

# DESARROLLO DE APLICACIONES MULTICANAL APLICANDO MVC

Miguel García Rodríguez, Daniel Fernández Lanvin, Jose Emilio Labra Gayo, Aquilino A. Juan Fuente

*Universidad de Oviedo  
Calvo Sotelo s/n 33007 Oviedo España*

## RESUMEN

En ingeniería del software una aplicación Web es aquella que los usuarios usan desde un servidor Web a través de Internet o de una intranet. Las aplicaciones Web son populares debido a la ubicuidad del navegador como un cliente, así como la habilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software en miles de potenciales clientes es otra razón de su popularidad. La última tendencia arquitectónica, a la zaga siempre de los avances tecnológicos, es acceder a los servicios de la red mediante diferentes tipos de dispositivos como PDAs, Teléfonos móviles, etc. Esto deriva en la necesidad de evolucionar las aplicaciones Web “clásicas” para que se puedan autoadaptar a las características de los distintos dispositivos cliente. Esta comunicación pretende mostrar un caso práctico de desarrollo de aplicaciones multicanal basándose enteramente en el empleo de tecnologías open-source, y dentro del ámbito de las tecnologías java, si bien los principios sobre los que se sustentan los frameworks aquí empleados son perfectamente adaptables a otros tipos de tecnologías.

## PALABRAS CLAVES

Multicanal, MVC, STRUTS, XSLT, XML, STXX

## 1. INTRODUCCIÓN

En ingeniería del software una aplicación Web es aquella que los usuarios usan desde un servidor Web a través de Internet o de una intranet. Las aplicaciones Web son populares debido a la ubicuidad del navegador como un cliente, así como la habilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software en miles de potenciales clientes es otra razón de su popularidad.

El mayor atractivo que este tipo de aplicaciones despierta en las empresas es sin duda la capacidad de llegar al mayor número de usuarios posibles, independientemente de su ubicación física o de la hora de acceso. Hasta hoy en día, el medio por el cual el usuario de Internet accedía a las aplicaciones publicadas era más bien homogéneo: el navegador Web. Si bien esto presentaba ciertas discrepancias en las capacidades de los clientes -dado podía tratarse del Explorer o el Mozilla, sobre Windows o sobre Linux-, las diferencias eran más bien asumibles, y relativamente sencillo (y barato en coste) conseguir que una misma aplicación se comportara de forma similar en, cuando menos, aquellos navegadores más difundidos. Sin embargo, el mercado tecnológico ofrece un amplio abanico de dispositivos que se están comenzando a emplear como clientes Web, como PDAs, Teléfonos móviles, etc. Los diferentes dispositivos difieren en capacidades de visualización y generación de interfaces en general. Esto deriva en la necesidad de evolucionar las aplicaciones Web “clásicas” para que se puedan autoadaptar a las características de los distintos dispositivos cliente. Estas aplicaciones Web se conocen como “Aplicaciones Web Multicanal”, es decir aplicaciones Web que pueden ser consumidas desde distintos tipos de clientes remotos.

## 2. MODELO DE ARQUITECTURA WEB

La evolución de la arquitectura Web ha sido acelerada [LANVIN03], como la tecnología que la precedía. Podemos marcar como punto de ruptura importante la combinación de JSPs y los servlets como implementación del patrón de diseño Modelo-Vista-Controlador [GAMMA02] al desarrollo Web en Java, lo cual se conoce como Modelo 2 de arquitectura.

El framework Struts [STRUTS] es un proyecto de la “Apache Software Foundation” que tiene como objetivo proporcionar un entorno específico para la construcción de grandes aplicaciones Web. Se trata de un entorno basado en el concepto de “Modelo 2” (Model View Controller) [GAMMA02], patrón proveniente del SmallTalk, donde se separan claramente las responsabilidades de control de navegación, presentación, y lo que se denomina *modelo* desde el punto de vista del MVC, que en términos de separación de capas vendría a ser la confusión de la capa de negocio junto con la de persistencia. El núcleo del entorno está constituido por un conjunto de tecnologías estándar, como son los Servlets, JavaBeans, ResourceBundles y XML.

### 2.1 Modelo 2x: Extendiendo el MVC

El Modelo 2X, el más reciente de todos junto con las implementaciones de SOA [ERL04] aparece como evolución del Modelo 2 para dar soporte a las mencionadas aplicaciones Web multicanal. El medio habitual para lograr publicar la misma aplicación para distintos dispositivos es emplear plantillas XSL para transformar los datos XML.

Así, las implementaciones Java del Modelo 2X son una combinación entre el MVC y transformaciones XSLT. Su implementación más popular usa el modelo de Struts, sustituyendo las JSPs (vistas) por las distintas hojas de estilo XSLT que se requieran para los distintos dispositivos, y Beans serializados a XML. De esta forma, cuando una petición (*request*) llega al controlador (un *servlet*), este delega en la parte del modelo que *sabe* que debe ejecutar. La diferencia estriba en que, una vez obtenida la información a mostrar al usuario, en lugar de renderizarla por medio de una JSP, el controlador *decide* (en base a su configuración) cual de las hojas de transformación debe aplicar a dicha información para que, previamente serializada a un documento XML, se genere la salida adecuada para el dispositivo solicitante. Es decir, una misma página que pueda ser mostrada a 3 tipos de dispositivos distintos implicará programar 3 hojas XSLT diferentes. La discriminación la realiza el controlador en base al parámetro *user-agent* que todo cliente HTTP incorpora en sus peticiones al servidor. Así, una misma petición puede dar lugar a un documento HTML para el Explorer, un documento HTML para el Mozilla, o bien un documento WML para un cliente WAP. La lógica de presentación subyacente es exactamente la misma, debiéndose siempre generar un documento XML con el superconjunto de información susceptible de ser mostrado en la página independientemente del dispositivo<sup>1</sup>. Al igual que el controlador decide que plantilla XSLT aplicar en cada caso, también es posible especificar quien realizara la tarea de convertir el XML a lo que convenga, es decir si el cliente se conecta desde un Explorer delegar la conversión a dicho navegador para liberar de esta carga al servidor sin embargo si el cliente se conecta desde un PocketPC o similar que sea el propio servidor el que asuma este rol y así simplemente al cliente le llegará el html, wml o lo que sea sin tener que realizar proceso alguno. La salida generada por tanto mediante la aplicación de la XSLT no tiene porqué limitarse a distintos formatos de la pantalla en HTML, sin que puede tratarse de PDFs (Mediante XML/FO, WML o cualquier otro lenguaje de marcado).

El modelo 2X de arquitectura pretende, en definitiva, reutilizar aquella parte común a todos los dispositivos de la capa de presentación de la aplicación, entendiendo como tal el mapa de navegación de la aplicación, entre otros aspectos de la misma.

### 2.2 El framework STXX

STXX [STXX] (Struts for Transforming XML with XSL) es una extensión del Framework Struts para soportar XML y tecnologías de transformación de XML como XSL. STXX tiene como objetivo proporcionar un entorno específico para la construcción de grandes aplicaciones Web multicanal y seguir permitiendo

<sup>1</sup> El desarrollador puede decidir, ante un navegador con una superficie pequeña, prescindir de banners u otra información no esencial dentro de la página, información que no obstante deberá estar en el XML en el caso de que el cliente sea un navegador Web completo.

también el desarrollo de aplicaciones Web basadas en Struts. Se trata de un entorno basado en el concepto de “Modelo 2x”. En este framework se modifican los objetos *action* de Struts para que devuelvan XML que será transformado por tecnologías como XSL y Velocity (Anakia). La idea de STXX es eliminar la necesidad de usar JSP y las “tag libraries” de la capa de presentación del framework de Struts. Sin embargo STXX no fuerza a seguir el camino del XML, ambas tecnologías pueden trabajar juntas.

### 3. CASO PRÁCTICO. DESARROLLO DE UNA APLICACIÓN MULTICANAL

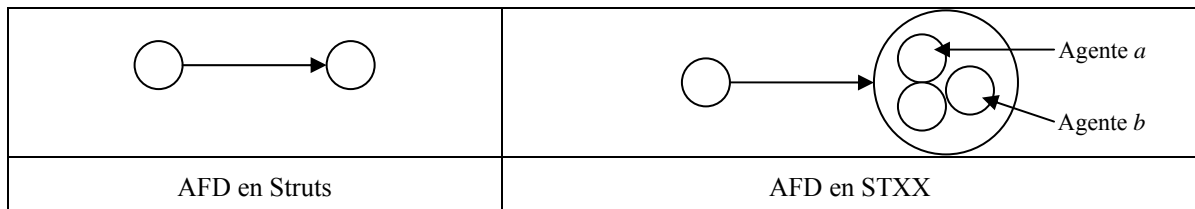
Para evaluar el desarrollo de aplicaciones con de STXX y poder realizar una valoración real de sus posibilidades se ha realizado una aplicación Web multicanal siguiendo el modelo de n capas [FOWLER03] que atenderá peticiones de varios navegadores distintos y mostrará los mismos contenidos de forma diferente en cada caso.

La aplicación se trata de un pequeño portal de noticias relacionadas con el mundo de informática y la E.U.I.T.I.O. [EUITIO], donde uno puede registrarse y puede enviar mensajes tipo e-mail a otros usuarios registrados, pudiendo gestionar tanto los mensajes entrantes como salientes.

#### 3.1 Tratamiento del Mapa de Navegación

Uno de las mayores virtudes de Struts, por extensión de STXX, y en definitiva de las implementaciones del MVC en general es la centralización y externalización a ficheros manipulables de la responsabilidad de decidir a qué vista se va desde un determinado estado a partir de un determinado evento. Esto, a fin de cuentas, es la definición del autómata finito determinista que describe el mapa de navegación de cualquier aplicación MVC. En el caso de Struts, dicho mapa o autómata se describe en términos de ternas **origen + evento = destino**, concretamente dentro de la sección *action-mappings* del fichero de configuración.

En STXX, y siguiendo la misma línea, la definición del mapa de navegación se reduce igualmente a una labor de configuración técnica, con la salvedad de que en este caso se añade un elemento discriminante más que determina cual de las vistas debe ser aplicada: el agente de usuario. De esta forma, el origen o estado actual, junto con el evento determinan la familia de estados o vistas final, pero la última discriminación dentro de estas vistas *hermanas* se produce en base al tipo de navegador del usuario.



En definitiva, pese a que en STXX se crean nuevos microestados, a nivel de navegación se mantiene la estructura original del AFD, es decir, desde el punto de vista de este enfoque del modelo 2X el mapa de navegación es común a todos los dispositivos cliente de una aplicación, y por lo tanto debe ser reutilizado.

#### 3.2 Desarrollo del Piloto

El modelo de desarrollo de la aplicación STXX sigue la pauta de Struts según al cual reparte las distintas responsabilidades funcionales (o acciones) entre distintos objetos denominados *Action*. Mientras que en una aplicación Struts debemos extender la clase *Action* de Struts, en este caso deberemos extender la clase *Action* de STXX. La otra diferencia substancial es que mientras que en Struts un ciclo de vida de request se representa por un encadenamiento de actions que terminan en una *jsp*, con STXX deberán finalizar en un recurso “.dox” para que se realice la transformación.

En estos actions se construye el documento XML conteniendo, como se dijo antes, el superconjunto de elementos que vayan a aparecer en las distintas representaciones de la página que se está generando. Esto se

puede realizar explícitamente por programa, o bien por medio de la serialización automática que realiza STXX de todos los objetos pasados en la “request” sean de “session”, “application”, etc.

Desde el fichero de configuración, se determina qué recurso dox debe ser aplicado dependiendo del evento generado por cada action, mientras que en otro fichero distinto se discriminará qué plantilla XSLT aplicar al documento XML generado por el objeto action para cada posible agente susceptible de consumir la aplicación<sup>2</sup>.

#### 4. IMPRESIONES Y CONCLUSIONES

El modelo evaluado se presenta como una solución válida para un amplio abanico de modelos de aplicaciones. La extensión de la implementación más popular del MVC en el ámbito de las aplicaciones web conlleva la aceptación de sus inconvenientes ya conocidos (como la curva de aprendizaje), pero también el aprovechamiento de sus ventajas (reutilización de código, separación de responsabilidades, facilidad de mantenimiento, etc). La posibilidad de generar las vistas discriminando diferentes hojas XSLT no sólo permite la adaptación a diferentes navegadores sobre distintos protocolos, sino que abre la posibilidad de generar documentos no web que compartan contenidos con las vistas de la aplicación, siendo el caso más común la generación de informe en formato PDF o DOC.

Sin embargo, este modelo de desarrollo presenta un problema grave difícil de salvar, en tanto en cuanto su modelo de reutilización se sustenta sobre la base de mantener el flujo de navegación, el AFD que representa el mapa de pantallas. Mediante la adaptación al dispositivo a nivel de página (tal y como hace STXX), podemos reorganizar cualquiera de las pantallas de la aplicación para evitar adornos innecesarios cuando la consumamos desde un dispositivo de capacidad limitada. Así, conseguimos la adaptación a dispositivos con restricciones en la representación gráfica, o poco ancho de banda. Sin embargo, no solucionamos uno de los más habituales problemas de este tipo de cliente: la representación en pantallas de tamaño muy reducido. En este tipo de clientes, la adaptación razonable no pasa tanto por reorganizar los elementos de una vista, como por dividir dicha vista en varias, de forma que se eviten los scrolls en la medida de lo posible. El problema radica en que dicha solución implica precisamente la modificación de ese mapa de navegación que STXX trata de mantener, puesto que la división de una vista en varias implica la aparición de nuevos estados en el AFD, imposibilitando su utilización común.

Esto, pese a ser un inconveniente importante, no descarta la solución propuesta como válida. Sin organizamos las vistas HTML (entendiendo estas como las menos restrictivas) de forma que la carga de la página se organice por partes, se puede alcanzar un modelo de AFD común a todos los dispositivos. Sin embargo, aunque se presenta como una solución factible, estamos perdiendo la independencia del modelo de navegación que se pretende reutilizar con respecto a las subcapas superiores de la aplicación, dado que estamos condicionados por el catálogo de dispositivos susceptibles de consumir la aplicación, algo que no siempre podemos conocer a la hora del diseño del sistema.

#### REFERENCIAS

- [GAMMA02] Erich GAMMA, et al. 2002. Patrones de Diseño. Addison Wesley
- [ERL04] Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services by Thomas Erl Publisher: Prentice Hall PTR (April 16, 2004) ISBN: 0131428985
- [EUITIO] Escuela Universitaria de Ingeniería Técnica de Informática de Oviedo (<http://www.euitio.uniovi.es>).
- [STXX] <http://stxx.sourceforge.net>
- [STRUTS] <http://struts.apache.org>
- [XML] <http://xml.apache.org>
- [MODEL 2x] <http://www.javaworld.com/javaworld/jw-02-2002/jw-0201-strutsxslt.html>
- [LANVIN03] Arquitectura Web en aplicaciones java/j2ee. Cuaderno didáctico nº 39, Ingeniería informática. Abril 2004. Editorial SERVITEC. ISBN: 84-688-6580-9. Depósito Legal AS-1841-04
- [FOWLER03] Patterns of Enterprise Application Architecture, Martin Fowler. Addison-Wesley 2002. ISBN: 0321127420

<sup>2</sup> Además se establece una plantilla por defecto para aquellos navegadores no contemplados por el desarrollador de la aplicación

# UM SISTEMA DE INICIAÇÃO DE SESSÃO MULTIMÍDIA PARA PLATAFORMAS LINUX E $\mu$ CLINUX

Francisco Helder Candido do Santos Filho  
*Universidade Estadual de Campinas - Unicamp*  
*heldercs@decom.fee.unicamp.br*

Luís Geraldo Pedroso Meloni  
*Universidade Estadual de Campinas - Unicamp*  
*meloni@decom.fee.unicamp.br*

## RESUMO

Este artigo descreve um serviço de Voz sobre IP baseado no protocolo de iniciação de Sessão (SIP), denominado RT-DSPhone. O sistema foi desenvolvido para as plataformas Linux e  $\mu$ Clinux. O  $\mu$ Clinux é um sistema operacional com um *kernel* compacto, desenvolvido para dispositivos embarcados com espaço reduzido de memória. A aplicação foi desenvolvida em linguagem C para executar tanto em computador pessoal como em dispositivo embarcado. Ela utiliza a linguagem SDL (*Specification and Description Language*) para descrever e especificar as funcionalidades do sistema VoIP desenvolvido.

## PALAVRAS-CHAVE

VoIP, SIP,  $\mu$ Clinux, sistema embarcado, Protocolo, SDL.

## 1. INTRODUÇÃO

Nos últimos anos, vários estudos vêm sendo realizados na área de aplicações multimídia na Internet, as quais permitem a transmissão de áudio, vídeo e dados. Esse tema tem despertado o interesse de profissionais da área de redes de computadores e usuários em geral, principalmente o interesse pelo serviço de transmissão de voz sobre IP, ou de forma mais restrita, de telefonia IP, que assume maior importância no cenário de telecomunicações [1]. Juntamente com o crescimento da Internet, verificou-se uma expansão na utilização de dispositivos embarcados ligados à redes de computadores, para explorar a convergência de serviços como áudio e vídeo no tráfego de dados. É inevitável que o número de dispositivos embarcados irá continuar a crescer rapidamente [2]. Atualmente, é inegável a grande participação do sistema operacional Linux em vários segmentos, o qual inicialmente voltado para servidores, vem cada vez mais se popularizando como opção efetiva para *Desktops*, PDAs, dentre outras. O SIP é um protocolo de sinalização para iniciar, manter e terminar uma sessão de áudio, vídeo ou texto entre dois usuários finais.

Este artigo inicia-se apresentando as funcionalidades e serviços do sistema desenvolvido. Na sequência, são apresentados as medidas (conhecido por *footprint*) dos arquivos gerados para o dispositivo embarcado, tais como imagem do sistema operacional Linux, conhecido como  $\mu$ Clinux, sistema de arquivo e aplicação VoIP. Além disso, são descritas as características e arquitetura do sistema SIP implementado. Também se prevê alguns testes de robustez do sistema VoIP rodando em uma placa PowerQUICC II da Motorola.

## 2. PROTOCOLO SIP

O SIP é um protocolo de sinalização desenvolvido por um grupo de trabalho da IETF na RFC 3261 [3], usado para iniciar, manter e terminar uma sessão de áudio, vídeo ou texto entre dois usuários finais, utilizando o *Session Description Protocol* (SDP).

## 2.1 Usando o SIP para sinalização

SIP é um protocolo de controle do nível de aplicação que trata a sessão multimídia (conferências) como chamadas telefônicas pela internet. O SIP atua durante a fase de sinalização em que os usuários definem parâmetros para estabelecer a chamada, realizando uma troca de dados, tais como endereço de transporte, tipo de meio e Codecs.

O SIP é um protocolo estruturado em camadas, o que significa que seu comportamento é descrito em termos de um conjunto de estágios independentes, provendo suas funcionalidades. Para a implementação das funcionalidades de um terminal SIP é necessário o componente *User Agent (UA)*, que se divide em duas partes: o *User Agent Client (UAC)* sendo uma aplicação cliente que inicia um pedido SIP, e o *User Agent Server (UAS)* sendo uma aplicação servidor que recebe o pedido SIP e retorna uma resposta SIP.

A Figura 1 mostra a chamada realizada entre dois usuários, sem a participação da entidade *proxy*, pois o endereço do destinatário é conhecido. O usuário `sip:userA@143.106.50.220`, com o seu RT-DSPhone instalado no computador rodando Linux, realiza uma chamada para o usuário `sip:userB@143.106.50.218`, que está com seu RT-DSPhone instalado no dispositivo embarcado PowerQUICC II rodando o  $\mu$ CLinux.

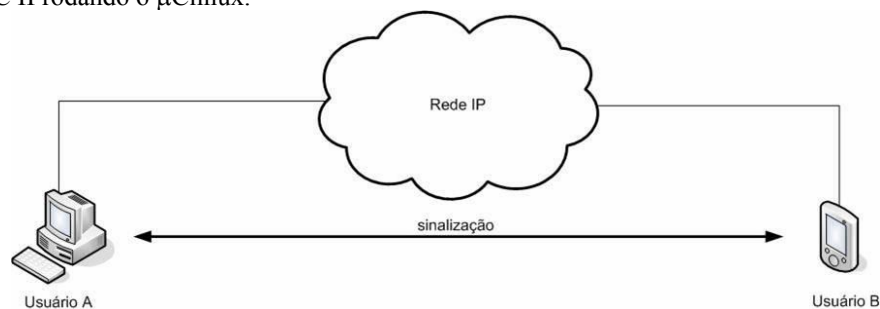


Figura 1. Cenário de chamada entre dois usuários

Para iniciar uma sessão SIP (de áudio, ou texto), como o usuário A conhece o endereço do usuário B, `UserA` gera um pedido INVITE que é enviado direto para o destinatário do pedido para o estabelecimento da sessão. Esse pedido é entregue para um UAS, que pode potencialmente aceitar o pedido de INVITE. O UAS deverá enviar uma mensagem para o originador do pedido, informando que o pedido de INVITE foi aceito, que é representada pelo envio de uma resposta 2xx. Se o pedido de INVITE não for aceito, uma resposta 3xx, 4xx, 5xx ou 6xx é enviada, dependendo do motivo pelo qual foi rejeitado. Antes de se enviar uma resposta final, o UAS pode enviar uma resposta provisória (1xx) para o originador do pedido, informando que a chamada esta em progresso.

## 3. QUALIDADE DE SERVIÇO EM VOZ SOBRE IP

Qualidade de serviço (QoS - *Quality of Service*) refere-se a capacidade de uma rede de computadores em prover melhor serviço para um tráfego de rede sobre várias tecnologias. Intuitivamente, qualidade de serviço expressa, em última análise, o grau de satisfação do usuário para com as aplicações que fazem uso da rede.

### 3.1 Fatores que Influenciam na Qualidade de Serviço

Vários são os fatores que influenciam na qualidade de transmissão de Voz em uma rede IP, que são:

- Atraso;
- Perda de Pacotes;
- Banda;
- Eco.