

Algunas Experiencias en el Desarrollo de Interfaces Web

Luis A. Guerrero[±], Roberto C. Portugal[‡], David A. Fuller[†]

RESUMEN

Internet es una plataforma ideal para el desarrollo de aplicaciones distribuidas y el World Wide Web es un framework ideal para el desarrollo de interfaces. En el presente artículo se describen algunas experiencias sobre la implementación de interfaces Web para aplicaciones distribuidas, principalmente aplicaciones colaborativas. También se muestra un esquema de comunicación para aplicaciones cliente-servidor, un sistema alternativo para el almacenamiento temporal de datos y un sistema de notificación de eventos, todo bajo Web. Este esquema está basado en los patrones de diseño: MVC y broker. Se muestran también algunas aplicaciones desarrolladas sobre Web siguiendo el enfoque propuesto.

Palabras clave: Sistemas Colaborativos, Internet, aplicaciones Web, Interfaces.

1. Introducción

Actualmente Internet es una de las plataformas más atractivas para el desarrollo de aplicaciones distribuidas. Esto debido principalmente a su amplia extensión y al hecho de ser de dominio público, abierta, independiente del sistema operativo y de muy bajo costo. Por otra parte, la arquitectura del World Wide Web es muy simple y está constituida por un cliente (Web browser) y un servidor (Web server) que acepta y atiende solicitudes de los clientes a través del protocolo de comunicación HTTP.

La arquitectura cliente-servidor inicial del Web permitía transmitir únicamente páginas estáticas de información desde localizaciones fijas. Sin embargo, esto no fue suficiente para satisfacer la necesidad de las nuevas aplicaciones que, por ejemplo, pretendían presentar la misma información a varios usuarios de forma diferente [Trev97]. Además si algún cambio ocurría en la información que se mostraba en la interfaz del cliente, no había forma de hacerla conocer al usuario a menos que éste hiciera un *reload*. Finalmente, y debido a razones de seguridad frente a posibles virus, no había forma de almacenar ningún tipo de información en el cliente, lo que en algunas ocasiones se hacía necesario. Sin embargo, este panorama fue cambiando con el tiempo, y actualmente es posible desarrollar en el Web aplicaciones más complejas [Girg96, Rice96, Gall97, Parn97, Trev97].

[±] Depto. de Ciencias de la Computación, Universidad de Chile. E-mail: luguerre@dcc.uchile.cl

[‡] Depto. de Ciencia de la Computación, Pontificia Universidad Católica de Chile. E-mail: rportug@ing.puc.cl

[†] Depto. de Ciencia de la Computación, Pontificia Universidad Católica de Chile. E-mail: dfuller@ing.puc.cl

En el presente artículo mostramos parte de nuestra experiencia en el desarrollo de aplicaciones Web, principalmente en la construcción de interfaces Web para aplicaciones colaborativas. Describimos un enfoque basado en patrones de diseño, para la construcción de estas aplicaciones. Este enfoque provee un mecanismo de propagación de cambios para mantener actualizada la interfaz del usuario cuando ocurre algún evento. También provee un sistema de almacenamiento temporal de datos y un esquema de comunicación para aplicaciones cliente-servidor.

2. Construcción de aplicaciones Web

Después de muchas aplicaciones Web desarrolladas, hemos establecido que el mejor enfoque para la construcción de éstas se obtiene mediante una combinación de dos patrones de diseño: *Model-View-Controller (MVC)* y *Broker* [Busc96]. La combinación de estos dos patrones nos permite crear aplicaciones tipo cliente-servidor donde se separa la interfaz, del modelo de datos. De este modo, la interfaz puede ser construida sobre Web, y el modelo de datos puede ser construido como un servidor en algún lenguaje como Java. La siguiente figura muestra este modelo.

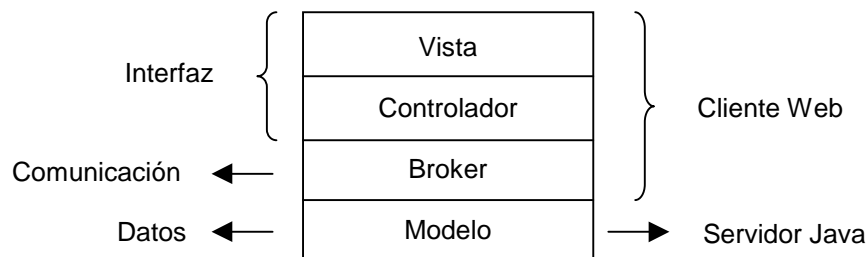


Figura 1. Modelo basado en los patrones de diseño MVC y broker

La arquitectura del Web está basada principalmente en el patrón *cliente-servidor*. Se introduce en esta arquitectura el patrón *broker*, como componente intermedio de comunicación entre el cliente y el servidor. Así, la aplicación cliente tiene acceso a la funcionalidad del servidor mediante el envío de solicitudes de servicio vía el *broker*. Éste establece la comunicación con el servidor, envía los requerimientos y regresa los resultados y excepciones a la aplicación cliente. Con el empleo de este patrón se logra encapsular todas las rutinas de comunicación, tales como definición de puertos, protocolos de comunicación, localización del servidor, etc. Esto hace que la aplicación cliente sea capaz de acceder a los servicios del servidor sin tener que preocuparse por los aspectos técnicos de la comunicación.

Esta arquitectura la combinamos con el patrón *MVC* para el diseño de la interfaz del usuario. Con el empleo de este patrón se logra separar los datos almacenados en el servidor, de la interfaz de la aplicación cliente, y se obtiene, entre otras cosas, independencia entre el lenguaje de programación de la interfaz del usuario y el lenguaje de programación del servidor.

En este esquema, el componente *vista* es el responsable de presentar la información al usuario y organizar los elementos de la interfaz, como botones, áreas de texto, áreas de

selección, menús etc. Este componente define los eventos que son generados cuando el usuario interactúa con la interfaz de la aplicación. Las acciones asociadas a estos eventos son manejadas por el componente *controlador* por medio de procedimientos o funciones. El componente controlador también es responsable de activar el mecanismo de propagación de cambios, cuando sea necesario, para mantener la vista de la interfaz de la aplicación consistente y actualizada.

A través del mecanismo de propagación de cambios se asegura que todos los usuarios sean notificados de los cambios ocurridos en los datos de la interfaz de la aplicación. A través de este servicio se hace llegar una notificación a cada cliente para actualizar la vista de su interfaz. Finalmente, el componente *modelo* es representado por el servidor, y encapsula los datos de la aplicación y las operaciones sobre éstos. Para implementar el componente *broker* (o capa de comunicación en nuestro esquema) hemos usado un applet llamado *WebBroker* que describimos a continuación.

3. El applet WebBroker

Siguiendo el esquema de patrones definido en la sección anterior podemos ahora decir que, en general, una aplicación consta de dos partes principales: la interfaz (capa de vistas), y el contexto (capa de modelo). La siguiente figura muestra este esquema.

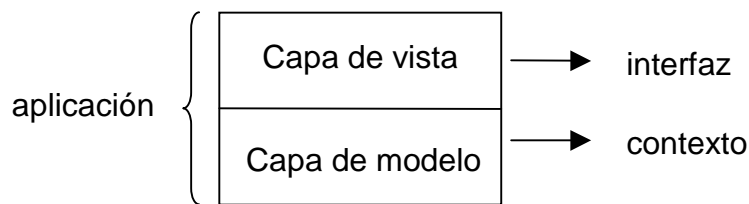


Figura 2. Capas de una aplicación

La interfaz se encarga de la representación de los datos y de la interacción de los usuarios con el contexto. Ésta puede ser creada usando los lenguajes HTML [Bern94] y JavaScript [Flan97b]. El contexto contiene los datos de la aplicación, y puede ser creado con algún lenguaje de uso general como Java [Flan97]. En el presente trabajo el contexto de la aplicación está implementado como un servidor Java.

El modelo presentado en la figura anterior permite separar la interfaz, de los datos de la aplicación. Falta ahora proveer una forma eficaz para comunicar estos dos componentes. Esta comunicación por lo general requiere el uso de sockets o de RMI. [Haro97]. En nuestro caso, hemos creado un applet llamado "*WebBroker*" que encapsula la comunicación con el servidor a través de sockets. Este applet es insertado en la hoja HTML de la aplicación, y sus métodos y atributos son invocados desde funciones JavaScript, usando la tecnología *LiveConnect* de Netscape [Flan97b].

WebBroker encapsula tres características principales: comunicación con el servidor, almacenamiento de variables temporales y un mecanismo de notificaciones. Las siguientes secciones describen estas características.

3.1. Comunicación con el servidor

La comunicación se realiza a través de sockets. El siguiente extracto de código muestra las variables del WebBroker, el método de inicialización, el destructor y el método de comunicación con el servidor.

```
import java.applet.Applet;
import java.net.*;
import java.io.*;
import netscape.javascript.JSObject;
import java.lang.Integer;

public class WebBroker extends Applet implements Runnable {

    String srvname;           // Server name
    int srvport;             // Server port
    JSObject handWindow;    // Handler for the current browser window
    Socket s;               // Communication socket
    DataInputStream i;      // Communication input
    PrintStream o;         // Communication output
    private String vars[] = new String[10]; // Variables vector
    Thread notificador;     // For the notifications handler
    ServerSocket socketClient; // For the notifications handler
    private int ntPort;     // Notification port

    public void init() {
        // Applet initialization
        handWindow = JSObject.getWindow(this); //handler for netscape window
        URL appletCodeBase = this.getCodeBase();
        srvname = appletCodeBase.getHost();
        srvport = Integer.parseInt(getParameter("WebServerPort"));
        ntPort = Integer.parseInt(getParameter("notificationsPort"));
        if (ntPort != 0) {
            notificador = new Thread(this);
            notificador.start();
        }
        for (int k = 0; k < 10; k++) vars[k] = "null";
    }

    public void destroy() {
        // Destroy the thread
        if (ntPort != 0)
            notificador.stop();
    }

    public String requestService(String service) {
        // Java server request services method
        String answer = "";
        try {
            this.s = new Socket (srvname,srvport);
            this.i = new DataInputStream(new BufferedInputStream(s.getInputStream()));
            this.o = new PrintStream(new BufferedOutputStream(s.getOutputStream()));
            o.println(service); // Send the message
            o.flush();
            answer = i.readLine(); // Wait for an ACK
            s.close();
        } catch (IOException ex) {
            answer = "ERROR: " + ex.toString();
        }
        return(answer); // return answer to JavaScript function
    }
}
```

El método de inicialización requiere que se pase como parámetro si se desea o no recibir notificaciones. En caso de querer recibir notificaciones, el applet crea un thread para

capturar los eventos que son propagados por el *contexto* y que llegan al puerto de notificación del thread.

Un típico llamado al applet WebBroker desde una hoja HTML es el siguiente:

```
<APPLET CODE="WebBroker.class" WIDTH=0 HEIGHT=0 NAME="WebBroker" MAYSCRIPT>
<PARAM name="WebServerPort" value="1234">
<PARAM name="notificationsPort" value="9595">
</APPLET>
```

En este ejemplo se indica que la interfaz va a recibir las notificaciones en el puerto 9595. En caso de que no querer usar este servicio, se debe pasar como parámetro el puerto 0. También se pasa como parámetro el puerto por el que escucha el servidor. El método destructor detiene y elimina el thread notificador de eventos, en caso de haberse creado. Cada vez que el cliente envía un mensaje al applet, éste lo pasa al servidor, espera la respuesta y la regresa nuevamente al proceso que invocó el servicio. De esta forma, se pueden tener funciones, por ejemplo en JavaScript, que invoquen métodos de un servidor Java. El siguiente ejemplo muestra una invocación de servicios desde una hoja HTML con una función JavaScript.

```
function fnCreateBox() {
if ((document.frm.name.value != "") && confirm("Create a new box?")) {
    msg = "createNewBox:" + env + "," + session + "," + document.frm.name.value
    serverMsg = top.menu.document.WebBroker.requestService(msg)
    if (serverMsg != "OK")
        alert(serverMsg)
    else
        alert("The box object have been created!")
}
}
```

En el ejemplo anterior se invoca, desde JavaScript, un método para crear un objeto *box* en un servidor hecho en Java, imprimiendo luego un mensaje con el resultado de la operación. En este caso, el programador de la interfaz no tiene que preocuparse de los aspectos de comunicación. Sólo invoca al applet y usa los métodos que el servidor provee.

3.2. Almacenamiento temporal de variables

En muchas aplicaciones se requiere el almacenamiento temporal de variables o datos. La única opción que se tiene, al momento de escribir este artículo, son las "*cookies*" [Flan97b] las cuales brindan una pequeña (aunque generalmente suficiente) área para almacenar datos en la máquina del usuario. Sin embargo, por lo general la opción de poder almacenar cookies puede ser desactivada desde el menú de opciones del browser del usuario (en los browsers más usados). El applet WebBroker cuenta con un sistema de almacenamiento de variables, que consiste en un vector de tiras de caracteres. A través de dos funciones (*putVar* y *getVar*) es posible almacenar y recuperar estas variables. A continuación se muestran estas dos funciones que son implementadas como métodos del applet WebBroker.

```
public void putVar(int x, String st) {
    // puts a variable in the array
    vars[x] = st;
}
```

```

public String getVar(int x) {
    // Get a variable from the array
    return(vars[x]);
}

```

En el caso del código del applet mostrado en este artículo, se definió un vector de variables de tamaño 10, aunque este valor puede ser modificado en caso de necesitarlo. Así por ejemplo, desde JavaScript se pueden guardar y recuperar variables de la siguiente manera:

```

function fnUserData() {
    if ((document.frm.usr.value == "") || (document.frm.pas.value == "")) {
        alert("Incomplet data!")
    }
    else {
        msg = "checkUserData:" + document.frm.usr.value + "," + document.frm.pas.value
        msgServer = document.WebBroker.requestService(msg)
        if (msgServer == "OK") {
            document.WebBroker.putVar(1,document.campos.usr.value)
            document.WebBroker.putVar(2,document.campos.pas.value)
            location = "init.html"
        }
        else alert("Access denied!")
    }
}

function fnMenu(op) {
    if (op == 1) { // New file
        if (document.WebBroker.getVar(1) == "null")
            alert("Acces denied!")
        else {
            location = "menu3.html"
        }
    }
    ...
}

```

La primera función chequea el nombre y contraseña del usuario, y si son correctos, los almacena en el vector de variables. La segunda función permite el ingreso a una opción del menú solamente si el usuario tiene su nombre registrado en el vector de variables. Cuando el usuario abandona la aplicación, estas variables se limpian. De este modo se obtiene cierto grado de seguridad, pues aunque la hoja HTML haya quedado en memoria caché, y sea luego recuperada, al estar las variables sin datos, no se permite al usuario tener acceso a la información.

3.3. Mecanismo de notificaciones

En la mayoría de aplicaciones distribuidas, por ejemplo en sistemas colaborativos, las notificaciones a los usuarios de ciertos eventos que ocurren, son fundamentales. El applet WebBroker también implementa un mecanismo de manejo de notificaciones, de modo que si una notificación llega desde el servidor, el applet permite tomarla y propagarla al manejador de eventos definido en el cliente.

El siguiente fragmento de código muestra el mecanismo usado en el applet para recibir las notificaciones del servidor, y transmitir las al cliente.

```

public void run() {
    try {
        socketClient = new ServerSocket(ntPort);
        while (true) {
            Socket socketClass = socketClient.accept();
            DataInputStream dis = new DataInputStream(new BufferedInputStream _
                (socketClass.getInputStream()));
            String notify = dis.readLine();
            String args[] = {""} ;
            args[0] = notify ;
            handWindow.call("notificator",args);
            socketClass.close();
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    } finally {
        validate();
        try {
            socketClient.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}

```

En el código anterior, el applet recibe la notificación en la variable `notify` y la envía a una función llamada `notificator(args)` que debe estar definida en el cliente. Por ejemplo, el siguiente fragmento de código JavaScript (de una aplicación chat), define esta función manejadora de eventos.

```

function notificator(notify) {
    if (notify.substring(0,13)=="<chatMessage>") {
        document.formData.text_chat.value = notify.substring(13) + "\n" +
            document.formData.text_chat.value
    }
    else
        if (notify.substring(0,13)=="<userArrived>") {
            newUser = new Option(notify.substring(13))
            fnInsertUser(newUser)
        }
    else
        if (notify.substring(0,12)=="<userDeparture>") {
            userDepart = notify.substring(12)
            fnRemoveUser(userDepart)
        }
}

```

En el ejemplo anterior la función `notificator` o manejador de eventos, maneja tres tipos de notificaciones: la llegada de un mensaje de texto, el arribo de un nuevo usuario y la partida de un usuario.

4. Ejemplos de aplicaciones

En esta sección presentamos dos aplicaciones desarrolladas utilizando el enfoque propuesto. La primera aplicación es un presentador de slides HTML y la segunda aplicación es un editor de texto colaborativo asincrónico. Para la implementación del modelo de datos de ambas aplicaciones se utilizó el servidor Java del framework para la construcción de aplicaciones colaborativas TOP [Guer98].

4.1 Aplicación "Slider"

En esta aplicación un usuario coordinador va mostrando distintas hojas HTML a un grupo de personas que se encuentran geográficamente distribuidas. Cuando el coordinador de la sesión cambia la hoja HTML de su browser, a todos los otros usuarios conectados en esa sesión se les muestra la nueva hoja cargada por el coordinador. El coordinador puede además mover un telepuntero, que será movido de igual forma en las interfaces de todos los otros usuarios. También puede resaltar partes del texto a través de una herramienta para hacer líneas. Este texto también será resaltado en todos los otros usuarios. La siguiente figura muestra la interfaz Web de esta aplicación.



Figura 3. Aplicación "Slider"

La interfaz de esta aplicación fue desarrollada usando solamente HTML y JavaScript. El applet WebBroker fue incluido en la hoja principal de Slider. A través del servicio de notificaciones del WebBroker se propagan tres eventos: el cambio de una hoja por parte del coordinador, el movimiento del telepuntero y las líneas para resaltar texto. El primer evento envía por parámetro la dirección URL de la nueva hoja HTML, y los otros dos eventos envían coordenadas de pantalla.

4.2 Editor colaborativo asincrónico

Este editor permite la escritura colaborativa de documentos entre varios usuarios. Para ingresar al editor los usuarios deben indicar su nombre y contraseña. El editor presenta opciones para crear nuevos documentos, asignar usuarios a documentos, escribir y modificar texto, y dos opciones para visualizar el documento, una opción muestra el documento final, y la otra opción muestra el documento con más información, que indica el autor de cada sección, la última hora y fecha en que fue modificada y los comentarios hechos por otros autores.

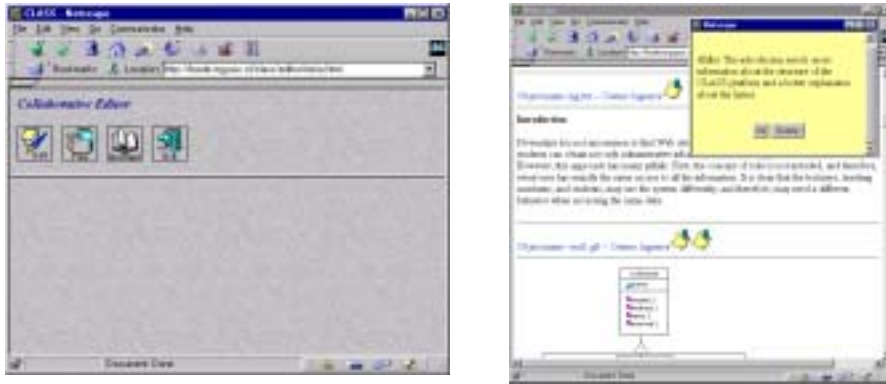


Figura 4. Editor colaborativo asincrónico sobre Web

Esta aplicación no utiliza el servicio de notificaciones del applet WebBroker, debido principalmente a que es una aplicación de naturaleza asincrónica. Los servicios de almacenamiento temporal de datos y de comunicación con el servidor sí son utilizados.

5. Conclusiones y trabajo futuro

En el presente artículo se mostró un modelo para el desarrollo de aplicaciones sobre Internet con interfaz Web. El modelo está basado en dos patrones de diseño: MVC y Broker y permite la creación de aplicaciones distribuidas tipo cliente-servidor. Como componente de comunicación del modelo se mostró un applet llamado WebBroker (del cual se incluye el código completo) que tiene tres características principales: permite la comunicación entre un cliente Web y un servidor, permite el almacenamiento temporal de datos, y provee un servicio para recepción y propagación de eventos.

El enfoque propuesto permite lograr independencia entre la interfaz y el contexto de una aplicación. El applet WebBroker provee la comunicación necesaria entre esta interfaz y su contexto. Esto permite sacar el mejor provecho de Internet para el desarrollo de aplicaciones distribuidas, y a la vez sacar el mejor provecho del World Wide Web para el desarrollo de las interfaces de estas aplicaciones.

Como trabajo futuro, se espera incluir algoritmos de criptografía en el applet WebBroker para proveer un servicio de comunicación más seguro.

Reconocimientos

Este trabajo fue parcialmente apoyado por el Fondo Nacional de Ciencia y Tecnología (FONDECYT), proyecto 198-0960.

Referencias

[Bern94] Berners-Lee, T. y Connolly, D. *Hypertext Markup Language Specification - 2.0*. IETF HTML Working Group, RFC1866, 1994.

[Busc96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. y Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[Flan97] Flanagan, D. *Java in a Nutshell*. Segunda Edición, O'Reilly, Mayo, 1997.

[Flan97b] Flanagan, D. *JavaScript, The Definitive Guide*. O'Reilly & Associates, Inc., Segunda Edición, Enero, 1997.

[Gall97] Gall, U. y Hauck, F. *Promodia, A Java-Based Framework for Real-Time Group Communication in the Web*. Proceedings of the Sixth International World Wide Web Conference, Santa Clara, California, EEUU, Abril, 1997.

[Girg96] Girgensohn, A., Lee, A. y Schlueter, K. *Experiences in Developing Collaborative Applications Using the World Wide Web "Shell"*. Proceedings of ACM Hypertext'96, pp. 246-255, Washington, EEUU, Marzo, 1996.

[Guer98] Guerrero, L.A. y Fuller, D. *Objects for Fast Prototyping of Collaborative Applications*. Proceedings of CRIWG'98, Rio de Janeiro, Brasil, Setiembre, 1998.

[Haro97] Harold, E.R. *Java Network Programming*. O'Reilly & Associates, Inc., Febrero, 1997.

[Parn97] Parnes, P., Mattson, M., Synnes, K y Schefstr, D. *The mWeb Presentation Framework*. Proceedings of the Sixth International World Wide Web Conference, Santa Clara, California, EEUU, Abril, 1997.

[Rice96] Rice, J. Farquhar, A., Piernot, P. y Gruber, T. *Using the Web Instead of a Windows System*. Proceedings of CHI'96, Vancouver, Canada, 1996.

[Trev97] Trevor, J., Koch, T. y Woetzel, G. *MetaWeb: Bringing Synchronous Groupware to the World Wide Web*. Proceedings of the European Conference on Computer Supported Cooperative Work, ECSCW'97, Lancaster, 1997.