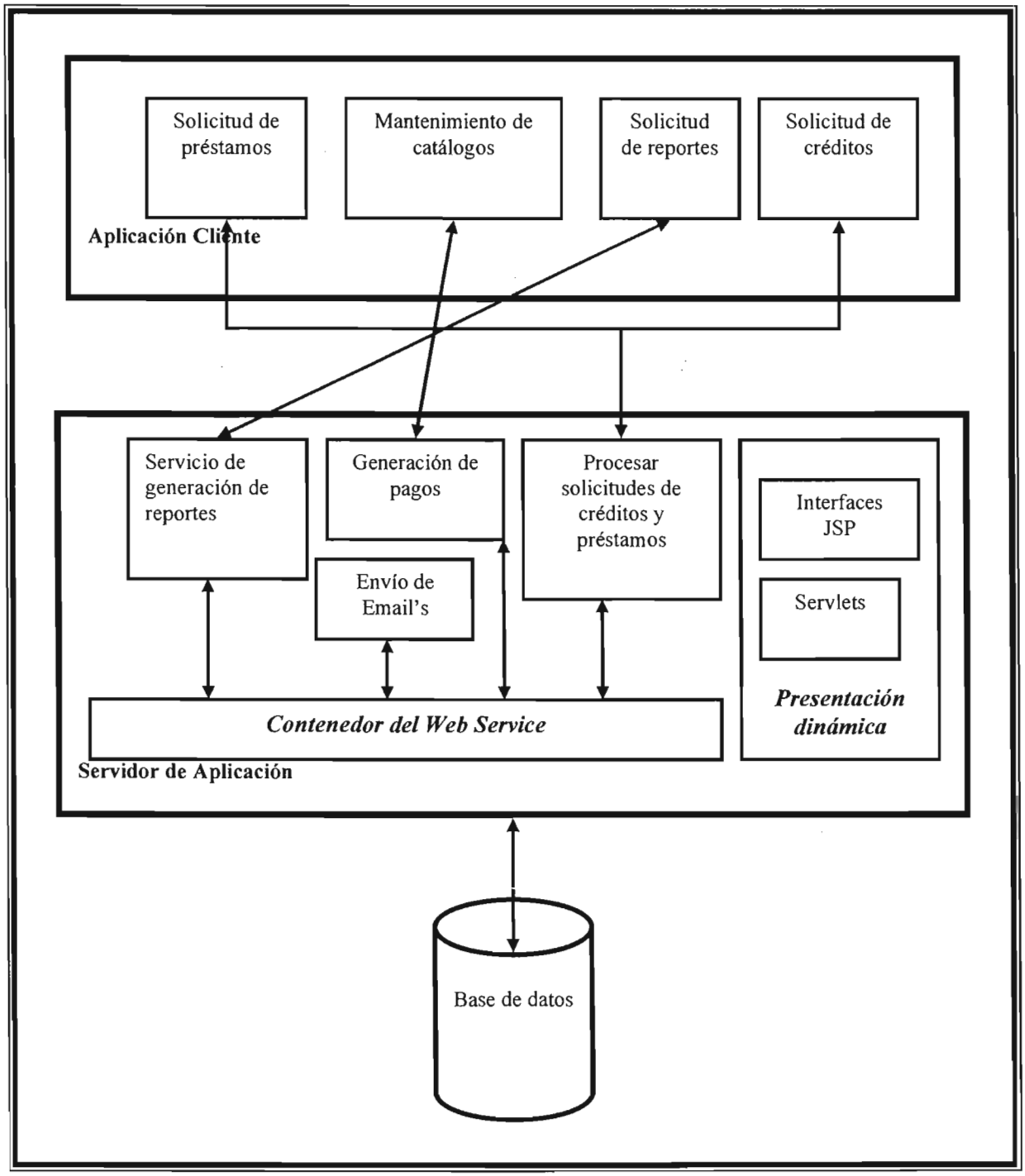


Figura 3.2.2 "Diseño arquitectónico"

3.4 Diseño modular

El componente estará formado por varios módulos, los principales se describen a continuación.

Solicitud de Operación.

En este módulo se procesarán las solicitudes de renovación o nuevos préstamos o créditos, estas solicitudes tendrán que ser evaluadas por su correspondiente áreas para ser corregidos en caso de que exista algún error. Posteriormente, se procesa la solicitud pasando por un análisis financiero para conocer el estado del cliente que está firmando la solicitud. En caso de ser aprobada la solicitud, se notifica al cliente.

En la figura 3.3.1 "Solicitud de operación", se muestra como ambos procesos solicitud de crédito y solicitud de préstamo son enviados a rechazar o aceptar la solicitud después de ser procesada, los procesos se inican cuando un usuario hace la solicitud de un nuevo préstamo o crédito. De igual manera, el usuario hace la solicitud de reporte de actividades y envío de datos generales al cuentahabiente.

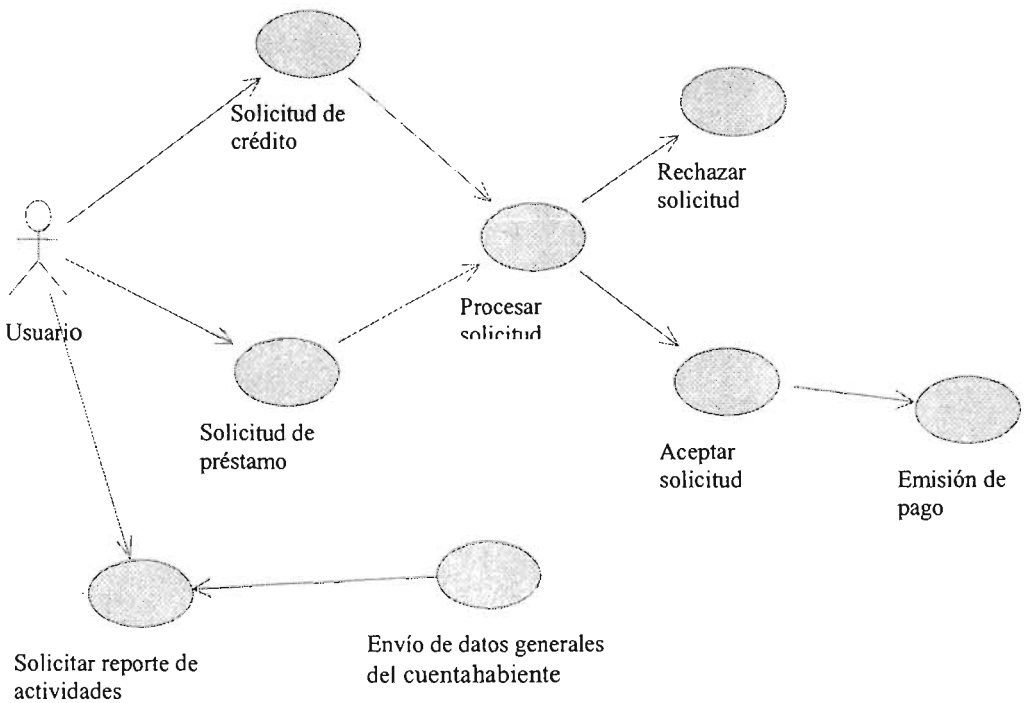


Figura 3.3.1 "Solicitud de operación"

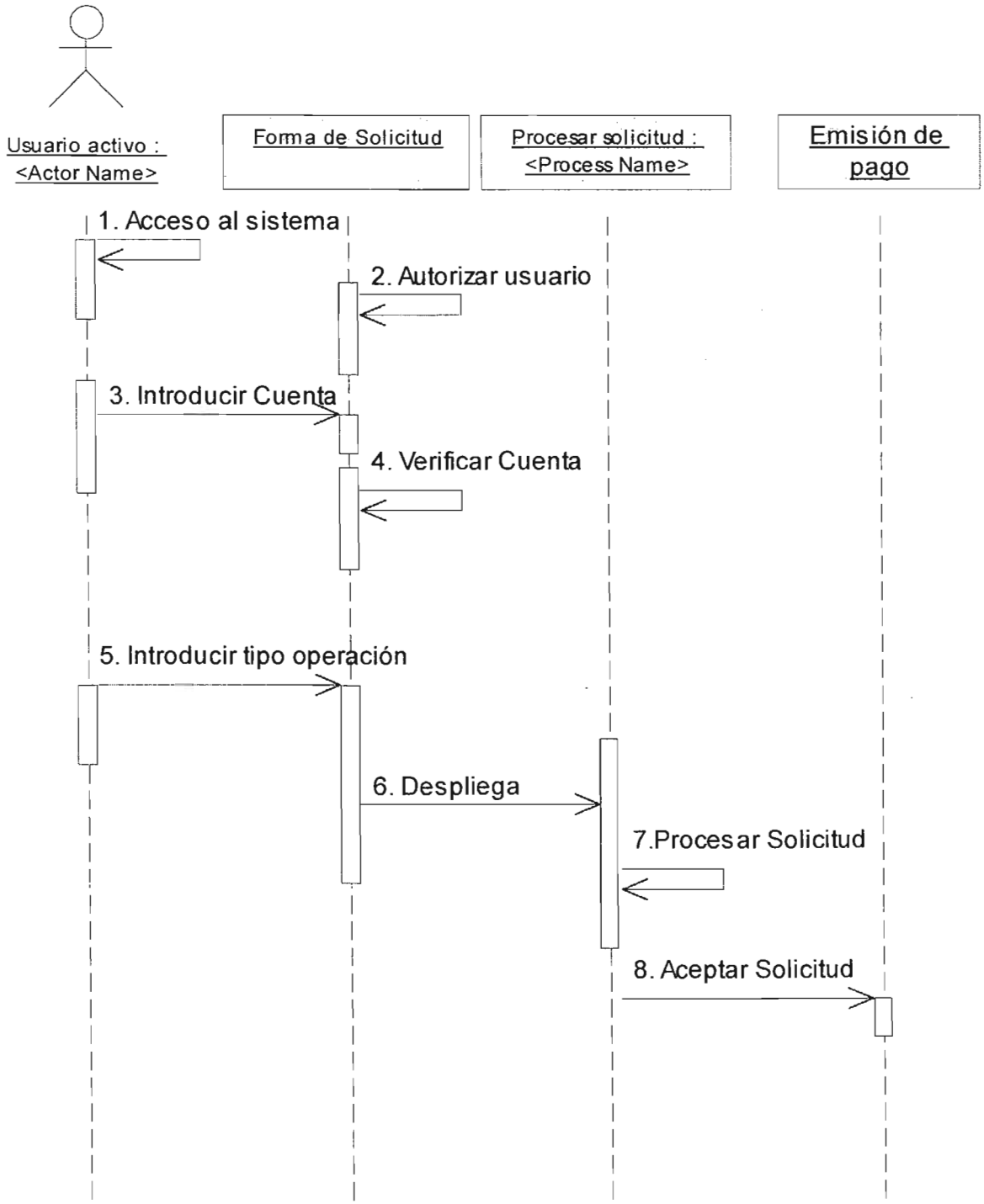


Figura 3.3.2 "Diagrama de secuencia de solicitudes"

En la figura 3.3.2 "Diagrama de secuencia de solicitudes" se puede observar el mecanismo de una solicitud al ser lanzada por el usuario, hasta ser rechazada o aceptada.

Definición de reportes

Aquí se definirá la estructura de los reportes que se puedan generar, en donde el solicitante podrá de manera dinámica, seleccionar los campos que quiera ver del reporte y establecer los criterios con que se va a formar el reporte. También podrá definir el campo de ordenamiento que tendrá el reporte. De esta manera, los reportes serán más útiles y más completos, ya que serán formados de acuerdo a las necesidades de cada usuario. También contará con la opción de exportar la información generada a un archivo de texto, o a una hoja de cálculo.

En la figura 3.3.3 "Definición de reportes", se muestran los procesos de solicitud de reporte, seleccionar el tipo de reporte, seleccionar los campos que van a formar el reporte, establecer los criterios y modificar la selección de los campos o envío de impresión.

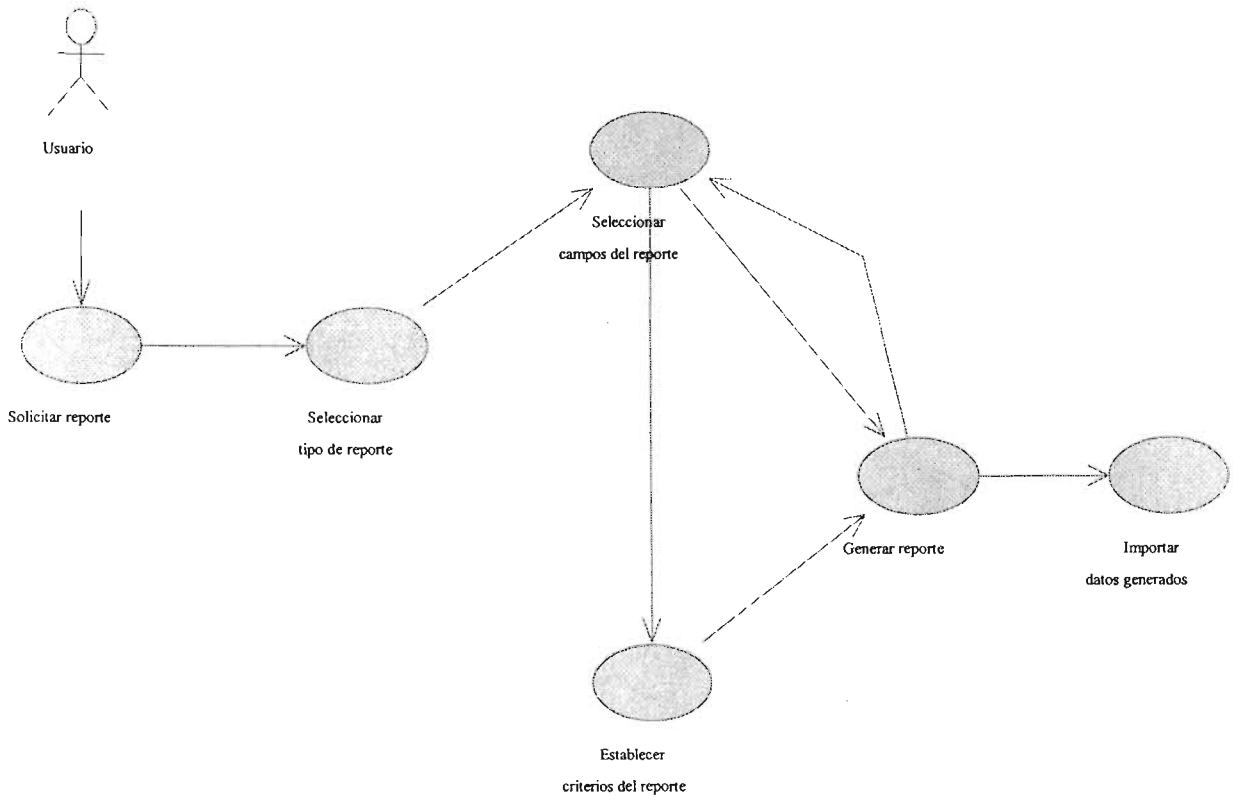


Figura 3.3.3 "Definición de reportes"

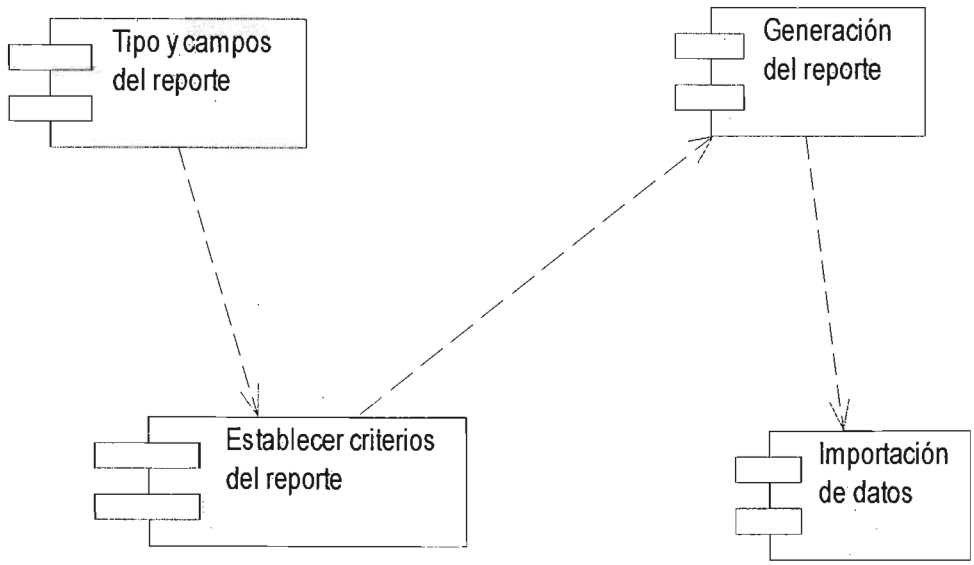


Figura 3.3.4 "Componentes de reportes"

La figura 3.3.4 "Componentes de reportes" muestra los principales componentes con los que contará el prototipo

Pagos por vencer.

En este módulo, se creará la interfase para el envío automático de correos a los clientes que tengan pagos próximos a vencer. Este correo será enviado con un día de anticipación antes de que el pago llegue a su vencimiento, y tendrá la opción de agregar o quitar clientes de la lista de envío, para evitar que lleguen correos no deseados, en caso de ser considerado así por el cliente. También se maneja la emisión de pagarés.

En la figura 3.3.5 "Pagos por vencer", el usuario es el actor que lanza el proceso de actualizar la lista de los cuentahabientes y da como resultado una lista actualizadas, la cual es enviada a la aplicación, que después de verificar si existen pagos próximos a vencer, lanza la emisión de pagarés y el envío de E-mail a los clientes que estén en la lista actualizada, notificándoles de sus próximos pagos.

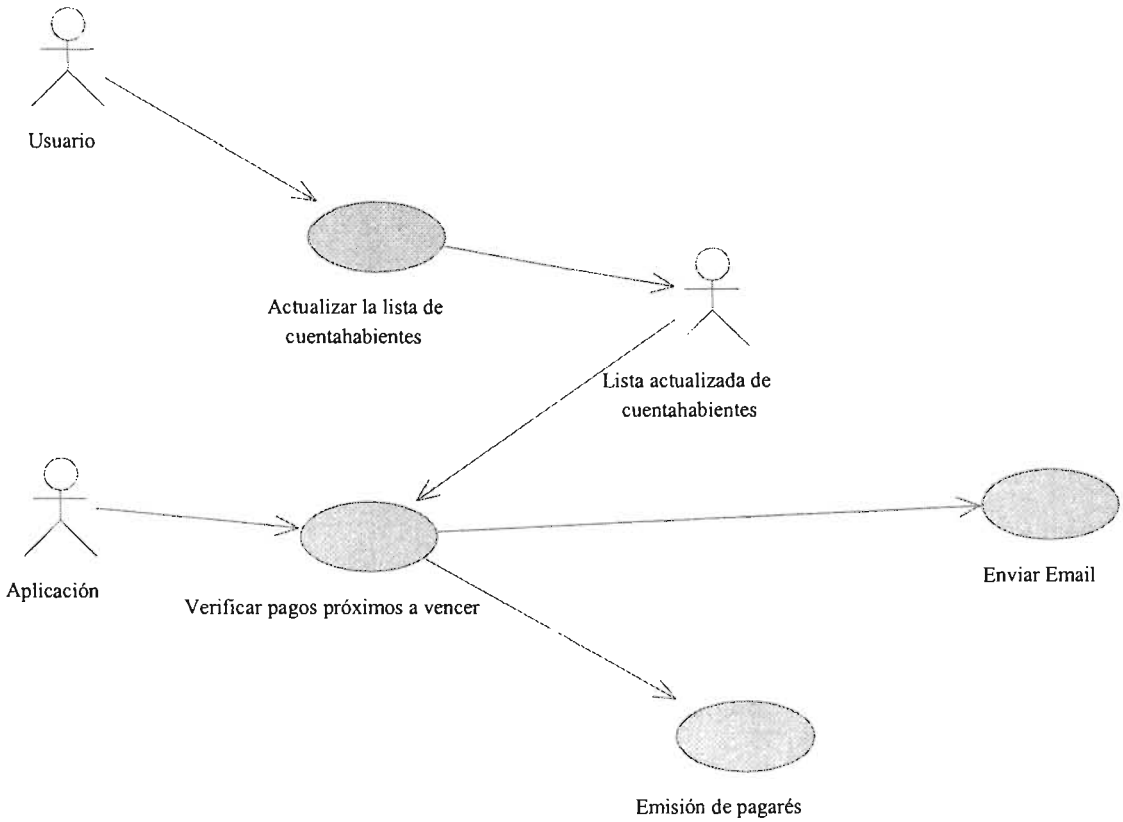


Figura 3.3.5 "Pagos por vencer"

Alcances del nuevo componente.

Los servicios que podrán ser invocados por las aplicaciones cliente son los siguientes:

- Recibir la solicitud de un nuevo crédito o préstamo y procesarlo
- Tratamiento para préstamos sindicados
- Dado un número de cliente, desplegar información acerca de sus movimientos y saldos, así como sus datos generales
- Administración de pagos
- Envío automático de E-mail por las operaciones a vencer en el día (Este deberá ser enviado en 3 noches anteriores al vencimiento)
- Manejo de parámetros de autorizaciones
- Emisión de estados de cuenta
- Validación de días hábiles
- Generación de amortizaciones[6]
- Renovación de préstamos y créditos
- Histórico de movimientos
- Obtención de tasas corporativas del sistema TAS (TIIE y LIBOR)[7]
- Actualización de tasas en automático
- Cierre de cálculo de procesos en automático
- Control de garantías para préstamos caucionados

[6] Amortización: Se denomina así a la representación contable de la depreciación monetaria que en el transcurso del tiempo sufren los activos inmovilizados.

[7] Tasa TIIE: Es la tasa de interés intercambiaria de equilibrio a plazo de 28 días, publicada diariamente por el Banco de México en el Diario Oficial de la Federación.
Tasa Libor: Es la tasa de interés de Londres (London Inter Bank Offered Rate) en dólares americanos a 3 meses, correspondiente al tercer viernes del mes inmediato anterior a aquel en que se devenguen los intereses, publicada en el Prontuario Internacional del Banco de México.

Contará con los siguientes reportes:

- Operaciones nuevas y en proceso
- Solicitudes denegadas
- Tasas de crédito
- Estados de cuenta individual y general
- Movimientos contables
- De PAS DUE (Clientes que no pagaron)
- De operaciones por cliente y tipo de operación

Dentro de los requerimientos que solicitan los usuarios quedan como opcionales los siguientes:

- Pantalla de facturación
- Inventario de facturas emitidas
- Control de documentos entregados y verificados

Quedan fuera del alcance de esta primera fase los siguientes puntos, pero a consideración para la segunda fase:

- Calculadora para prestamos de descuento
- Control absoluto de Nassau para operaciones desde México para CAD
- Inventario de pagares, letras de cambio, etc. por cliente y por operación
- Calculadora para determinación del IVA para operaciones de arrendamiento financiero
- Modulo de elaboración de pagares para recursos humanos
- Remainder de procesos con agenda para programación de operaciones
- Metrics (Reportes estadísticos)

Descripción de datos utilizados

Parte de la información que se estará enviando y recibiendo a manera de peticiones y respuestas por medio de XML por parte de los sistemas involucrados es:

- IDC (Identificador del cuentahabiente). Es, el número de identificación general del cuentahabiente reconocido por el sistema.
- Tipo de operación. Que tipo de operación es la que se ha de procesar o consultar.
- IDP (Identificador de operación). Número de identificación de la operación.
- Fecha de operación. Fecha en que se realizó la operación.
- Fecha de vencimiento. Fecha de vencimiento de la operación.
- Monto Inicial. El monto con el que se inició la operación.
- Tipo de moneda. La divisa o tipo de moneda con la que se pactó la operación.
- Tasa de interés. El porcentaje que llevará la tasa de interés sobre el monto adeudado.

Con estos datos se pueden generar los procesos de solicitud de crédito o de préstamo, así como de solicitud de reporte de operaciones vencidas, operaciones en espera y estados de cuenta del cuentahabiente, que son algunos de los principales servicios que ofrecerá el componente *Web Service*.

Estos datos se cargarán en una tabla principal que lleva por nombre Operaciones, que va relacionada con la tabla de Clientes, Créditos, Préstamos, Tasas_Interes, y Tipos_Monedas.

3.5 Aplicación cliente

Es importante contar con una aplicación que haga el papel de cliente y que utilice algún servicio del componente, para que sirva de ejemplo y poco a poco ir adaptando todas las aplicaciones existentes dentro de la institución para que todas trabajen como una sola sin dejar de cumplir con los requerimientos para las cuales fueron creadas.

La aplicación cliente será la encargada de proporcionar una interfase gráfica para que los usuarios puedan explotar los datos generados por el componente *Web Service*. Para cubrir los requisitos de la institución, se debe incorporar el esquema de seguridad que actualmente manejan algunas aplicaciones desarrolladas en Power Builder, este esquema incluye accesos, mecanismos de bloqueos de menús, y conexiones a la base de datos.

Esta aplicación estará disponible para todos los usuarios que cuenten con los privilegios necesarios.

Esta aplicación será desarrollada en entorno de páginas dinámicas, utilizando la plataforma proporcionada por J2EE, con servlets y Java Server Pages.

Se ha elegido java por las siguientes características:

- Sintaxis muy parecida a C
- Orientado a objetos
- Distribuido
- Robusto
- Seguro
- Portable
- Uso eficiente de memoria
- Concurrente (MultiThread)
- Permite reutilización de componentes

En la figura 3.7.1 "Interacción entre aplicaciones" se aprecia como están ligadas las aplicaciones por medio del componente Web Service y la aplicación cliente JSP.

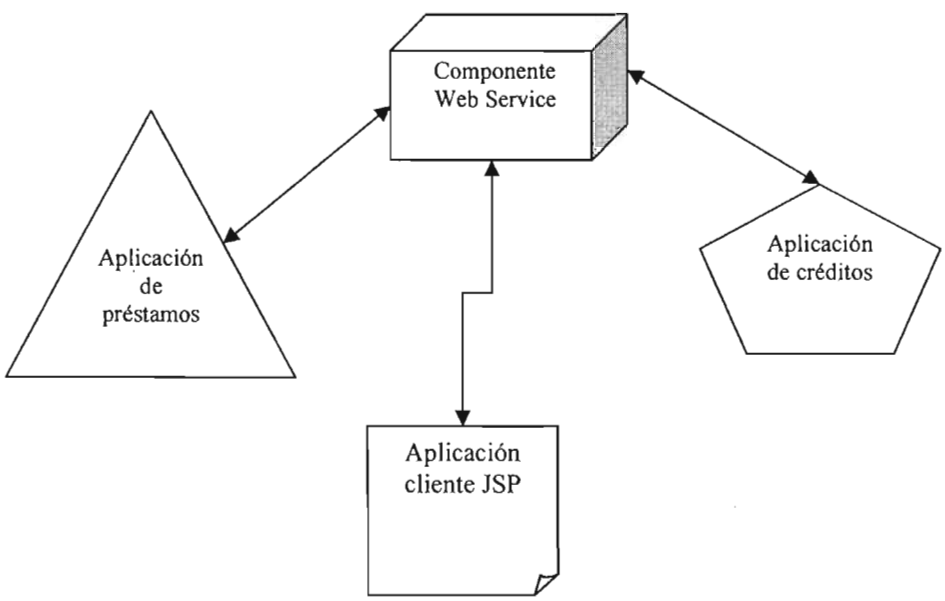


Figura 3.7.1 "Interacción entre aplicaciones"

Con esta parte ya se tienen los aspectos necesarios para comenzar con la construcción del prototipo y ponerlo a prueba.

4 Desarrollo del prototipo

Objetivos

- Dar a conocer la metodología utilizada.
- Mostrar el desarrollo del componente.
- Desarrollar la interfaz cliente, para poder acceder a los servicios ofrecidos

En este capítulo se desarrolla el prototipo del componente *Web Service* de acuerdo a las especificaciones citadas en la etapa de diseño, así como bloques de código de ejemplos. Se presenta también un panorama general de las funciones y métodos que incluirá el prototipo.

4.1 Prototipo

Es un diseño experimental de todo un sistema. El propósito de un prototipo es probar ciertos aspectos o características de un nuevo sistema. Un prototipo es una escala o modelo parcial, puede o no ser parcialmente funcional, o puede ser un modelo de objetos y procesos, los cuales son ejecutados en un entorno de pruebas, ya que tiene funcionamiento limitado en cuanto a capacidades, confiabilidad y eficiencia.

Los prototipos pueden ser muy útiles durante la etapa de desarrollo, ya que a menudo son utilizados para evaluaciones y pruebas.

4.2 Metodología

Durante el desarrollo de este proyecto, la metodología a la que se apegó el equipo de desarrollo, es la XP (eXtrem Programming), debido a que todo el trabajo fue orientado directamente al objetivo, basado en las relaciones interpersonales y velocidad de reacción. Se intentó minimizar el riesgo de fallo del proceso por medio de la disposición permanente de un representante competente del cliente en condiciones de contestar rápida y efectivamente a cualquier pregunta requerida para el desarrollo, de manera que no hubo retraso en la toma de decisiones.

Todos los desarrolladores conocían cada parte del software que se estaba desarrollando, para en caso de requerirse algún cambio, cualquiera lo pueda hacer sin mayor problema. Se definió una etapa de iteraciones durante las cuales se discutieron los objetivos de las mismas junto con el representante del cliente.

El resultado de cada iteración se transmitió al cliente para que lo juzgue. En base a su opinión se definieron las siguientes iteraciones del proyecto, y si el cliente no estuvo contento, se adaptó el plan de

liberaciones e iteraciones hasta que el cliente dió su aprobación y el software estuvo a su gusto.

Se programó sólo la funcionalidad requerida para la liberación actual. Se siguió un diseño evolutivo con la siguiente premisa: conseguir la funcionalidad deseada de la forma más sencilla posible.

También se tomó en cuenta la Ingeniería del Software, que implica que un programa bien estructurado satisfaga las siguientes condiciones:

- Contar con una estructura general en forma de módulos, que a su vez esten formados por procedimientos o segmentos (en función del lenguaje empleado).
- Debe existir una interfase claramente definida entre los diversos módulos.
- Cada módulo debe de ser una combinación sencilla de construcciones elementales de un lenguaje de programación.
- Debe existir una fuerte correspondencia entre la estructura de los módulos y la de los datos sobre los que operan.
- Cada módulo debe dejar las estructuras de datos sobre las que opera en un estado consistente con su definición.
- Un módulo no debe tener efectos secundarios

4.3 Desarrollo del componente

Pensando que la arquitectura J2EE es la más madura para la implementación de la lógica de negocios, en presentar los *Web Services* como una extensión natural del modelo de componentes EJB de J2EE y utilizando XML como base del protocolo entre componentes, se decidió utilizar la arquitectura J2EE para la implantación.

Para la creación del componente, se ha utilizado el estilo XML-RPC, basado en la plataforma J2EE 1.4 con el servidor de aplicación WASP (Web Application and Services Plataform), este es un servidor de aplicación desarrollado por la empresa Systinet.

Las herramientas que se van a utilizar para el desarrollo del componente son Java J2EE, JCreator, WASP Server, Dreamweaber y MySql.

Los pasos par crear el componente fueron:

- Crear la lógica de negocio. Primero es necesario programar las clases que implementen la lógica del negocio del *Web Service*. Un ejemplo puede ser una clase que reciba el IDC (Id del cliente), el IDP (Id del Proceso) y el monto, para que muestre el estado de cuenta del cliente. Después de evaluar el cliente y de hacer el respectivo análisis financiero, acepta o rechaza la solicitud del préstamo o crédito bancario.

- Desarrollar las clases java en el servidor SOAP. Después, es necesario empaquetar las clases y desplegarlas para convertirlas en un Web Service. El despliegue de estas clases se hará en el servidor WASP SOAP, usando la herramienta de despliegue de WASP.

- Generar las clases de acceso del cliente: Una petición de un cliente utiliza un objeto próxy para acceder al cliente. Al momento de recibir la petición, el próxy acepta un método de java llamado desde la aplicación, y lo traduce a un mensaje de petición SOAP. Al momento de responder, el próxy recibe el mensaje SOAP de

respuesta, lo traduce en un objeto java, y regresa el resultado a la aplicación cliente.

- Desarrollo de la aplicación cliente. La aplicación cliente trata al próxy como un objeto java estándar, que transparentemente establece la comunicación con el Web Service.

El primer paso para la creación del componente es construir las clases que conformará el servicio, en las cuales se declaran los métodos que el cliente va a poder invocar. Para esto es necesario extender la interfaz *java.rmi.Remote* y no manejar declaraciones constantes públicas estáticas en las clases principales.

Los métodos de estas clases arrojarán sus propias excepciones en caso de que las hubiera, y por último, regresar tipos de dato soportados por la arquitectura JAX-RPC.

El componente contará inicialmente con una interfase que utiliza 3 métodos principales OperationMet y reportMet, SendEmMet con las siguientes características:

OperationMet: este método toma el IDC (Identificador del cuentahabiente), date_op (fecha de operación), amount (Monto inicial), currency (tipo de moneda), user_id (Identificador del usuario que captura la operación), type_operation (tipo de operación) e int_rate (tasa de interés). Con esta información, guarda la operación en la base de datos para posteriormente generar los reportes diarios y mensuales. Regresa un booleano indicando si la operación se realizó con éxito (0 éxito, 1 error).

reportMet: este método recibe como parámetros rep_type (tipo de reporte) y date_interval (el intervalo de las fechas en el que se requiere el reporte)

El objeto de la interfaz será un paquete llamado OpdbPak, el cual es una extención de la Interface *java.rmi.Remote*

SendEmMet: Aquí es donde se procesan los pagos próximos a vencer para enviar la notificación de manera automática al cliente para que haga el correspondiente depósito.

4.4 Desarrollo de las clases

Cuando todas las clases están compiladas exitosamente, se empaquetan para después hacer el despliegue de componentes y hacer el servicio disponible.

En la figura 3.5.1 "Diagrama de clases principales" se muestra un diagrama, donde se muestran las clases principales que se implementarán en el ambiente.

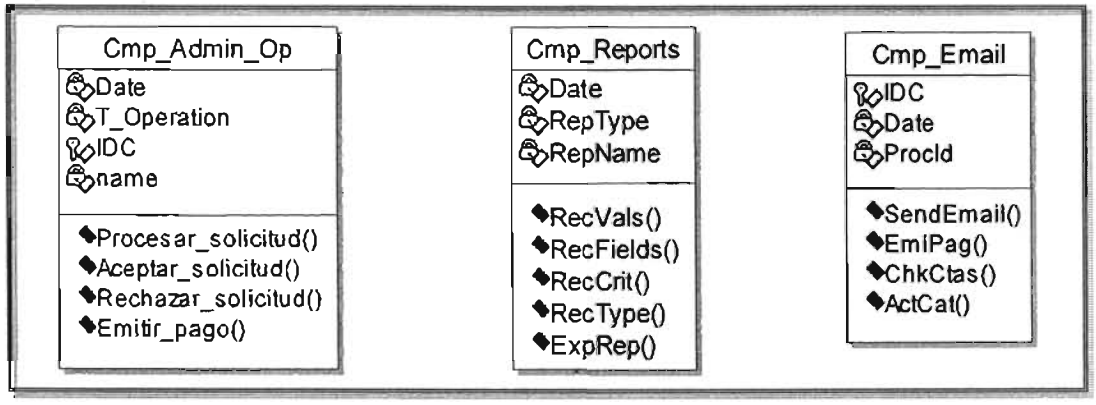


Figura 3.5.1 "Diagrama de clases principales"

Cmp_Admin_OP

Procesar_solicitud(): en este método se evalúa la solicitud para ser aceptada o rechazada por las áreas involucradas.

Aceptar_solicitud(): este actúa cuando una solicitud de crédito o préstamo es aceptada, se calculan las tasas de interés, se establecen las fechas de pago, el tipo de moneda, el plazo, etc.

Rechazar_solicitud(): este método se ejecuta cuando una solicitud es rechazada por algún motivo en el proceso de procesar_solicitud(), lleva control de los usuarios que la rechazan y las causas por la que no fue aceptada.

Emitir_pago(): cuando una solicitud de crédito o préstamo es aceptada, se emite un pago por la suma acordada, y es cuando este proceso es invocado.

Cm_Reports

RecVals(): aquí se evalúan los campos de entrada para la generación de los reportes.

RecFields(): este método recopila todos los campos que el usuario seleccione para que salgan en el reporte que está solicitando.

RecCrit(): aquí se procesan los criterios de filtración que utilizará el reporte.

RecType(): hace la diferencia entre los diferentes tipos de reporte que se pueden manejar en la aplicación.

ExpRep(): con este método se pueden exportar los datos hacia otra aplicación para ser manipulados más fácilmente como por ejemplo a una hoja de cálculo o un archivo de texto.

Cmp_Email

SendEmail(): en este método se envían los correos electrónicos a los clientes que no estén marcados como "no enviar correo", notificándoles de su próxima fecha de pago.

EmiPag(): aquí se revisan los pagos próximos a vencer para notificarle al cliente por correo electrónico.

ChkCtas(): checa que estén actualizadas las cuentas de los clientes, de no ser así, invoca al método correspondiente para actualizarlas.

ActCat(): aquí se actualizan los datos de las cuentas de los clientes.

4.5 Funciones y métodos

Para mostrar como se implementan las funciones y métodos que va a exponer el componente *Web Service*, se muestra a continuación una clase java que implementa la función de búsqueda de operaciones de un determinado cuenta habiente. Esta operación puede ser Crédito, Préstamo o Pago.

```
package com.systinet.webService.FindOper;
public class FindOperationsService {
/* Este método hace la búsqueda del tipo de servicio que se está invocando*/
 public int  getOp(String CGI) {
      if(CGI!=null && OPType.equalsIgnoreCase("PRESTAMO"))
        return PrestID;
      if(CGI!=null && OPType.equalsIgnoreCase("CREDITO"))
        return CredID;
      if(CGI!=null && OPType.equalsIgnoreCase("PAGO"))
        return PagoID;
       }
/*Agrega una lista con las operaciones disponibles    */
public java.util.LinkedList getAvailableOperation() {
      java.util.LinkedList list = new java.util.LinkedList();
      list.add("PRESTAMO");
      list.add("CREDITO");
      list.add("PAGO");
      return list;
   }
}
```

Esta clase, es parte del módulo que recibe solicitudes de Créditos o Préstamos, para ser procesados mediante una serie de pasos descritos con anterioridad en el capítulo III de este trabajo.

Para convertir estas clases en un *Web Service*, se compilan y después se despliegan en el servidor WASP con ayuda de la herramienta de despliegue.

Ya desplegado el componente, desde la consola de administración ubicada en navegador **http://localhost:6060/admin/console** se puede ver

una lista de todos los paquetes desplegados en el servidor. El paquete FInOper debe ser desplegado con un *FindOperationsService* funcionando en el entorno de ejecución. El entorno de ejecución de Web Services genera automáticamente el archivo WSDL y lo publica en la dirección **http://localhost:6060/FindOperationsService/**.

El archivo WSDL generado es el siguiente:

```
// El siguiente código es el resultado de la compilación del WSDL, en formato XML
<?xml version='1.0'?>
<wsdl:definitions name='com.systinet.demos.stock.FindOperationsService '
targetNamespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/Services/Fin
dOperations/'
   xmlns:mime='http://schemas.xmlsoap.org/wsdl/mime/'
   xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
   xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:ns0='http://idoox.com/containers'
   xmlns:http='http://schemas.xmlsoap.org/wsdl/http/'
xmlns:tns='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/Services/FindOpera
tions/'
   xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'>
   <wsdl:message name='FindOperationsService _getOP_Request'>
      <wsdl:part name='p0' type='xsd:string'/>
   </wsdl:message>
   <wsdl:message name='FindOperationsService _getOP_Response'>
      <wsdl:part name='response' type='xsd:double'/>
   </wsdl:message>
   <wsdl:message name='FindOperationsService _getAvailableOperation_Request'/>
   <wsdl:message name='FindOperationsService _getAvailableOperation_Response'>
      <wsdl:part name='response' type='ns0:LinkedList'/>
   </wsdl:message>
   <wsdl:portType name='FindOperationsService '>
      <wsdl:operation name='getAvailableOperation'>
         <wsdl:input name='getAvailableOperation'
            message='tns:FindOperationsService _getAvailableOperation_Request'/>
         <wsdl:output name='getAvailableOperation'
            message='tns:FindOperationsService _getAvailableOperation_Response'/>
      </wsdl:operation>
      <wsdl:operation name='getOP' parameterOrder='p0'>
         <wsdl:input name='getOP'
            message='tns:FindOperationsService _getOP_Request'/>
```

```
        <wsdl:output name='getOP'
          message='tns:FindOperationsService _getOP_Response'/>
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name='FindOperationsService ' type='tns:FindOperationsService '>
      <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='rpc'/>
      <wsdl:operation name='getAvailableOperation'>
        <soap:operation soapAction='' style='rpc'/>
        <wsdl:input name='getAvailableOperation'>
          <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/Services/FindOper
ations/'/>
        </wsdl:input>
        <wsdl:output name='getAvailableOperation'>
          <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'

namespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/Services/FindOper
ations/'/>
        </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name='getOP'>
        <soap:operation soapAction='' style='rpc'/>
        <wsdl:input name='getOP'>
          <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/Services/FindOper
ations/'/>
        </wsdl:input>
        <wsdl:output name='getOP'>
          <soap:body use='encoded'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
namespace='http://idoox.com/wasp/tools /output/com/systinet/Services/FindOperations/'/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name='JavaService'>
      <wsdl:port name='FindOperationsService ' binding='tns:FindOperationsService '>
        <soap:address location='http://localhost:6061/FindOperationsService /'/>
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

4.6 Intercambio de datos

Además de intercambiar datos primitivos como strings, int o double, también es necesario intercambiar mensajes complejos a través del protocolo SOAP, ya que es la manera más adecuada para traducir tipos complejos a XML, pues algunas traducciones ocurren automáticamente como son:

- Tipos primitivos de Java.
- Clases personalizadas, que son mapeadas utilizando un patrón Java Bean.
- Colecciones de Java2 como Hashtable, HashMap, Vector, List, etc.

En la siguiente clase se pasa una estructura reqSolicitud al *Web Service*. Esta estructura está implementada con un Java Beans, con métodos get y set para aceptar o rechazar la solicitud pedida. El método de servicio procesar_solicitud acepta la clase solicitudRequest como único parámetro. El método getSolicitud devuelve la colección de todos los datos relacionados con las solicitudes de préstamos o de créditos, incluyendo entre ellos la aceptación o rechazo. El método getSolicitud utiliza el contenedor java.util.Hashtable como valor de retorno.

```
package com.operaciones.solicitudes;
//En este método se forma la solicitud y se asignan valores al objeto solicitud
public class SolicitudService {
   private java.util.HashMap solicitud = new java.util.HashMap();
   public String processSolicitud(solicitudRequest solicitud) {
      String result = "PROCESANDO SOLICITUD";
      //Se forma la llave con la hora del servidor
      Long id = new Long(System.currentTimeMillis());
     //Aquí se forma la cadena que será enviada
      result      += "\n----------------------------";
      result      += "\nID:          "+id;
      result      += "\nTYPE:        "+
 ((solicitud.getType()==solicitud.SOLICITUD_TYPE?("PRESTAMO"):("CREDITO"));
      result      += "\nOPERACION:       "+solicitud.getOperation();
      result      += "\nMONTO :    "+solicitud.getAmount();
      result      += "\nTASA DE INTERES:       "+solicitud.getInterestRate();
      this.solicitud.put(id,solicitud);
      return result;
```

```
    }
    //Este método regresa un HashMap conteniendo los parámetros de la solicitud
    public java.util.HashMap getSolicitud() {
        return this.solicitud;
    }
}
```

4.7 Mapeo de tipos complejos

Con la ayuda de la herramienta *wscompile* se generan los archivos necesarios para el funcionamiento del servicio. Esto se consigue con el siguiente comando:

prompt> wscompile -define -mapping build/mapping.xml -d build -nd build -classpath build config.xml

Este comando lee el archivo config.xml, crea el archivo OpdbService.wsdl, y mapping.xml en el directorio build. Los parámetros enviados al comando significan lo siguiente:

-define: ordena a la herramienta leer el objeto de la interface para crear el archivo WSDL.

-mapping: especifica el archivo de mapeo y donde generarlo.

-d y –nd: especifican donde generar los archivos resultantes, las clases y no clases respectivamente.

Al momento de hacer el despliegue se genera un archivo WSDL en http://localhost:6060/MappingService/, que es el que describe la funcionalidad del servicio, como se comunica y como se puede acceder, proporcionando un mecanismo más estructurado para descubrir las operaciones que puede realizar el *Web Service*. La parte más importante es la definición del mapeo del tipo Java SolicitudRequest:

```
/*Este fragmento de código es una parte del WSDL generado por el servidor, al hacer el
despliegue*/
<xsd:complexType name="Solicitud
  <xsd:sequence>
    <xsd:element name="limitAmount="xsd:double"/>
    <xsd:element name="OperayionId="xsd:short"/>
    <xsd:element name="type" type="xsd:string"/>
```

```
  <xsd:element name="InterestRate" type="xsd:long"/>
 </xsd:sequence>
</xsd:complexType>
```

El mapeo de SolicitudRequest a XML está definido como un conjunto de campos de tipos primitivos. El HashMap devuelto por el mapeo getSolicitud importa la siguiente definición:

```
//Definición de elementos del HashMap devuelto por getSolicitud
<complexType name="HashMap">
 <sequence>
  <element name="Solicitud" minOccurs="0" maxOccurs="unbounded">
   <complexType>
    <sequence>
     <element name="key" type="int" />
     <element name="value" type="int" />
    </sequence>
   </complexType>
  </element>
 </sequence>
</complexType>
```

Archivo de configuración.

Es un archivo de configuración XML, el cual será procesado a través de la herramienta *wscompile*, y el nombre del archivo es config.xml. En este archivo de configuración se describe el nombre del servicio, su namespace[8], el nombre del paquete que lo contiene (OpdbPak), y el nombre de la interface (OpdbImp).

El archivo queda de la siguiente forma:

```
//Archivo generado por wscompile
<?xml version="1.0" encoding="UTF-8"?>
<configuration
 xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
```

[8] NameSpace es un nombre utilizado para identificar elementos y nombres de atributos en un documento XML.

```
 <service
    name="OpdbService"
    targetNamespace="urn:Opdb"
    typeNamespace="urn:Opdb"
    packageName="OpdbPak">
    <interface name="OpdbImp"/>
 </service>
</configuration>
```

Este archivo le dice al *wscompile* que cree un archivo con la siguiente información:

El nombre del servicio es OpdbService.

El WSDL namespace es urn:Opdb.

Las clases para el servicio estan en el paquete OpdbPak, bajo el directorio build.

El nombre de la interface del servicio es OpdbImp

4.8 Interfase cliente

La aplicación del lado del cliente simplemente crea dos tipos de solicitudes (préstamo o crédito), y los envía al *Web Service*. Después recupera el Hashtable con la respuesta de estas solicitudes y los muestra en la consola.

Parte del código de la aplicación cliente:

```
package com.systinet.demos.mapping;
import org.idoox.wasp.Context;
import org.idoox.webservice.client.WebServiceLookup;
public final class TradingClient {
  /*Método main en donde hace la búsqueda del servicio, se definen los límites para cada
operación y se recibe en un HashMap */
  public static void main( String[] args ) throws Exception {
    WebServiceLookup lookup =
     (WebServiceLookup)Context.getInstance(Context.WEBSERVICE_LOOKUP);
    OrderServiceProxy service =
     (OrderServiceProxy)lookup.lookup("http://localhost:6060/MappingService/",
```

```
        SolicitudServiceProxy.class);
    com.operaciones.solicitudes.SolicitudRequest solicitud =
      new com.operaciones.solicitudes.struct.SolicitudRequest();
    solicitud.symbol = "PRESTAMO";
    solicitud.type =
com.operaciones.solicitudes.SolicitudRequest.SOLICITUD_TYPE_PRESTAMO;
    solicitud.limitAmount = 10000000;
    String result = service.processOrder(solicitud);
    System.out.println(result);
    solicitud = com.operaciones.solicitudes.struct.OrderRequest();
    solicitud.symbol = "CREDITO";
    solicitud.type = com.operaciones.solicitudes.OrderRequest.ORDER_TYPE_CREDITO;
    solicitud.limitAmount = 10000000;
    result = service.processOrder(solicitud);
    System.out.println(result);
   /* Obtiene el tipo de solicitud en el HashMap, enviado del método  getSolicitud
     java.util.HashMap solicitudes = service.getSolicitudes();
     java.util.Iterator iter = solicitudes.keySet().iterator();
     while(iter.hasNext()) {
         Long id = (Long)iter.next();
         OrderRequest req = (OrderRequest)solicitudes.get(id);
         System.out.println("\n----------------------------");
       System.out.println("\nID:          "+id);
       System.out.println("\nTYPE:        "+
((req.getType()==com.operaciones.solicitudesOrderRequest.SOLICITUD_TYPE_PRESTA
MO)?("PRESTAMO"):("CREDITO")));
       System.out.println("\nFECHA:       "+req.getDate());
       System.out.println("\nMONTO:    "+req.getAmount());
       System.out.println("\nTASA INTERES:     "+req.getInterestRate());
       System.out.println("\nTIPO OPERACION:       "+req.getOperType());
       System.out.println("\nID OPERACION:       "+req.getOperId());
     }
    }
}
```

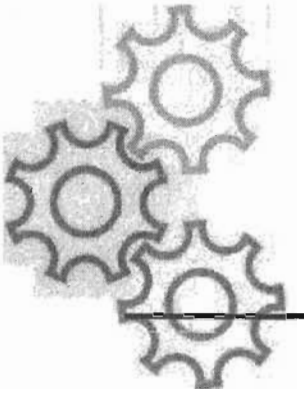
Para unir la aplicación cliente al servicio remoto, se utiliza un componente proxy de Java. Con el servidor WASP, este proxy se genera en tiempo de ejecución desde el archivo WSDL. Se necesita una interfase Java que pueda mantener una referencia a este objeto creado dinámicamente. Esta interface se puede hacer manualmente o de manera automática con el WSDLCompiler del servidor WASP. El único requerimiento es que los métodos de la interfase deben estar en un

subconjunto de los métodos de la clase Java de la lógica del *Web Service*. Primero se crea un objeto WebServieLookup que es utilizado para crear el proxy del Web Service, invocando el método lookup devuelve el proxy que se utiliza para invocar el Web Service. Este método requiere dos parámetros: una referencia al archivo WSDL y la clase de la interfase Java que hará referencia al proxy.

```
/*Clase que muestra las operaciones encontradas y disponibles en el proxy que se utiliza
para mostrar los servicios*/
public class OpClient {
   public static void main( String[] args ) throws Exception {
//Aquí es donde hace la búsqueda del servicio
    WebServiceLookup lookup =
(WebServiceLookup)Context.getInstance(Context.WEBSERVICE_LOOKUP);
    OpServiceProxy OperationService = (OpServiceProxy)lookup.lookup(
     "http://localhost:6060/OperstionService/",
     OpServiceProxy.class
    );
/*Aquí se despliegan la soperaciones disponibles*/
    System.out.println("Operaciones encontradas");
    System.out.println("----------");
    java.util.LinkedList list = OperationService.getAvailableOp();
    java.util.Iterator iter = list.iterator();
    while(iter.hasNext()) {
      System.out.println(iter.next());
    }
    System.out.println("");
    System.out.println("Obteniendo Prestamos");
    System.out.println("----------");
    System.out.println("PRESTAMOS: "+
OperationService.getOperation("PRESTAMOS"));
    System.out.println("");
   }
```

Para generar la interfase de java, se ejecuta el script runJavaClient.bat, el cual ejecuta a su vez el WSDLcompiler, lo compila y ejecuta la aplicación cliente, generando la salida de getAvailableOp y getOP.

En esta etapa de desarrollo se mostraron ejemplos de archivos críticos para el buen funcionamiento del componente, así como el desarrollo de las principales clases que lo componen.



5 Implementación

Objetivos

- Hacer la implementación del prototipo.
- Hacer pruebas de rendimiento.
- Comparar el componente contra el método actual

En este capítulo se pone en marcha el prototipo desarrollado y se somete a un proceso de evaluación. De igual manera, se hacen algunas comparaciones con el método utilizado actualmente para el proceso de operaciones de préstamos y créditos

5.1 Equipo

El prototipo y todo el software necesario para su funcionamiento se instalaron en un servidor Dell PowerEdge 2600 con las siguientes características:

- 2 procesadores Intel Xenon a 2.8 con Micro Arquitectura Hyper-Threading.9
- 1 Gb de memoria RAM
- Disco duro SCSI de 72 Gb
- Sistema Operativo Windows 2000 Advanced Server

Este servidor tiene una buena combinación de desempeño, disponibilidad y flexibilidad en la configuración, que lo hace ideal para el proyecto.

[9] La tecnología Hyper-Threading habilita el software multithread para ejecutar procesos en paralelo, así, las aplicaciones se ejecutan simultáneamente en el mismo procesador.

5.2 Base de datos

La base de datos que se utilizó para el prototipo fue MySql 4.1 para windows.

El servidor de base de datos MySql incluye todo lo necesario para desarrollar aplicaciones robustas de almacenamiento múltiple, algunas de sus principales características son:

- Soporte de syntaxis estándar SQL
- Multiplataforma
- Máquinas de almacenamiento independiente
- Transacciones
- Sistema de seguridad flexible, incluyendo soporte para SSL.
- Replicación
- Indexación y búsqueda
- Librerías incrustadas
- Soporte para Unicote
- Soporte para subconsultas

Inicialmente se cargaron los datos de las operaciones de los últimos 6 meses para hacer las pruebas. Estos datos que fueron cargados, son los generados por las aplicaciones de créditos y préstamos durante el periodo antes mencionado. Se llenaron los catálogos correspondientes y se crearon 5 usuarios para poder entrar al sistema.

La portabilidad de Java brinda la oportunidad de cambiar de Manejador de Base de Datos, si es que el usuario lo requiera o la implementación de otros sistemas. Para esto se diseñó una clase de java llamada DBManager, la cual se encarga de realizar la conexión a la base de datos, y maneja los elementos como el path de la base de datos, y algunos aspectos particulares de los manejadores, como el uso de comillas, entre otros.

De esta manera, se cuenta con la opción de implementar el sistema no sólo en cualquier plataforma, sino en cualquier base de datos, con tecnología JDBC u ODBC.

5.3 Puesta en marcha

Se hizo el despliegue del componente dentro del servidor de aplicaciones WASP 5.5, y el acceso a los servicios desde la aplicación cliente a través de una red local.

Se observó buen desempeño por parte del servidor.

El servidor de aplicaciones WASP de Systinet provee la arquitectura base, características de seguridad, características de administración, soporte para el esquema de XML, y lo más importante para nuestro proyecto, integración con Java.

Este servidor está enfocado en la interoperabilidad con líderes en frameworks de *Web Services*, incluyendo Microsoft .NET, Apache SOAP/Axis, BEA, y muchos otros. Este servidor también asegura compatibilidad a varios niveles:

- Interoperabilidad de transporte
- Interoperabilidad de datos
- Interoperabilidad de seguridad

Algunas de sus características más importantes son las siguientes:

- Brinda soporte a los últimos estándares
- Soporte completo para mensajes de confianza
- Soporte para persistencia
- Performance y escalabilidad
- Mensajes asíncronos
- Serialización flexible
- Soporte para referencias remotas de Java
- SOAP con Attachments

- Múltiples niveles de invocaciones del Web Service
- Soporte comprensivo para WSDL
- Desarrollo flexible
- Suite completa de herramientas de desarrollo
- Soporte para http proxy
- Internacionalización
- Mecanismo de autentificación por http
- Mecanismo de autentificación por SSL
- Mecanismo de autentificación por SPKM
- Políticas de autorización
- Encriptación XML
- Monitoreo
- Consola de invocación
- Independiente de plataforma
- Integración transparente con J2EE

5.4 Pruebas del prototipo

Se realizó una evaluación de los elementos principales que incluye el prototipo para detectar posibles errores, y no para demostrar el desempeño del componente.

Antes de ser presentado a los usuarios que realizarían las pruebas, se hicieron pruebas intensivas de rendimiento para garantizar la estabilidad del prototipo. Primero individuales para ver su comportamiento, y después integrales, para analizar su acoplamiento.

Entre las pruebas que se realizaron destacaron las siguientes:

- Se consultó las operaciones hechas de un determinado cliente en un periodo de tiempo dado.

- Se elaboró un reporte de pagos por vencer.
- Se emuló un envío de e-mail a los clientes con pagos próximos a vencer.
- Se formaron reportes seleccionando los campos y los criterios para formarlos, de esta manera se puede crear una gran cantidad de combinaciones de los campos que se desean ver, así surgen reportes que se ajustan a las necesidades específicas de todos los usuarios.

Se hizo una comparación del método utilizado actualmente para la elaboración de un reporte de operaciones diarias contra el mismo reporte en el nuevo prototipo, y el tiempo de respuesta del prototipo fue instantáneo y exacto, mientras que por el otro método, aparte de ser más lento el proceso, había que revisar si los datos arrojados eran correctos.

Los principales resultados observados dentro de la evaluación fueron:

- La información se mostró uniforme por lo cual ya no es necesario volver a revisar la información procesada.
- La información se obtiene desde el momento en que se procesa la operación, no es necesario esperar la respuesta de algún área de la institución para continuar con el proceso.
- Se recuperaron tiempos muertos en el área de operaciones.
- Se unificaron procesos.
- Las aplicaciones cliente tienen comunicación directa sin importar en que lenguaje fueron desarrolladas y bajo que plataforma.
- No es necesario contar con equipos de características especiales para su implementación, ya que no consume demasiados recursos.

Para garantizar el buen funcionamiento de las partes involucradas en el *Web Service*, se deben ejecutar procesos de una aplicación en específico, y estos procesos se deben ver reflejados en la otra aplicación participante. Y que se puedan ejecutar procesos comúnes desde ambas aplicaciones y obtener los mismos resultados, estos generados por el *Web Service*.

El evalúo del prototipo lo hizo un usuario del área de préstamos y uno del área de créditos para que pudieran constatar que la información que se genera, se almacena en ambas bases de datos, en la original de su aplicación, y en la nueva que está utilizando el componente *Web Service*, que en un futuro, esta base de datos será la principal.

Estos usuarios se hicieron cargo de la operación, y simulando un ambiente de producción se trabajó con él cerca de tres horas, haciendo recomendaciones o propuestas y correcciones no críticas. Por lo cual ambos dieron su respectiva aprobación.

Todas estas pruebas fueron satisfactorias, ya que si cumplieron con las expectativas, ejecutando las tareas que se solicitaron.

5.5 Documentos

La documentación entregada a los usuarios consta de las siguientes partes:

- Descripción de los alcances del prototipo; sus limitaciones y sus habilidades.
- Métodos y funciones con las que cuenta el prototipo y las que gradualmente se irán agregando a su funcionalidad.
- Requerimientos del prototipo.

Con estas pruebas, es posible hacer el análisis para comenzar con el desarrollo de todos los requerimientos de usuario opcionales e implementar el componente a otros departamentos dentro de la institución.

Conclusiones

A lo largo de mi experiencia profesional, me he dado cuenta que en la mayoría de las grandes empresas, al igual que la institución analizada en este trabajo, es muy latente el problema de contar con diferentes piezas de software, que cuando se desarrollaron, su principal objetivo era resolver problemas de lógica de negocio, sin darle la importancia que se merece a mantener una arquitectura que nos ofrezca interoperabilidad entre las aplicaciones. Arquitectura que deben tener todos los componentes de software dentro de una empresa, sin importar la distancia física de por medio. El reto ahora está en interrelacionar todos esos componentes para rendir todo su potencial y ponerlo a trabajar para aumentar los beneficios o reducir los costes.

Las mejoras mas notables con el desarrollo del *Web Service* desde mi punto de vista son:

- Respuestas más rápidas en el proceso de nuevas solicitudes de préstamos y de créditos.
- Unificación de procesos duplicados.
- Confiabilidad en los procesos.
- Recuperación de tiempos muertos en ambas áreas.
- Lo anterior representa una gran ventaja, ya que así es posible dar un mejor servicio al cliente y con ello aumentar las utilidades de la institución.

Como se pudo observar, la solución más viable es la implantación de los *Web Services*, ya que una de sus principales características es su integración con sistemas existentes, esto es, reutilización de la infraestructura.

Las principales razones por las que se utilizaron Web Services son:

- Interoperabilidad Dinámica. Nuevas relaciones entre participantes pueden ser construidas dinámicamente y automáticamente ya que los Servicios Web aseguran una interoperabilidad completa entre sistemas.
- Accesibilidad. Los Servicios son completamente descentralizados y distribuidos sobre Internet y accedidos a través de una gran variedad de dispositivos.
- Eficiencia. Las empresas pueden liberarse de la tarea de desarrollar software propio, lo cual es lento, caro y complejo, y enfocarse en el valor agregado y los servicios específicos. El desarrollo incremental es natural y sencillo y ya que los *Web Services* se implementan y declaran en una forma humanamente entendible, y el seguimiento de los errores es más sencillo. En consecuencia se reducen los riesgos y los costos de mantenimiento asociados.
- Especificaciones Universalmente Aceptadas. Los Servicios Web se basan en especificaciones estándar para el intercambio de datos, mensajería, búsqueda, descripción de la interfase y coordinación de los procesos.
- Integración de plataformas y sistemas heterogéneos.
- Ahorran tiempo y dinero, ya que reducen la duración del ciclo de creación, más si la lógica del negocio ya existe.

Como se ha podido ver a lo largo de este trabajo, los *Web Services* no proporcionan soluciones a todos los problemas, pero sí a muchos de ellos. Uno de los principales, es el bajo costo que conlleva la implementación de los *Web Services* dentro de las empresas de todo tamaño, el lado que podría ser más costoso es la curva de aprendizaje que conlleva crear una arquitectura que haga un uso eficaz de esta tecnología, pero se recompensa con una inversión pequeña en gastos de operación y mantenimiento.

Se hace la recomendación de utilizar esta tecnología siempre que nos encontremos por lo menos con uno de las siguientes necesidades.

- Crear una aplicación accesible desde internet y desde cualquier plataforma.
- Hacer pública una aplicación sin saber desde que plataforma se va a acceder.
- Que los clientes sean las aplicaciones, no los usuarios.

Convendría ahora investigar cómo se puede implantar un buen nivel de seguridad dentro del prototipo que se desarrolló.

Por otro lado, el desarrollo del cliente para el prototipo como una página JSP (Java Server Pages) se facilitó con el uso de "tag libs". Sin embargo, se presentó la desventaja que no hay una separación entre contenido y presentación, por tanto, el mantenimiento se dificulta.

Finalmente quiero enfatizar mi orgullo por los conocimientos que adquirí dentro de la carrera de Matemáticas Aplicadas y Computación, porque aunque no nos enseñan con una laptop a la mano como en otras universidades, si tenemos las bases fundamentales no solo dentro del área matemática, sino también dentro del área informática, y con la combinación de ambas, aunque no tengamos el conocimiento de cómo hacer determinada tarea dentro de nuestra especialidad, es suficiente tener una ligera noción de cómo se puede resolver para que logremos realizarla, ya que nos enseñaron a hacer uso efectivo de la cultura autodidacta.

Glosario

Acoplamiento

Posibilidad que tiene un servicio de funcionar de forma autónoma. Se dice que un servicio o aplicación es bajamente acoplado cuando puede funcionar de forma independiente respecto de los demás servicios con los que interacciona.

Actor

Una persona u organización que puede ser propietario de agentes que igualmente buscan, usan o proveen Web Services. Un concepto físico o conceptual de una entidad que puede realizar acciones. Ejemplos: personas, compañías, máquinas, software en ejecución. Un actor puede tomar formas de uno o más roles. Un actor en un nivel de abstracción puede ser visto como un rol en un nivel más bajo de abstracción.

ASP

[Active Server Pages]. Sistema de programación desarrollado y propiedad de Microsoft para realizar las tareas antes realizadas por los CGI.

API

Aplication Program Interface. Herramientas que provee un fabricante de software para facilitar el uso de sus programas desde otros programas externos.

Applet

Pequeño programa en lenguaje de programación Java integrado en una página Web.

Arquitectura

Conjunto de las características, disposiciones e interconexiones de los principales componentes de un sistema.

CGI (Interfaz de gateway común)

Interfaz para programadores que crean archivos de comandos o aplicaciones que se ejecutan internamente en un servidor de Web. Estos archivos de comandos pueden generar texto y otros tipos de datos de forma inmediata, en respuesta a una entrada del usuario, o bien tomando la información de una base de datos.

Componente

Un componente es una pieza de software que puede interactuar con otros componentes, encapsulando cierta funcionalidad o conjunto de funcionalidades. Un componente tiene una interfaz claramente definida y se adapta a un comportamiento común preescrito para todos los componentes dentro de la arquitectura. Es una unidad abstracta de software que provee transformaciones de datos por medio de su interfaz

Contexto

Un contexto especifica un patrón (o camino) de acceso: un conjunto de interfaces que nos ofrecen un medio de interaccionar con un modelo. Por ejemplo, imaginemos un modelo con arcos de diferentes colores que conectan nodos de datos. Un contexto podría ser una hoja de acetato coloreada que se coloca sobre el modelo, permitiéndonos obtener una vista parcial de la información total contenida en el modelo.

CORBA

(Common Object Request Broker Architecture). Arquitectura que posibilita el intercambio de objetos entre aplicaciones. Un objeto CORBA puede estar desarrollado en cualquier lenguaje y correr en cualquier sistema operativo.

CSS

(Cascading Style Sheets) Método que permite definir por separado las reglas para definir las características de los elementos HTML, DHTML y XML. Sus versiones se distinguen por el número de edición: CSS1, CSS2, etc

DCOM

(Distributed Component ObjectModel). Extensión del modelo COM que posibilita el intercambio de objetos entre aplicaciones, conceptualmente similar a CORBA pero basado exclusivamente a entornos Windows.

Dirección URL (Uniform Resource Locator)

Formato de las direcciones de sitios que muestra el nombre del servidor en el que se almacenan los archivos del sitio, la ruta de acceso al directorio del archivo y su nombre.

Descubrimiento

El acto de localizar un proceso en una máquina que cumple ciertos criterios de funcionalidad, el objetivo es localizar un recurso apropiado disponible en un *Web Service*

DHTML

(Dynamic html) Combinación de html, hojas de estilo y Javascript que permiten modificaciones automáticas en los elementos de las páginas.

DOM

(Document Object Model) Es una interfase independiente de la plataforma y del lenguaje, que permite que los programas y scripts tengan acceso dinámicamente y actualicen el contenido, la estructura y estilo de los documentos.

Documento bien formado (well-formed document)

Un documento está bien formado si es válido a nivel de etiquetas y las entidades están limitadas a elementos simples (es decir, un solo sub-árbol).

DTD

Definición de Tipo Documento. Una parte opcional del prólogo del documento, según lo especifica el estándar XML. El DTD especifica restricciones de etiquetas válidas y secuencias de etiquetas que pueden estar en el documento.

Entidad

Un elemento individual que puede incluirse en un documento XML. Como una referencia de entidad puede nombrar una entidad tan pequeña como un caracter (por ejemplo, "<", que representa el símbolo de menor que (<). Una referencia de entidad también puede referenciar un documento completo, o una entidad externa, o una colección de definiciones DTD

Elemento

Todo documento contiene uno o más elementos, cuyos límites están limitados por etiquetas iniciales y etiquetas finales, o bien, en el caso de documentos vacíos, por una etiqueta de elemento vacío. Cada elemento tiene un tipo, identificado por un nombre, y puede tener un conjunto de atributos. Cada atributo tiene un nombre y un valor.

Gateway

Conversor de protocolos. Nodo específico de la aplicación que conecta redes que de otra forma serían incompatibles. Convierte códigos de datos y protocolos de transmisión que permiten la interoperatividad.

HTML

El Lenguaje para el Formato de Documentos de Hipertexto (HyperText Markup Language) es un lenguaje simple de formato de documentos usado para crear documentos de hipertexto portables de una plataforma a otra. Los documentos HTML son documentos SGML con una semántica genérica que es apropiada para representar información de un amplio rango de aplicaciones.

HTTP

Protocolo de transferencia de hipertexto. Es el protocolo estándar de Internet que permite la transferencia de páginas HTML.

Inconsistencia

Es cuando se tiene la misma información en varios sistemas, pero además de forma diferente. Por ejemplo, tener repetido el nombre asociado a un DNI, pero además tenerlo con alguna variación, de forma que no coincidan.

Interfaz

Aspecto que ofrece un programa al exterior, bien a la persona que lo utiliza como usuario, o bien a otros programas que lo utilizan sin intervención humana.

Interfaz (interface)

Una interfaz es una declaración de un conjunto de métodos sin información sobre su implementación. En los sistemas de objetos que soportan interfaces y herencia, las interfaces normalmente puede derivar unas de otras.

Interoperable

Que admite posibilidades de conexión con otros sistemas.

J2EE. Java 2 Enterprise Edition. Plataforma tecnológica basada en la tecnología Java.

LAN

Local Area Network. Red de área local.

Middleware

Servicios o programas que hacen de intermediario entre dos sistemas, de forma que ambos pueden comunicarse y compartir información a través de este middleware. Tienen la capacidad de poder entenderse con ambos sistemas.

Método

Un método es una operación o función que está asociada a un objeto y que tiene permiso para manipular los datos del objeto.

modelo

Un modelo es la representación real de los datos obtenidos a partir de la información disponible. Ejemplos son el modelo de estructura y el modelo de estilo que representan la estructura analítica y la información de estilo asociada a un documento. El modelo podría ser un árbol, o un grafo orientado, o cualquier otra cosa.

Persistencia.

Característica de un servicio que permite guardar en el disco (es decir, de forma permanente) el estado de ese servicio en un momento dado.

Protocolo

Un conjunto de reglas para describir como transmitir datos, especialmente a través de una red.

Proxy

Servidor especial encargado, entre otras cosas, de centralizar el tráfico entre Internet y una red privada, de forma que evita que cada una de las máquinas de la red interior tenga que disponer necesariamente de una conexión directa a la red

RPC

Remote Procedure Call. Protocolo para llamadas a procedimientos remotos.

Servidores

Máquinas especializadas en proveer servicios específicos, por ejemplo bases de datos, web, correo o servicios de red.

Servlet.

Pequeño programa que se ejecuta dentro del entorno de un Servidor Web, en el servidor.

SGML

Standard Generalized Markup Language. Lenguaje para la descripción de otros lenguajes de documentos estructurales basados en etiquetas. Por ejemplo, el HTML está definido mediante el SGML.

SOAP

Protocolo ligero basado en XML, para el intercambio de información en un ambiente descentralizado y distribuido.

State

Conjunto de atributos que representan las propiedades de un componente en determinado momento.

SSL

Nivel de socket de seguridad. Protocolo que utiliza Netscape para proporcionar transacciones seguras a través de la red.

TCP/IP

(Transmission Control Protocol / Internet protocol). Sistema de protocolos, definidos en RFC 793, en los que se basa buena parte de Internet. El primero se encarga de dividir la información en paquetes en origen, para luego recomponerla en destino, mientras que el segundo se responsabiliza de dirigirla adecuadamente a través de a red.

UDDI

Universal Description, Discovery and Integration. Protocolo que permite descubrir *Web Services* existentes en Internet y conocer su descripción para saber qué tarea realizan.

URL

"Universal Resource Locator". Un puntero a una localización específica (dirección) en la Web que es única en todo el mundo. La primera parte de la URL define el tipo de dirección. Por ejemplo http:/ identifica una localización Web. El prefijo ftp:/ especifica un fichero descargable. Otros prefijos incluyen file:/ (un fichero del sistema local) y mailto:/ (una dirección de email).

W3c

World Wide Web Consortium. Organismo internacional que gobierna los estándares de Internet.

Web Services

Componentes de software, independientes de plataforma, que permite a las aplicaciones compartir datos vía internet, fácilmente publicadas, localizadas e invocadas mediante protocolos Web estándar, como XML, SOAP, UDDI o WSDL.

WSDL

Web Services Description Languaje. Describe la manera adecuada de utilizar un *Web Service* desde otros programas.

XHTML

Un XML con aspecto de HTML definido por varios DTDs XHTML. Para usar XHTML en todo, iría en contra del propósito de XML, ya que la idea de XML es identificar los contenidos de información, no sólo decir como mostrarlos.

XML

El Lenguaje Extensible para el Formato de Documentos (Extensible Markup Language) es un dialecto extremadamente simple de SGML. Su objetivo es hacer posible servir, recibir y procesar SGML genérico en la Web en el modo en que hoy es posible con el HTML. El XML se ha diseñado para ser fácil de implementar y para su interoperatividad con SGML y HTML.

XSL

(Extensible Stylesheet Language), es un documento que define como deben ser transformados elementos en XML.

Bibliografía

Referencias bibliográficas

Anne Thomas Manes, ***Web Services: A Manager's Guide***, Addison-Wesley Pub Co, 2003

Douglas K. Barry, ***Web Services and Service-Oriented Architectures: The Savvy Manager's Guide***, Morgan Kaufmann, 2003.

Eric Newcomer, ***Understanding Web Services: XML, WSDL, SOAP, and UDDI***, Addison-Wesley Pub Co, 2002.

Ethan Cerami, ***Web Services Essentials***, O'Reilly & Associates, 2002

Frank P. Coyle, XML, ***Web Services, and the Data Revolution***, Addison-Wesley Pub Co, 2002.

Kim Topley, ***Java Web Services in a Nutshell***, O'Reilly & Associates, 2003.

Ramesh Nagappan, Robert Skoczylas, Rima Patel Srig, ***Developing Java Web Services: Architecting and Developing Secure Web Services***, John Wiley & Sons, 2003.

Ray Lai, ***J2EE Platform Web Services, Prentice Hall PTR***, Agosto 2003

Referencias Web

Java Technology and Web Services, java.sun.com/webservices, 2004

Unified Modeling Language (UML), www.omg.org/uml, 2005

W3c, www.w3.org/2002/ws, http://www.w3.org/TR/SOAP, 2004

Web Services Apache Project, ws.apache.org, 2005

Web Services Architecht, www.webservicesarchitect.com, 2005

Web Services ORG, www.webservices.org, 2005

Web Services XML, webservices.xml.com, 2005

XML, xml.org, 2004

XML-RPC, www.xml-rpc.com, http://www.xmlrpc.com/, http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm , 2005