Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Cáceres Alvarez, Luis Marco; Pinto Bernabé, Miguel Alejandro
Modelo de programación asíncrona para Web transaccionales en un ambiente distribuido
Ingeniare. Revista Chilena de Ingeniería, vol. 19, núm. 1, junio, 2011, pp. 26-39
Universidad de Tarapacá
Arica, Chile

Disponible en: http://www.redalyc.org/articulo.oa?id=77219386004

- Cómo citar el artículo
- Número completo
- Más información del artículo
- Página de la revista en redalyc.org

Sistema de Información Científica
Red de Revistas Científicas de América Latina, el Caribe, España y Portugal
Proyecto académico sin fines de lucro, desarrollado bajo la iniciativa de acceso abierto

Ingeniare. Revista chilena de ingeniería, vol. 19 Nº 1, 2011, pp. 26-39

Modelo de programación asíncrona para Web transaccionales en un ambiente distribuido

Asynchronous programming model for transactional Web in a distributed environment

Luis Marco Cáceres Alvarez[1] Miguel Alejandro Pinto Bernabé[1]

Recibido 4 de enero de 2010, aceptado 25 de noviembre de 2010
Received: January 4, 2010 Accepted: November 25, 2010

RESUMEN

El presente trabajo define y detalla un modelo de programación asíncrono para sistemas de información Web transaccionales orientado a servicios en un ambiente distribuido uniendo las ventajas de las técnicas de programación Web asíncronas (AJAX), patrones de diseño orientados a objetos y servicios Web, para la obtención de aplicativos caracterizados por ser tolerantes a fallos, distribuidos, eficientes y usables. Principalmente se puntualizan los problemas encontrados en el modelo para uso de servicios Web clásicos, por ende se define, documenta y desarrolla un modelo de programación que solucione y mejore los servicios Web clásicos y se valide la solución a través del desarrollo de un prototipo basado en el modelo de programación definido.

Palabras clave: Programación asíncrono, Web transaccionales, Web clásicos, servicios distribuidos, AJAX.

ABSTRACT

The present work defines and details an asynchronous programming model for transactional web information systems in distributed environment, joining the advantages of web asynchronous programming techniques (AJAX), object oriented design patterns and web services, to obtain fail tolerant, distributed, efficient and usable applications. The problems found in the classical Web services model are pointed out and a new model to solve and improve most common web services is defined, documented and developed. Also, a prototype is developed to validate the given solution using the defined model.

Keywords: Asynchronous programming, transactional webs, classic web, distributed services, AJAX.

INTRODUCCIÓN

En la actualidad el desarrollo de sistemas de información en Plataformas *Web* no sólo es un hecho, sino que está reemplazando las tradicionales aplicaciones *Desktop*[2]. En cierta forma todo tiende a la *Web* a que nuestras aplicaciones puedan ser accedidas desde cualquier lugar del mundo, sin necesidad de instalar *software*, sin limitaciones en la usabilidad y sin que nuestro *hardware* tenga que sufrir cambios por requerimientos del sistema en sí.

De ese modo, se ha llegado a que las páginas *Web* tengan lugar donde sólo se habla de un medio netamente hipertextual (para ver páginas estáticas y nada más) y que tengan que evolucionar con lenguajes orientados al manejo de transacciones en servidores que trabajen con protocolos Web generando código extensible (XHTML-*eXtensible Hypertext Markup Language*) por los actuales *browsers*[3]. Pero esto no fue suficiente ya que desde el punto de vista del usuario, en una aplicación *Web* se tendría que llenar todo el formulario para mandar información y recibir nuevamente una página completa, donde

1 Escuela Universitaria de Ingeniería Industrial, Informática y Sistemas. Universidad de Tarapacá. 18 de Septiembre 2222. Arica, Chile. E-mail: lcaceres@uta.cl; mapintob@uta.cl

2 Sistema estacionario que necesita una instalación física en el computador y se ejecuta locamente.

3 Navegadores Web.

además del formulario ya visto, sólo agrega la confirmación de la transacción.

Esto se debe principalmente, porque los sistemas de información *Web* se han adecuado a un medio netamente hipertextual donde el *browser* (*Internet Explorer*, *Mozilla FireFox*, *Opera*, etc.) no se basa en enviar información necesaria y recibir sólo la confirmación de nuestra transacción en un mensaje (éxito o errado), sino que pide que se ejecute un archivo (ASP – *Active Server Pages*, PHP – *Hypertext Preprocessor*, JSP – *Java Server Pages*) y debe recibir una página completa como respuesta.

Para solucionar este problema se creó una técnica de programación asíncrona que permite enviar y recibir únicamente la información necesaria en *background*[4] sin la necesidad de hacer *postback*[5] a toda la pagina al realizar una transacción con el servidor, esta técnica es conocida como AJAX (*Asynchronous JavaScrit and XML*). Con esta técnica las aplicaciones *Web* cumplen con la ventaja de una programación basada en eventos y que transfiere sólo la información necesaria en las transacciones, aprovechando mucho mejor el ancho de banda de la red utilizada [1].

Hace mucho tiempo se introdujo el concepto de programación distribuida, en pocas palabras la ejecución remota de procedimientos haciendo uso de la red y si hablamos de este concepto aplicado a la *Web* por excelencia, es que nos referimos a *Web Services*. Los servicios Web cada vez toman más fuerza y hasta existen arquitecturas completas de programación orientadas a servicios, ya que son fáciles de utilizar, usan protocolos estándares sobre http, como SOAP (*Simple Object Access Protocol*) y seguros, es más, muchas entidades publican sus propios servicios y pueden cobrar por cada utilización del mismo.

Los últimos dos temas expuestos, *AJAX* y *Web Services,* son la propuesta del presente proyecto de investigación, ya que el objetivo es el de desarrollar un modelo de programación de sistemas de información Web transaccionales distribuido, donde el cliente Web (*browser*) sin la necesidad de utilizar un servidor pueda usar *Web Services* expuestos en cualquier servidor utilizando *AJAX*. De esa forma el sistema es totalmente independiente de un servidor específico, ya que él mismo ejecuta y procesa las respuestas de los servidores logrando de ésta forma tener dividida la lógica de negocios de nuestra aplicación en *N* servidores (si deseamos replicados), programada con arquitecturas de desarrollo totalmente distintas, en lenguajes diferentes, desarrollada por proveedores diferentes y aún así contar con características de una programación basada en eventos y una interfaz de usuario transparente, ya que el usuario no es expuesto a esperas de respuesta por parte del servidor con una pantalla en blanco frente a él, además, que se logra la independencia entre módulos del sistema pudiendo avanzar éste en paralelo (construcción del sistema) y ser tolerante a fallas por el mismo hecho que se encuentra distribuido.

SERVICIOS WEB

Todas las empresas utilizan procesos internos para gestionar sus flujos de información y procesos externos de negocio a fin de interactuar con sus socios, clientes y empleados en la cadena de valor. Ambos procesos pueden ser gestionados con servicios Web para reducir costos y potenciar la agilidad del negocio.

Los servicios Web proporcionan un conjunto de estándares que permiten que las compañías se comuniquen entre sí, tanto interna como externamente. Esto lo logran sin necesidad de modificar significativamente sus programas actuales de software, gracias a que los servicios Web usan XML como idioma común en sus comunicaciones [2]. Se definen dos tipos de servicios Web en [3] que citamos a continuación:

- ***Servicios Web Internos***: pertenecen a la empresa que los utiliza y pueden servir para conectar distintos departamentos de la empresa, como Ventas, Recursos Humanos, Finanza y Producción. Por ejemplo, una aplicación financiera puede llamar en tiempo real a un servicio Web que convierte euros en dólares o el Departamento de Ventas puede llamar a otro para consultar determinada información desde la base de datos de clientes.
- ***Servicios Web Externos***: permiten a las empresas intercambiar servicios a través de Internet. Una

[4] Término usado para expresar que la información se envía de forma transparente sin que el usuario lo perciba.

[5] Término acuñado por Microsoft para expresar envío y recepción de información con el servidor Web.

empresa interesada en un servicio determinado puede acudir a un directorio de servicios Web como UDDI (*Universal Description, Discovery and Integration*) para buscarlo. Ejemplo de estos registros públicos es www.xmethods.net donde se pueden encontrar cientos de servicios Web gratuitos.

Los servicios Web Internos serán utilizados en un primer momento por grandes empresas que ya disponen de infraestructura Web y pueden ganar tiempo y dinero automatizando procesos dentro de la empresa, mientras que las empresas de menor tamaño pueden no encontrar a corto plazo ventajas al no disponer de un entorno de sistemas complejo. Por otro lado, aunque la empresa no desarrolle sus propios servicios Web, puede utilizar los servicios de información de empresas externas, como los de información meteorológica o cotización en Bolsa.

El propósito de los servicios Web es el poder utilizar estos servicios por cualquier compañía tranquilamente desde su ordenador con acceso a Internet, e incluso poder incorporar los servicios Web dentro de las aplicaciones informáticas propias, como si fueran un servicio más [4].

CONTEXTO ACTUAL DEL MODELO DE SERVICIOS WEB

Actualmente en el contexto de la forma de utilizar un servicio Web, es mediante una petición HTTP desde un cliente (*browser)* a un *script* en el "Servidor Web" y desde éste se consume el servicio Web que se encuentra en otro servidor llamado "Servidor de Servicios", se recibe la respuesta y se devuelve una página completa al cliente (*browser)*. Podemos apreciar este flujo en el siguiente diagrama de la Figura 1.

Según este modelo de trabajo en un contexto de servicios Web "tradicional", podemos plantear diferentes problemas, que se listan a continuación:

- Toda la aplicación depende del "Servidor Web", no importando lo distribuida que se encuentre la lógica de negocios en los "Servidores de Servicios". Si se llega a caer el "Servidor Web" no funciona el sistema.
- En el supuesto que el "Cliente 1" sólo esté trabajando con los servicios en el "Servidor de Servicios 1" y cae el "Servidor Web", no podrá seguir trabajando.
- Los mensajes, respuestas y peticiones transitan a través de una gran distribución geográfica, que va desde el cliente al "Servidor Web", hasta el "Servidor de Servicios" y de regreso al cliente.
- Las respuestas del "Servidor Web" siempre son páginas completas no importando si sólo se necesita una confirmación de una transacción, lo cual genera lentitud en transporte de las respuestas y mayor tráfico de información.
- Si deseamos que los clientes 1, 2 reaccionen de manera diferente a una transacción, tendría que

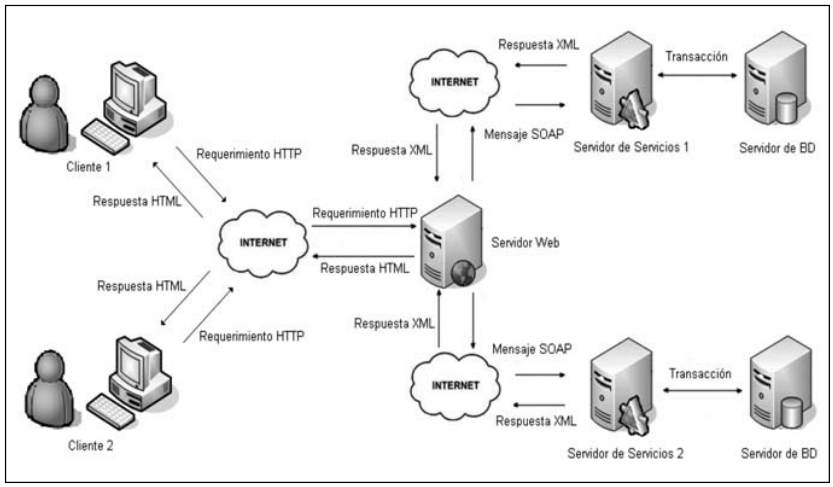

Figura 1. Diagrama de uso de servicios Web clásicos.

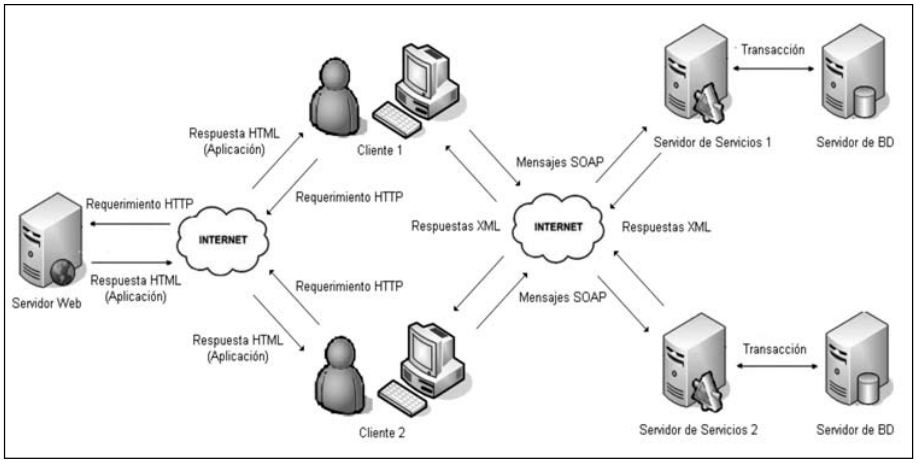

Figura 2. Diagrama del modelo propuesto para uso de servicios Web.

implementarse otro "Servidor Web", uno por cada tipo de aplicación o por lo menos poner otra aplicación diferente en el "Servidor Web". O realizando la personalización oportuna en el lado del cliente, precisamente como se propone con el uso de AJAX.

- Cada vez que hacemos uso de un servicio o queremos acceder a la capa de negocios, el usuario percibirá una página en blanco deteniendo su trabajo, lo cual dificulta la usabilidad del sistema. Lo que condiciona el acceso al servicio a un tiempo de respuesta no deseable por el usuario.
- Siempre que se quiera realizar una transacción, debemos de realizar el *submit*[6] de toda la página y no podemos concentrarnos en una programación más limpia orientada a eventos como en los sistemas *desktop*.

Debido a estos problemas anteriormente mencionados, es que a continuación se describe una solución a este modelo clásico.

SOLUCIÓN PROPUESTA AL MODELO CLÁSICO

Se propone un modelo de programación asíncrono para sistemas Web transaccionales orientados a servicios, en este modelo podemos apreciar que clientes que quieran trabajar con el sistema, primero lo solicitan a un "Servidor Web", como un requerimiento de página común y corriente, este último responde no sólo con la página requerida sino con la aplicación completa (XHTML, CSS*: Cascading Style Sheets* y Javascript), al recibir toda esta información el cliente se vuelve independiente del "Servidor Web", ya que ahora tendrá la capacidad de usar y procesar respuestas de los servicios en los "Servidores de Servicios" por sí mismo y sin necesidad del "Servidor Web". Esto se puede apreciar en el siguiente diagrama de la Figura 2.

Según este modelo de trabajo, podemos resolver los problemas anteriormente mencionados de la siguiente manera:

- El cliente ya no depende del "Servidor Web" todo el tiempo, ya que sólo al inicio de su sesión recibe la aplicación y luego por sí mismo puede acceder a los servicios Web, si se cae el "Servidor Web" no afectará al cliente.
- En el supuesto que el "Cliente 1" sólo esté trabajando con los servicios en el "Servidor de Servicios 1" y cae el "Servidor Web" podrá seguir trabajando sin problemas.
- Los mensajes, respuestas y peticiones ya no viajan un gran espacio debido a que ya no pasamos por un "Servidor Web" para acceder a los servicios, sino que accede directamente a los "Servidores de Servicios" desde el cliente.

[6] Término usado cuando se envía información de un formulario Web al servidor.

- Las respuestas entre el "Servidor de Servicios" y el cliente son mensajes XML que contienen sólo la información necesaria de la transacción, al contrario que con un "Servidor Web" que devuelve páginas completas; de esta nueva forma, sólo existirá el tráfico de información necesario.
- Como los clientes manejan la respuesta XML de los "Servidores de Servicios" localmente en el *browser* son independientes entre ellos y pueden trabajar o comportarse de formas diferentes. Aunque bajo un modelo "tradicional" de acceso se pueden obtener las mismas condiciones de independencia, pero con los problemas ya descritos en la sección anterior.
- Por usarse una programación asíncrona AJAX, no percibimos páginas en blanco, ya que el uso del servicio se hace en *background* sin interrumpir el trabajo del usuario.
- Podemos programar orientado a eventos ya que no dependemos del *submit* de toda la página para usar un servicio; esto hace nuestra aplicación similar a los sistemas de información *desktop*.

Una vez descrita la propuesta de trabajo, describiremos algunos trabajos relacionados con respecto al tema, estos son los siguientes:

AJAX Extensions de Microsoft (ATLAS)
Es una extensión del visual estudio, básicamente un conjunto de librerías DLL y Asemblies, que permiten rápidamente crear páginas con las características de AJAX utilizando los ya conocidos controles y elementos de ASP.NET 2.0.

Se caracteriza por ser una técnica de desarrollo Web con componentes conformados por librerías script tanto en el cliente como en el servidor, usando para el cliente tecnologías como JavaScript y DHTML [5].

Si bien esta herramienta es muy útil para el desarrollo de aplicaciones con AJAX, con respecto al presente proyecto se han encontrado principalmente los siguientes inconvenientes:

- Puede acceder a servicios Web principalmente con fines de autenticación y validación de perfiles.
- No provee de métodos o librerías para la gestión de las respuestas de los servicios Web en caso de que no se dé validación.
- Si aún así deseamos utilizar la extensión de AJAX, debemos someternos a altísimos costos en licencias para las herramientas de Microsoft, así como vernos restringidos a utilizar sus herramientas para el desarrollo de nuestras aplicaciones.

JavaScript SOAP Client
Es un proyecto que consta de una librería de un solo archivo Javascript, el cual permite la ejecución de servicios Web por SOAP y está basado en un desarrollo orientado netamente en el cliente usando XmlHttpRequest y Javascript; es totalmente independiente del lenguaje de programación de servidor (ASP, PHP, JSP) ya que funciona en el cliente [6].

Esta librería es muy útil al momento de llamar y recibir respuesta del servicio Web pero no tiene métodos para la gestión de la página actual o para procesar la respuesta y mostrarla al usuario.

En el presente proyecto se utilizará esta librería como base para el desarrollo en la comunicación del *browser* con los servicios Web.

JavaScript SOAP Client library
Es una librería que permite la comunicación con servicios Web mediante SOAP basada en un desarrollo íntegramente orientado al cliente porque está escrita en JavaScript por lo que es totalmente independiente del lenguaje de servidor; esta librería provee tres clases *core* (SOAPClient, SOAPRequest y SOAPObject.) [7].

También es muy útil al llamar y recibir respuesta del servicio Web pero no tiene métodos para poder procesar la respuesta y mostrarla al usuario.

Se utilizará como base de forma de trabajo de esta librería para el desarrollo de la biblioteca de clase AJAX SOAP.

En este contexto, es que se presenta a continuación la definición y propuesta del modelo de proyección.

DEFINICIÓN Y ESPECIFICACIÓN DEL MODELO

Diagrama de despliegue
Para el presente modelo de programación del proyecto de investigación se han podido definir dos formas de despliegue o uso de la misma.

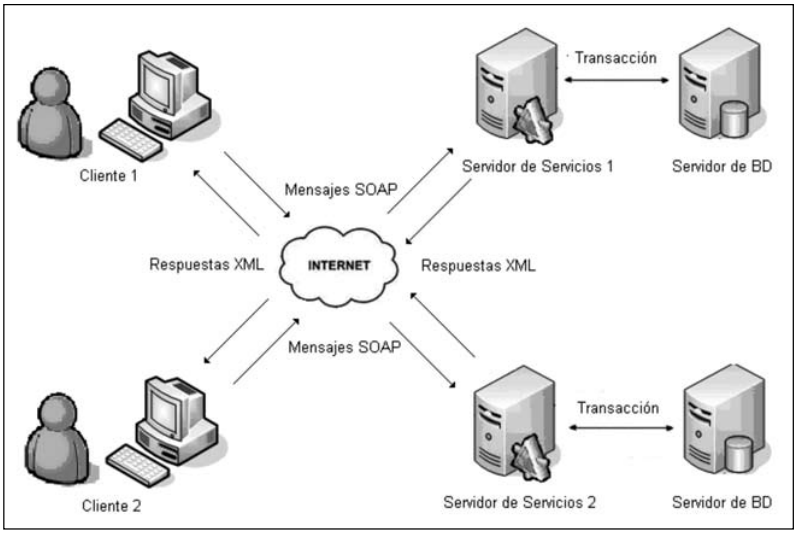

Figura 3. Diagrama de Despliegue Offline.

El primero contempla principalmente una característica *Online* debido a que depende parcialmente de un servidor Web para poder recibir las diferentes páginas que conforman el sistema de información que se desarrolle, el servidor Web se encarga de entregar páginas o módulos del sistema solicitados mediante requerimientos HTTP por el aplicativo cliente, cuando la información ya se encuentra en el cliente éste puede usar directamente los servicios publicados en los servidores que realizan las transacciones necesarias de negocios, según el requerimiento y devuelven las respuestas de las mismas a los clientes mediante documentos XML que son procesados por éste para regenerar únicamente partes del formulario actual. Podemos apreciar el despliegue de este tipo de aplicación en la Figura 2.

El segundo contempla una característica *Offline* debido a que no depende ni necesita de un servidor Web para poder funcionar, esto se debe a que la aplicación cliente ya se encuentra en los computadores respectivos donde correrá el sistema y el sistema al encontrarse en el cliente ya está en capacidad de usar directamente los diferentes servicios publicados y cumplir con el ciclo de procesamiento mencionado en el párrafo anterior. Podemos apreciar el despliegue de este tipo de aplicación en la Figura 3.

Diagrama de componentes

En el modelo de programación se aprecian principalmente dos tipos de nodos los cuales se detallan a continuación:

- ***Browser* del Cliente:** El nodo de *browser* del cliente es el más importante ya que es el que contiene la lógica, clases y herramientas necesarias para comunicarse con los servidores de servicios. Dentro de estas clases, librerías y objetos que se utilizan para lograr la gestión de la página del cliente, el uso de los servicios así como la gestión de las respuestas XML de los mismos están estructuradas siguiendo el patrón de diseño *Modelo Vista Controlador*[7] y lógicamente agrupados en tres paquetes:
  - MODELO (*Js Soap Access* y *Js Object Model*): Aquí se concentran las clases tanto para el uso de los servicios Web como las clases que nos permitirán modificar elementos de la página del cliente, tales como tablas, combos, textos, etc.
  - CONTROLADOR (*Js Page Logic*): La lógica de la página son archivos de *JavaScript* (uno por cada página) que son los encargados de gestionar los eventos disparados por la página del cliente y de acuerdo al evento

[7] Patrón de diseño arquitectónico o estructural utilizado en un inicio por programadores de SmallTalk, pero que por su gran flexibilidad y orden resulto idóneo para su utilización en servidores Web.

disparado, éstos acceden al modelo para usar servicios y dependiendo de las respuestas obtenidas, poder regenerar la página usando también las clases del modelo destinadas para esta labor.

- o VISTA (XHTML Y CSS): La vista consta de los archivos XHTML, los cuales definen la estructura de las páginas en el sistema y los archivos CSS que definen el diseño (color, alineación y forma).

- **Servidor de servicios:** Los servidores de servicios son computadores remotos que exponen métodos, éstos son totalmente independientes del modelo de programación propuesto, pudiendo ser escritos tanto en C#, PHP o Java y estructurar su lógica de negocio como la gestión de sus transacciones siguiendo cualquier arquitectura de programación, ya que lo importante es que tengan un archivo WSDL que define la forma de uso del servicio y devuelvan archivos XML los cuales pueden ser interpretados por el *browser.*

Podemos apreciar los diferentes nodos explicados, así como sus paquetes y componentes en la Figura 4.

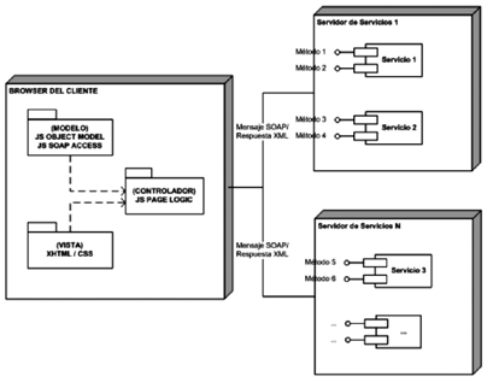

Figura 4. Diagnóstico de componentes del modelo propuesto.

Diagrama de clases

El diagrama de clases detalla los atributos, operaciones y relaciones entre clases existente en el modelo de programación propuesto, pero antes veremos una explicación de las funciones definidas para cada una de ellas:

- ***XmlHttpRequest***: Esta clase está inmersa en los *browsers,* es creada de diferente forma dependiendo si se está trabajando con Internet Explorer, Mozilla Firefox, Opera, etc. Es la encargada de realizar peticiones en *background* a servidores, puede ser cualquier archivo publicado y se accede mediante URL (*Uniform Resource Locator*).
- ***SOAPClientParameters***: Los objetos que son creados de esta clase sirven para adjuntar los parámetros que se enviarán a un método de un servicio, principalmente los objetos de esta clase contienen una colección de objetos donde se van juntando los parámetros para luego ser serializados en forma de cadena XML según lo defina el archivo WSDL del servicio a usar.
- ***SOAPClient***: Es la clase que permite realizar uso del servicio, utilizando las dos clases anteriores, trae el documento WSDL del servicio, si es la primera vez que lo va utilizar, y lo almacena en una caché para futuras transacciones, después pide la serialización de los parámetros en forma de cadena XML respetando las reglas definidas por el WSDL realizando uso del servicio y cuando regresa la respuesta del mismo en archivo XML lo convierte a variables de *JavaScript* (valores escalares, *arrays*, cadenas, objetos y colecciones) y lo retorna.
- ***SOAPClient_CacheWsdl***: Es una colección de objetos del tipo *XML Document* la cual sirve como caché para almacenar los diferentes archivos WSDL de los servicios que se van utilizando para que si solicita la ejecución de un método perteneciente a un servicio ya utilizado, no sea necesario traer todo el documento que especifica su forma de uso.
- ***JsObjectTable***: Es una clase cuyos objetos abstraen una tabla en la página del cliente con la capacidad de gestionar su forma (elementos que contiene la tabla) con el objetivo de no crear *tags* o conocer mucho de DOM, sino sólo llamar a ciertos métodos que expone este objeto, tales como agregar columnas y regenerar la tabla en base a las columnas definidas con la data disponible.
- ***JsObjectCombo***: Cumple funciones similares a la clase anterior con la diferencia que no trabaja con tablas sino con Combos o lo que formalmente se conoce en HTML como *Select tag.*
- ***JsObjectElement***: Permite la manipulación de propiedades de cualquier elemento de la página, solamente con hacer referencia al nombre de la propiedad y al nuevo valor de la misma página y sus elementos pueden variar por completo.

Para apreciar en detalle las relaciones entre las diferentes clases explicadas pertenecientes al modelo de programación propuesto se provee del diagrama de clases en la Figura 5.

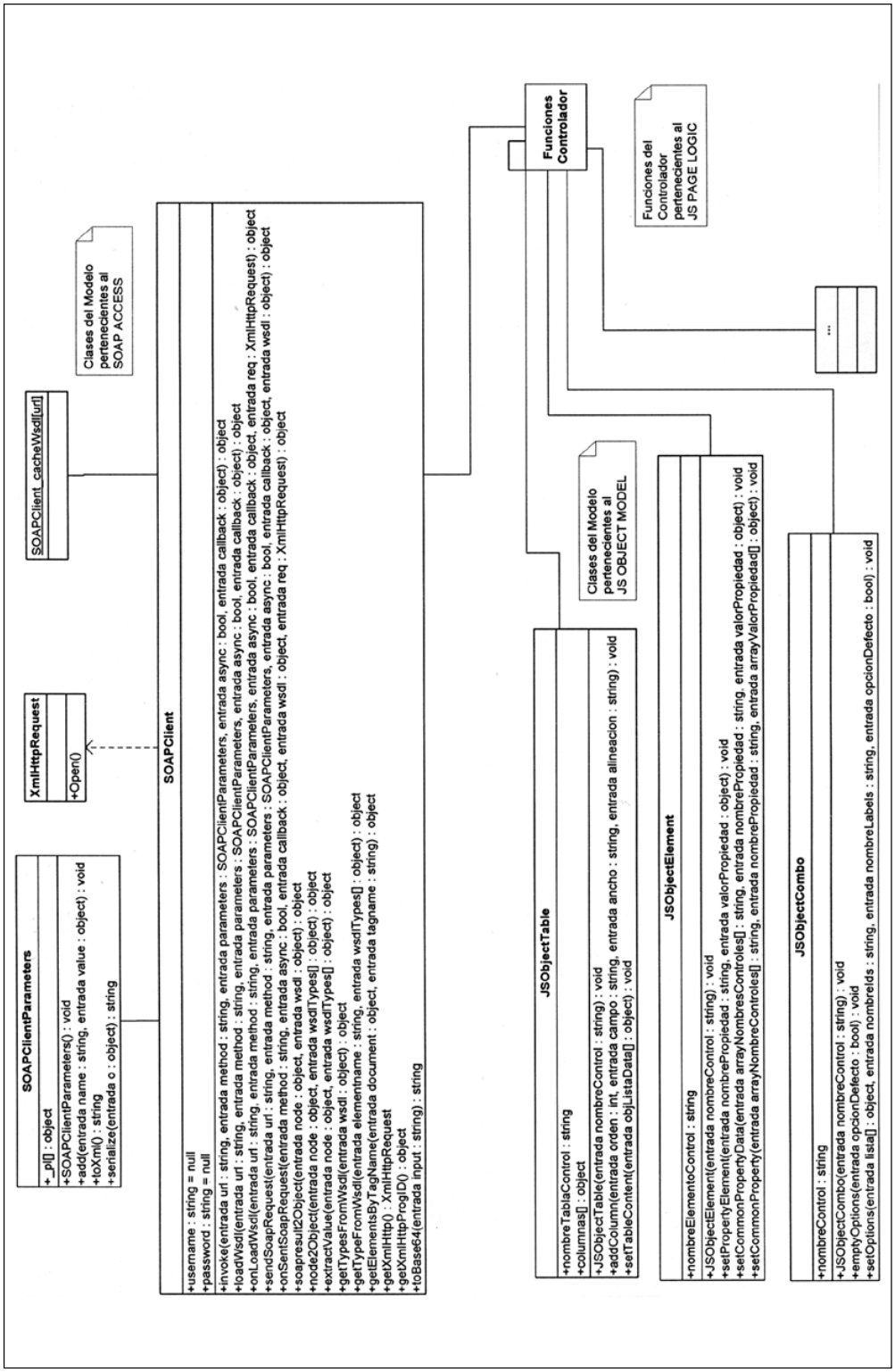

Figura 5. Diagrama de clases del modelo propuesto.

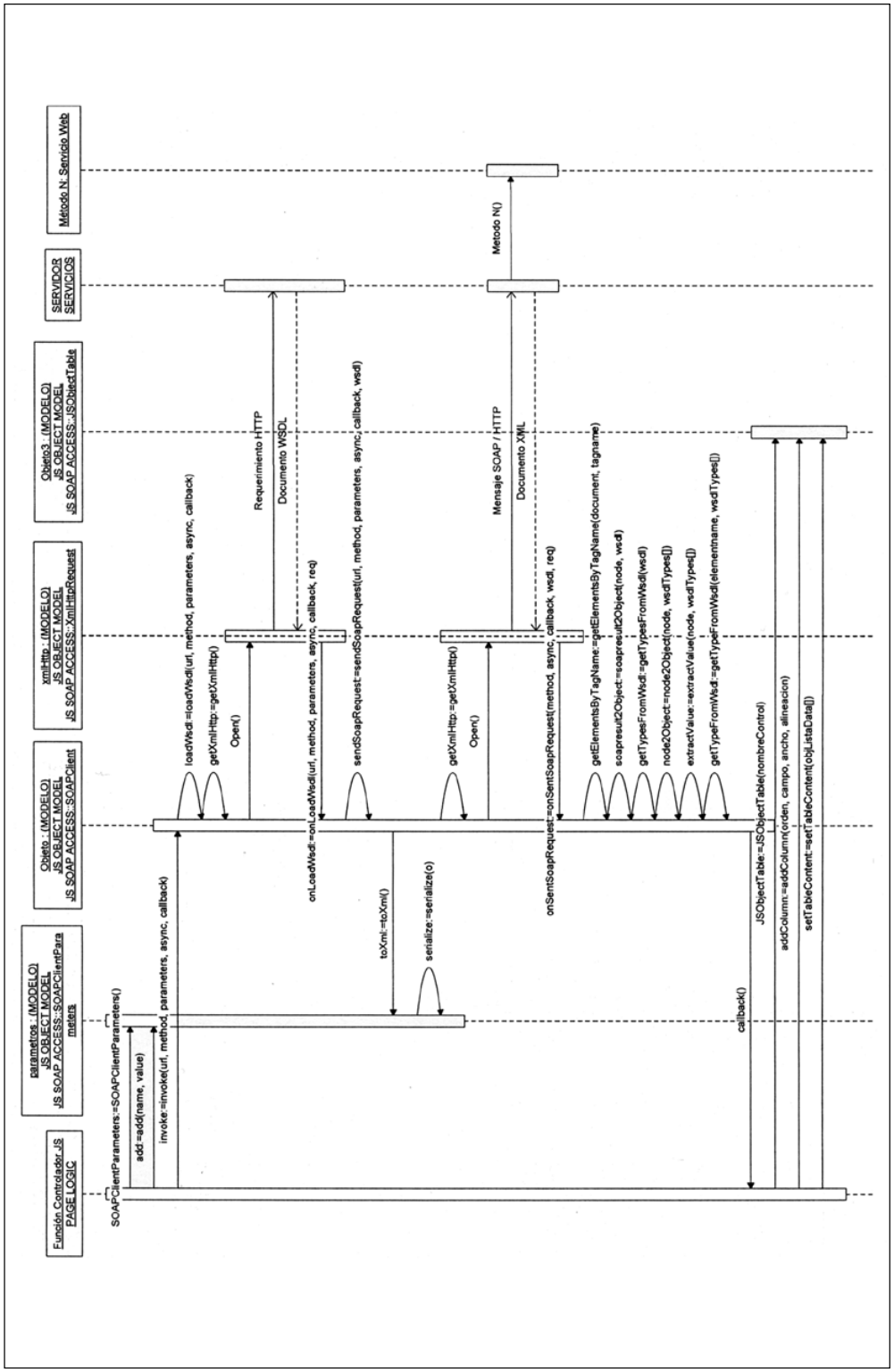

Figura 6. Diagrama de secuencias del modelo propuesto.

Diagrama de secuencia
El diagrama de secuencias detalla la iteración entre las diferentes clases a través de sus métodos en base al tiempo transcurrido; en la Figura 6 se muestra éste para el caso de uso de un servicio utilizando las clases del modelo de programación.

Por tanto, una vez definida la metodología de diseño del modelo se hace necesario validarlo a través de la implementación de un prototipo.

PROTOTIPO

Se desarrollará un prototipo utilizando el modelo de programación propuesto con el objetivo de poder verificar si el modelo de programación cumple con la solución del problema planteado, así como ratificar lo objetivos expuestos en el presente trabajo.

Especificación funcional
El prototipo trata de un solo formulario en el cual un ventanillero de una entidad financiera puede ver los movimientos (depósitos y retiros de efectivo) realizados por un cliente en sus diferentes cuentas (dólares, pesos o euros), verifica los saldos en cada una de éstas e inserta nuevos movimientos en las cuentas actualizándose de forma inmediata los saldos de las mismas, estos movimientos son conocidos también como transacciones en cuenta.

Casos de Uso
Dentro de la pequeña aplicación tenemos tres Casos de Uso principales que debe de seguir el ventanillero, primero debe buscar al cliente, cuando éste es encontrado (seleccionado) se deben mostrar sus datos así como todas las cuentas que tiene registradas, luego se selecciona una de las cuentas mostrándose todos los movimientos así como el saldo total de la misma, además, con ésta seleccionada puede realizarse la inserción de una nueva transacción (depósito o retiro) en la cuenta del cliente. Los casos de uso se muestran en la Figura 7.

Especificación Técnica
Diagrama de Despliegue
Se ha definido para el despliegue del prototipo utilizar la forma offline del modelo de programación propuesto, ya que no se utilizará un servidor Web teniendo la aplicación cliente alojada en el computador del usuario. Además, tendremos dos servidores de servicios, el primero expondrá métodos de un servicio que sólo podrán realizar operaciones de lectura o consulta sobre la base de datos ubicada en un tercer servidor y el segundo servidor de servicios expondrá métodos de un servicio que sólo pueden realizar escritura de datos. Los servicios Web están escritos en C# publicados en IIS (*Internet Information Service*), para el acceso a datos se usará ADO.NET 2.0 y la base de datos estará escrita en SQL Server 2005. El diagrama de despliegue con las características mencionadas se muestra en la Figura 9.

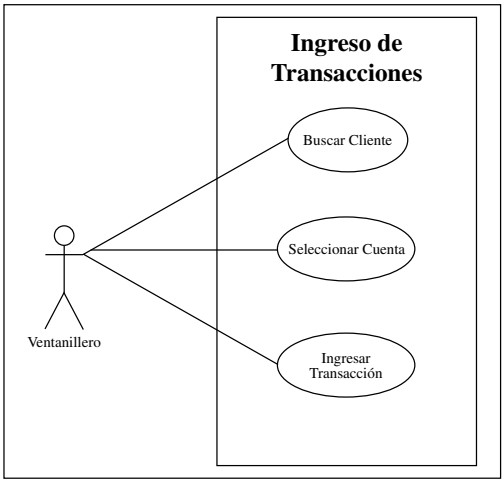

Figura 7. Diagrama de casos de uso del prototipo.

Diseño de pantalla
En la Figura 8 se muestra el diseño de pantalla de consulta e ingreso de transacciones que se desarrollará utilizando el modelo de programación propuesto.

Figura 8. Pantalla de consulta e ingreso de transacciones del prototipo.

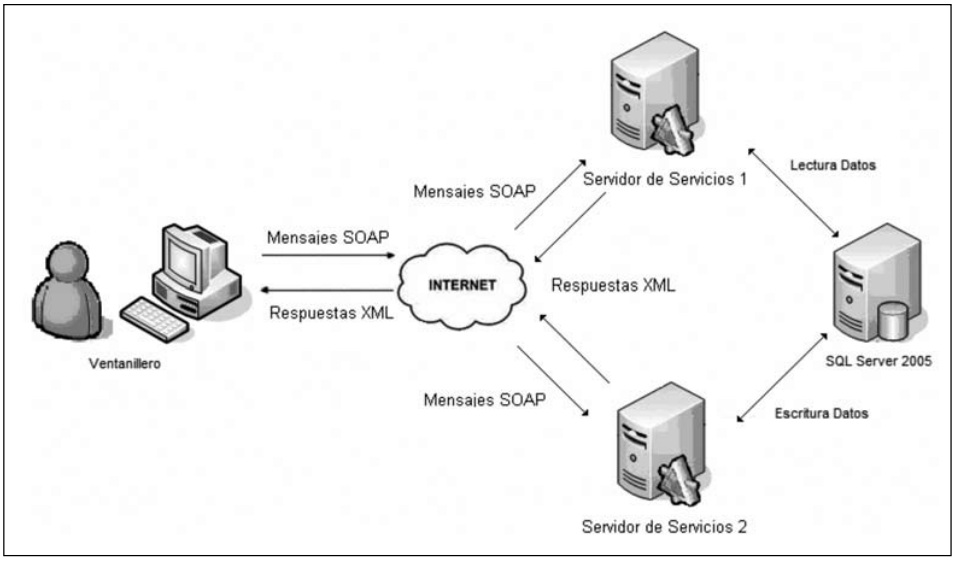

Figura 9. Diagrama de despliegue del prototipo.

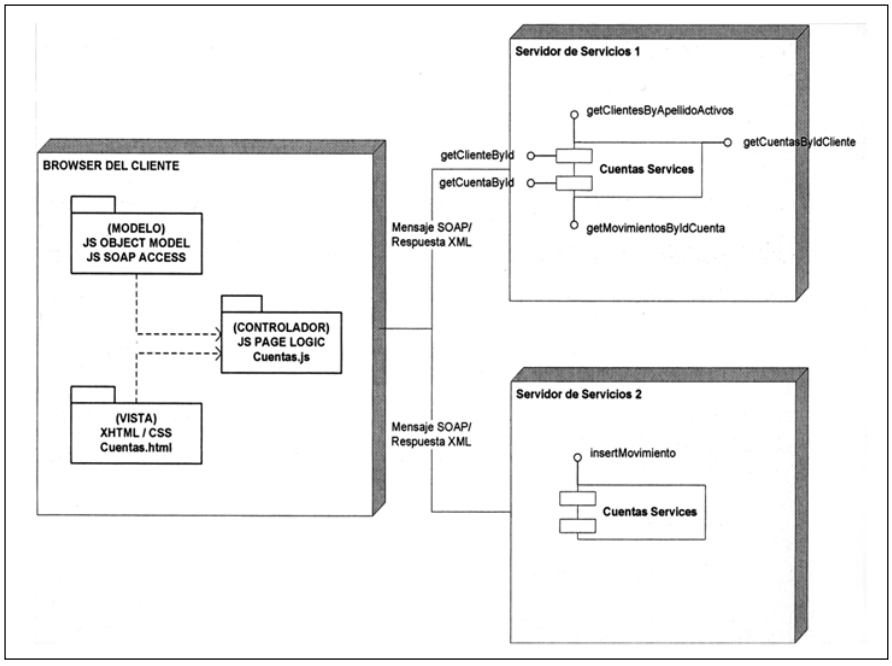

Figura 10. Diagrama de componentes del prototipo.

Diagramas de Componentes

Con respecto a la disposición de los componentes en el prototipo y basándonos en el modelo de programación propuesto es que tendremos tres nodos:

- ***Browser* del Cliente**: Contiene los tres paquetes definidos por el modelo de programación y en este caso puntual tendremos como *Js Page Logic* el archivo escrito en *JavaScript* Cuentas.js que corresponde a la única pagina del prototipo.

- **Servidor de Servicios 1**: Este servidor expondrá varios métodos de un servicio (*CuentasServices*) que sólo pueden realizar lectura o consulta en la base de datos, los nombres de los métodos se apreciarán mejor en el diagrama de la Figura 10.
- **Servidor de Servicios 2**: Este servidor expondrá un método de un servicio (*CuentasServices*) el cual sólo podrá realizar la inserción de nuevos movimientos (transacciones) en las cuentas de los clientes guardadas físicamente en la base de datos.

El diagrama de componentes con las características mencionadas anteriormente se muestra en la Figura 10.

Diagramas de Clases

El diagrama de clases es igual al definido por el modelo de programación, solo que en este caso del prototipo se agregan las clases del lado del servidor de servicios o también llamadas clases para gestión de la lógica de negocios. Resumidamente detallamos las funciones que cumplen las clases alojadas en los servidores de servicios:

- ***ClienteEntity, CuentaEntity y MovimientoEntity*** Son clases que sirven principalmente para el transporte de información entre todas las capas, son el reflejo orientado a objetos de un modelo relacional el cual nos provee la base de datos en forma de tablas. Usar clases para transporte de esta forma también es conocido como la utilización del patrón de diseño DTO (*Data Transfer Object*).
- ***ClienteData, CuentaData y MovimientoData*****:** Son las clases ubicadas en la capa de datos y las únicas con la capacidad de comunicarse con la base de datos para realizar transacciones o consultas en el mismo, para esto heredan de la clase *ModelData* e implementan la interfaz *ILogicModel*.
- ***ClienteLogic, CuentaLogic y MovimientoLogic:*** Son clases pertenecientes a la capa de negocios y cumplen la función de ejecutar métodos de la capa de datos según sea necesario y controlar las excepciones que puedan lanzarse de esta forma. Muestran una facha para la ejecución de los requerimientos que llegan; esta utilización de fachada es conocida como el patrón de diseño *Business Facade.*
- ***CuentasServices:*** Son el nombre de los dos servicios publicados en cada servidor, el primero solo ejecuta métodos de lectura ubicados en la capa de negocios y el segundo el método de inserción de nuevas transacciones.

Todas las clases mencionadas, las clases del *browser* y las relaciones entre ellas se pueden apreciar en la siguiente Figura 11.

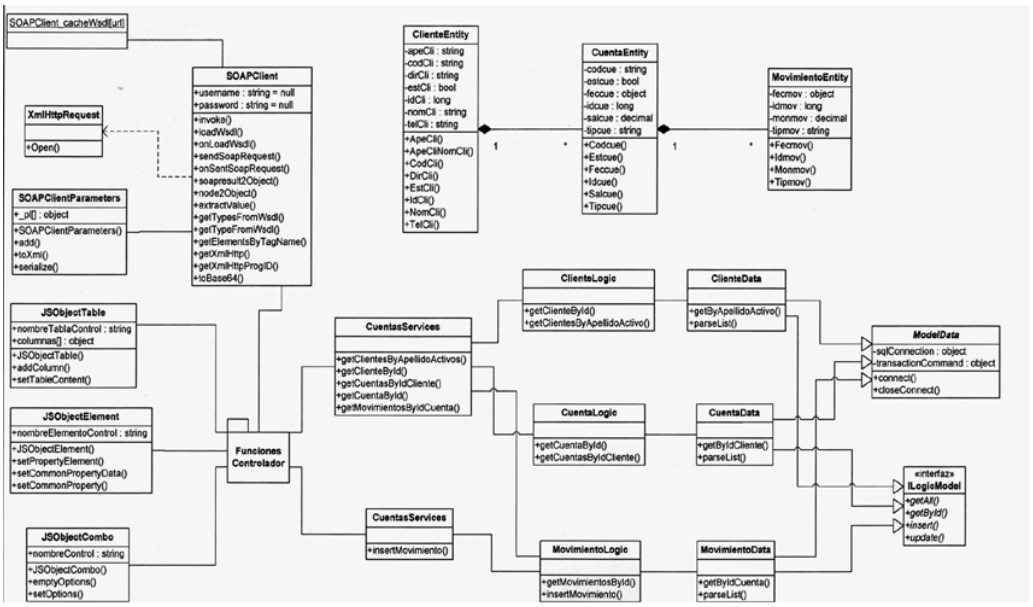

Figura 11. Diagrama de Clases del Prototipo.

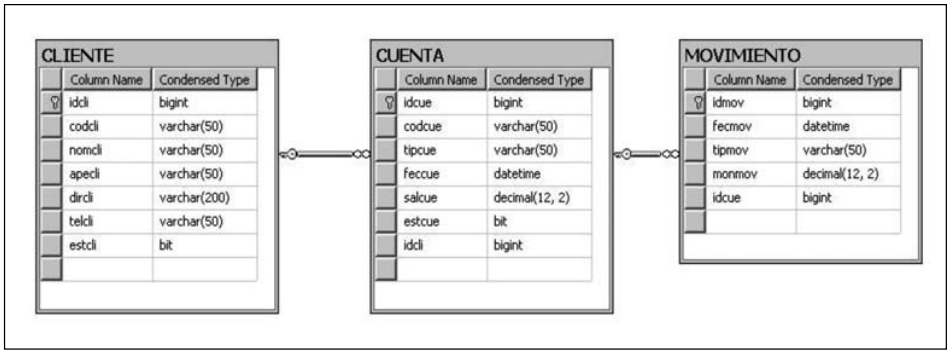

Figura 12. Diagrama Entidad Relación del Prototipo.

Diagrama Entidad Relación

Los datos de los clientes así como las cuentas y movimientos de éstas se encuentran alojados en una base de datos que sigue el modelo definido por el diagrama entidad relación de la Figura 12.

Comprobación de la Solución

A continuación mencionamos punto por punto los problemas expuestos y validamos la solución de los mismos contrastándolos con el prototipo realizado siguiendo el modelo de programación propuesto:

- **Problema:** Toda la aplicación depende del "Servidor Web", no importando lo distribuida que se encuentre la lógica de negocios en los "Servidores de Servicios". Si se llega a caer el "Servidor Web" no funciona el sistema.
  Validación: Como pudo verse en el prototipo, este problema desaparecería ya que ni siquiera es necesario un servidor Web para comunicarse con los servicios expuestos, es más, no es indispensable uno para levantar el sistema.
- **Problema:** En el supuesto que el "Cliente 1" sólo esté trabajando con los servicios en el "Servidor de Servicios 1" y cae el "Servidor Web" no podrá seguir trabajando.
  Validación: Como pudo verse en el prototipo si existiera un servidor Web y cayera no afectaría al sistema, ya que la aplicación cliente puede seguir comunicándose con los servicios de forma independiente.
- **Problema:** Los mensajes, respuestas y peticiones transitan a través de una gran distribución geográfica, que va desde el cliente al "Servidor Web", hasta el "Servidor de Servicios" y de regreso al cliente.
  Validación: Los mensajes ya no viajan una gran distancia, viajan únicamente entre *browser* del cliente y servidor de servicios.
- **Problema:** Las respuestas del "Servidor Web" siempre son páginas completas no importando si sólo se necesita una confirmación de una transacción, lo cual genera lentitud en transporte de las respuestas y mayor tráfico de información.
  Validación: Las respuestas recibidas ya no son del servidor Web sino son de los servidores de servicios en forma de archivos XML que solo contienen la respuesta a la transacción realizada, por lo que no traen información que no haya sido solicitada, sino solamente lo necesario teniendo mensajes más pequeños y ahorrando recursos de red como tiempo de transporte.
- **Problema:** Si deseamos que los Clientes 1, 2 reaccionen de manera diferente a una transacción, tendría que implementarse otro "Servidor Web", uno por cada tipo de aplicación o por lo menos poner otra aplicación diferente en el "Servidor Web".
  Validación: Ahora pudiendo instalar diferentes clientes en los computadores o ligeramente modificados según las necesidades de los usuarios, conseguiríamos una independencia entre las diferentes instancias que consumen los servicios Web.
- **Problema:** Cada vez que hacemos uso de un servicio o queremos acceder a la capa de negocios, el usuario percibirá una página en blanco deteniendo su trabajo, lo cual ayuda a la usabilidad del sistema.
  Validación: Como pudo verse el usuario ya no percibe una pantalla en blanco cuando se

realiza una transacción en los servidores ya que estas se realizan en *background*.

- **Problema:** Siempre que se quiera realizar una transacción, debemos de realizar el *submit* de toda la página y no podemos concentrarnos en una programación más limpia orientada a eventos como en los sistemas *desktop*.
  Validación: Como pudo verse ahora ya no se programa orientándonos al *submit* o *postback,* sino que nos enfocamos en una programación orientada a eventos capaz de realizar transacciones imperceptibles en el servidor, desde eventos como el *onChange* de un combo al igual que un sistema *desktop*.

Por tanto, se concluye que la definición del modelo de programación propuesto especifica las diferentes formas de despliegue (Online u Offline), los distintos componentes del modelo así como la forma en que se encuentra lógicamente agrupado, basándose en el estándar de diseño modelo vista controlador, las diferentes clases y las tareas que cumplen cada una de ellas y la forma en que éstas interactúan para poder realizar el acceso de un servicio desde su llamado hasta la forma en que la respuesta del servicio puede provocar la regeneración de parte de la página del cliente.

CONCLUSIONES

En el presente trabajo concluimos que los objetivos planteados se consiguieron al desarrollar un modelo de programación asíncrono sencillo y útil para sistemas de información Web transaccionales orientados a servicios (distribuidos), fusionando las ventajas de la programación asíncrona *AJAX* y los servicios Web.

De este modo es posible realizar uso de servicios Web publicados desde el *browser* del cliente utilizando técnicas de programación asíncrona AJAX. Una vez que sea verificada la posibilidad de uso de servicios Web por el *browser* solo es cuestión de ordenar adecuadamente las clases, capas y comunicación entre éstas al igual que si habláramos de una arquitectura corporativa de programación en el servidor. El manejo y uso de clases desde el lenguaje JavaScript es complejo y limitado, ya que éste no es orientado a objetos, es más, ni siquiera es basado en objetos, la forma de creación de clases es un artificio de la declaración de funciones en este lenguaje.

Es posible realizar un prototipo que valida la solución de los diferentes problemas planteados al inicio del presente trabajo de investigación.

Proveer de la capacidad de uso de servicios desde el *browser* aporta a los sistemas de una mayor tolerancia a fallos, mucho más que sólo usar servicios Web que son accedidos desde el servidor Web de la aplicación.

REFERENCIAS

[1] "Ajax: A New Approach to Web Applications". URL: www.adaptivepath.com/ideas/essays/archives/000385.php. Fecha de consulta: 20 de noviembre de 2007.

[2] K. Laudon. "Sistemas de Información Gerencial: Administración de la Empresa Digital". Prentice Hall, pp. 400-402. México. 2004.

[3] I. García Valcárcel y E. Munilla Calvo. "E-Business Colaborativo: Cómo Implantar Software Libre, Servicios Web y el Grid Computing para Ahorrar Costes y Mejorar las Comunicaciones en su Empresa". Fundación Confemetal Editorial, pp. 83-85. Barcelona, España. 2003.

[4] I. Laso. Internet, Comercio Colaborativo y MComercio: Nuevos Modelos de Negocio. Mundi-Prensa Ediciones, pp. 83-85. Madrid, España. 2003.

[5] "Ajax: Enhanced Interactivity and Responsiveness". URL: www.asp.net/ajax/. Fecha de consulta: Mayo de 2008.

[6] JavaScript SOAP Client. URL: www.codeplex.com/JavaScriptSoapClient. Fecha de consulta: Mayo de 2008.

[7] Proyecto JavaScript SOAP Client Library. URL: www.terracoder.com/soapclient/default.html. Fecha de consulta: Mayo de 2008.