Universidad del Mar
Escuela de Ingeniería
Tecnologías WWW – 2

Informe de Introducción al Desarrollo de Aplicaciones Web con Entorno de Desarrollo Eclipse y Jakarta Struts

Alumno:
Luis Machuca

Docente:
Cristian Verdugo

Segundo Semestre, 2007

I.- Índice

Tabla de Contenidos

II.- Introducción

Objetivos

El objetivo de este Documento es presentar una guía para el desarrollo de aplicaciones web basadas en Java, por medio del uso de dos herramientas principales: un entorno de desarrollo adecuado para las aplicaciones (entorno estructurado **Eclipse**), y un modelo de desarrollo adecuado para las aplicaciones (Model-View-Client representado por **Struts**).

Java es un lenguaje que se ha desarrollado mucho en los años recientes (desde el año 2000). La capacidad que tiene de expresar acciones en una forma independiente de la plataforma o sistema operativo, pero permaneciendo asociado a la plataforma para eejcutar las instrucciones, lo ha hecho muy requerido para el desarrollo de sistemas de aplicaciones en modelo de cliente-servidor. El desarrollo para Java se ha facilitado además gracias a la liberación de entornos de desarrollo como NetBeans y Eclipse, y a la liberación de cada vezmáś plug-ins y librerias para aumentar y complementar las capacidades del lenguaje.

Un caso pariticular de desarrollo son las aplicaciones web basadas en Java: programas Java que son ejecutados por medio de un servidor Java y que reciben conexiones de clientes basadas en navegador (es decir, HTTP) para realizar su trabajo. El modelo de desarrollo necesario para trabajar esta clase de aplicaciones desde la perspectiva de desarrollo requiere de niveles de abstracción más elevados para describir y planificar el comportamiento del sistema, por lo cual se han desarrollado entornos o "framework" como Jakarta Struts, que permiten modelar adecuadamente el desarrollo.

En este documento, se cubren los pasos necesarios para la instalación de las herramientas del servidor (Runtime y SDK de Java; servidor Jakarta Tomcat) bajo sistemas Linux, y se describe la forma de conectar un entorno Eclipse instalado en una máquina remota para crear una aplicación web usando el modelo de desarrollo de Struts. De este modo es posible llevar a cabo la ingeniería y codificación a un equipo de producción o a un servidor preinstalado sin necesidad de alterar drásticamente el uso de esa máquina.

Descripción de las Tecnologías

La Plataforma: Linux Ubuntu Server en la Universidad del Mar

Imagen 1: Ubuntu Server

Ubuntu Server es la versión para servidores de la distribución de Linux Ubuntu. Nacida hacia 2003-2004 como un intento de llevar Debian a los usuarios comunes y corrientes, se ha convertido en una de las mayores distribuciones gracias a su formato de LiveCD con instalador, su enorme base de usuarios, y la facilidad de su administración.

Ubuntu Server es la plataforma escogida para implementar el servidor base del proyecto de **Comité de Alumnos de Informática de la Universidad del Mar, Sede Temuco**; proyecto nacido en 2006 como respuesta a la necesidad de los alumnos de cursos de informática de disponer de sistemas con un mayor nivel de acceso al software y al hardware de modo de ejecutar práctica e investigación en sus cursos de informática. El Autor de este Documento forma parte del proyecto en un rol de asesor.

Ubunt Server fue escogida por la gran base de usuarios y el soporte extendido de la versión para servidores (soporte ocifial “Long Term Support” por 3 años). El servidor se puede acceder por medio de la URL: http://informatica.temuco.udelmar.cl/

En este documento se instalarán todos los sistemas en nuestro servidor. Se enfatizará claramente cuando sea necesario que tenemos los permisos necesarios por tratarse de nuestra plataforma, pero los pasos son esencialmente los mismos en cualquier otro servidor basado en Debian. Para servidores basados en RedHat, cambiarán las rutas a algunos comandos y archivos.

El Lenguaje: Java

Java es un lenguaje que se ha desarrollado mucho en los años recientes (desde el año 2000). Es un lenguaje de alto nivel basado en el paradigma de Orientación a Objetos (*OOP*) nacido de la mano de Sun Microsystems en la década de los '90.

Java está diseñado para ser independiente de la plataforma en su codificación: es decir, los programas en Java se escriben igual sin importar la plataforma o sistema operativo subyacente, y son traducidos a un código especial denominado “*bytecode*”. A continuación el código es

interpretado por un subsistema de aplicación llamado *Java Runtime Environment*, que convierte cada instrucción "independiente de la plataforma" en una instrucción "basada en la plataforma" (por ejemplo, gráficos y multimedia que dependen del hardware). De esta forma, Java no es realmente un lenugaje independiente de la plataforma, pero se acerca tanto a este ideal que es fuertemente usado en la programación para empresas, móbiles y conexiones distribuidas.

En comparación con otros lenguajes orientados a objetos, Java es más robusto, ya que fue diseñado enteramente bajo el paradigma. Lamentablemente, es tmabién mucho más lento y duro que, por ejemplo, C++, debido en parte a que el código debe ser interpretado, y en parte a la gran cantidad de modificaciones y adiciones de componentes por las que ha pasado el lenguaje, muchas de ellas convirtiéndose en "*core components*", y que han eventualmente llevado a Java a una situación similar al llamado "*infierno de las dependencias*" en Linux.

La página web de Java: http://java.sun.com/
La versión a utilizar en este documento: **Java 5 (SDK + Runtime 1.4.2)**.

Nota especial a considerar: Java 1.4.2 ya no forma parte de las aplicaciones ofrecidas en el "frontpage" de descargas de Java. Es necesario profundizar ligeramente en los menús, o en el peor caso acceder directamente por medio de una búsqueda (en el sitio o en Google).

El Servidor: Apache Tomcat

La página web principal de Tomcat: http://tomcat.apache.org/
La versión más reciente a Julio de 2007: Tomcat **5.5.**
La versión a utilizar: **Tomcat 4.1.**

El Entorno de Desarrollo: Eclipse "Europa"

Eclipse es un "IDE" o Entorno de Desarrollo Integrado, una aplicación de software que reúne, en un solo paquete y una sola vista de aplicación, las cuatro tareas de la programación (generación de código, compilación y construcción, debug y documentación) para un lenguaje determinado.

Eclipse está desarrollado enteramente en Java usando subsistemas de lenguaje Java para realizar la conexión a distintos modelos de ambiente gráfico, destacandose Windows y Gnome bajo Linux. A diferencia de otros IDE que están orientados por diseño a un solo lenguaje o conjunto de lenguajes (como por ejemplo Visual C que opera con C y C++), Eclipse fue diseñado para ser extensible: por medio de componentes agregados es posible utilizarlo para desarrollar desde código en Java, C o C++, hasta leguajes menos comunes y más esotéricos como Ruby o incluso Haskell.

La página web principal de Eclipse: http://www.eclipse.org/
La versión más reciente a Julio de 2007: **Eclipse 3.3, comunmente llamado "Europa".**

La Arquitectura de Software: Jakarta Struts

Jakarta Struts es un "*framework*", un paquete orientado a combinar las capacidades de un

lenguaje, un entorno de desarrollo y un gestor de aplicaciones, por medio de una implementación de un modelo de Ingeniería de Software. En este caso, Struts une la programación en lenguaje Java, el desarrollo orientado a aplicaciones web, y la estructura de cliente-servidor, por medio de una vista de Ingeniería de Software llamado **MVC**: Arquitectura de Model-View-Client.

Struts implementa MVC por medio de una visual de la cual es responsable el entorno de desarrollo, y una propuesta de gestión de archivos y componentes que permite organizar el desarrollo de la aplicación.

La página web principal de Jakarta Struts: http://struts.apache.org/

La versión más reciente a Julio de 2007: **1.2**

Variables a Utilizar a lo largo del Documento

A lo largo de este documento, se presentara el progreso llevado a cabo en una máquina específica que ha sidoa daptada para tales efectos. Sin embargo, para que el lector pueda adaptar los pasos de este Documento a su propio entorno de hardware y Sistema Operativo, se ha agregado un nivel de abstracción compuesto de una notación de "variables" que representan rutas de acceso, nombres de archivo u otras componentes que el lector puede o debe modificar.

Variable	Significado	Valor (en este documento)
{java.home}	El directorio del servidor donde está instalado el paquete de Java SDK.	`/opt/java/j2sdk`
{apache_tomcat.home}	El directorio del servidor donde está instalado el paquete Apache Tomcat.	`/usr/local/apache-tomcat`
{apache_tomcat.apps}	El directorio del servidor donde están descargadas las aplicaciones web que vienen con Tomcat; en este directorio tenemos que agregar cualquier aplicación que deseamos crear.	*`{apache_tomcat.home}`*`/webapps`
{miapp.carpeta}	El directorio donde almacenamos nuestra aplicación web en desarrollo. Puede estar en nuestra máquina o en un sevridor remoto gracias a las capacidades de Eclipse.	`$HOME/Extractos/luchoapp`

Instalación de los Componentes

Para la instalación de los componentes es necesario tener una cuenta de usuario que tenga **privilegios de administrador del sistema**. Notar que esto es diferente de tener una cuenta de "*superusuario*" (`root`): las cuentas de administración requieren ejecutar la utilidad especial "`sudo`" o sus equivalentes gráficos, y **confirmar sus credenciales de identificación**, antes de ejecutar instrucciones que usen comandos o modifiquen archivos del sistema.

Java 2 SDK

Dependencias

Recordar que el sistema operativo corriendo en el servidor es **Ubuntu Server 6.06**.

Antes de instalar el SDK de Java, es necesario cumplir con dos dependencias (al menos) que dispone el sistema, que son las bibliotecas de C/C++ del periodo de tiempo de ese instalador (**~2006**). Si se intenta ejecutar el instalador sin estas dependencias se obtiene el siguiente mensaje del loader de Linux (o unmensaje similar dependiendo de la distribucion):

```
[user@server /opt/java$] ./j2sdk-1_4_2-i586.bin
ld.so: j2sdk-1_4_2-i586.bin: the following linked library can not be found:
                     libstdc++.so.5 --> [libstdc++.so.5] (not found)
                     libbgcc.so  --> [ ] (not found)
```

Para instalar las dependencias en el servidor , se ejecuta:

```
sudo apt-get install libstdc++.so.5 libstdc++-dev.so.5 libgcc gcc-3.3-base
```

El método para instalar los paquetes, puede cambiar de distribución en distribución, pero usualmente los nombres de los paquetes son iguales o muy parecidos, por lo que la instalación se facilita si se consulta la documentación del Gestor de Paquetes de cada distribución: "apt-get" y "synaptic" para Debian/Ubuntu; "Yum" para Redhat/Fedora; "YaST" para SuSE/Opensuse, u otros.

Asumiendo una buena conexión a Internet (ADSL o similar), en aprox. 6 minutos los paquetes están completamente instalados.

Nota importante: las referencias citadas son para una instalación de Ubuntu Server con las actualizaciones de seguridad y de desarrollo hasta Junio de 2007. Instalaciones limpias de Ubuntu Server en ésta u otras versiones, pueden requerir otras dependencias, o no requerir ninguna.

Descargando e Instalando

El paquete “Java 2 SDK” se descarga del sitio de Sun (`java.sun.com`); nos dirigimos al vínculo “Downloads” en el menú superior de la página, y escogemos “**Previous Releases**” o directamente el vínculo “J2SDK 1.4.2 and previous” si está disponible, para que nos lleve a las descargas de selecciones donde la versión a escoger es la **1.4.2 para Linux plataforma Intel** (i586) . Paquetes más recientes como Java 1.5.0 están disponibles pero no se recomienda instalarlos a menos que se tenga software adecuado a esa versión de Java.

La dirección de descarga es generada automáticamente por medio del *Java Ticket Download Manager*, una utilidad que corre directamente en el sitio de Java y que gestiona las descargas por sistema operativo, ubicación geográfica y tipo de conexión; por tanto la dirección de descarga se compone de una URL bastante larga y válida sólo por un corto periodo de tiempo. Se copia esta dirección y se descarga el archivo por medio de la utilidad de Linux: `wget`. Tomar nota que es una descarga *ligeramente grande* (~50 **MiB**).

```
wget -bc -o javasdk.log URL_del_ticket
```

La opción -b permite continuar la descarga en el fondo, de modo que podemos seguir operando normalmente en el intertanto; la opción -c permite continuar la descarga si, por alún motivo, es interrumpida, y la opción -o crea un archivo de log, que podemos examinar periodicmente para consultar el progreso de la descarga.

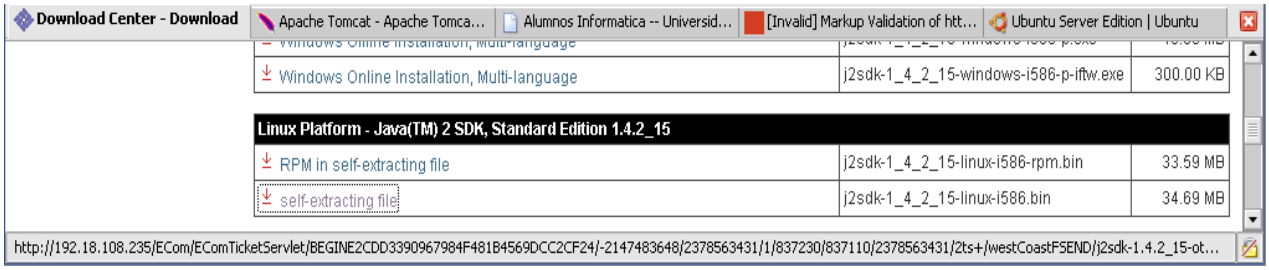

Imagen 2: La ventana del Ticket Manager. Observar la URL en la barra de estado.

Cambiemos el nombre de archivo es: `j2sdk-1_4_2-i586.bin`.

Una vez instaladas las dependencias, no es necesario realizar más ajustes (del entorno, o nada por el estilo); se procede a descargar y ejecutar el instalador con la línea de comandos:

```
cd /opt/java  # se debe crear el directorio si no existe
sudo cp /ruta/al/instalador/de/java ./j2sdk-1_4_2-i586.bin
sudo ./j2sdk-1_4_2-i586.bin
```

El programa de instalación nos mostrará el Contrato de Licencia; mantendremos presionado <*ENTER*> para saltárnosla rápidamente, hasta llegar a la pregunta si deseamos seguir

con la instalación (*aceptando los términos de la Licencia*): escogemos la respuesta "*yes<ENTER>*" y el instalador empieza a desempaquetar los archivos del SDK.

Preparación del Entorno

Una vez concluido el proceso de instalación y estando de vuelta en nuestro "shell" de usuario, debemos exportar la variable de entorno `JAVA_HOME` que determina la locación del SDK. Para esto, primero preparamos un vínculo simbólico:

```
# aun estamos en el directorio '/opt/java/'
sudo ln -s j2sdk jsdk-1.4.2
```

A esta ruta de instalación que hemos escogido, es necesario tomar nota, ya que otros programas necesitarán referirse a ella. A lo largo del documento la llamaremos por el nombre simbólico *{java.home}*.

*{**java.home**}* = /opt/java/j2sdk

Luego ejecutamos estos comandos:

```
export JAVA_HOME={java.home}
sudo echo "JAVA_HOME={java.home}" >> /etc/profile.local
```

Hecho esto, el SDK está instalado; pero para poder utilizarlo puede ser necesario reiniciar los archivos del shell, ya que la nueva variable de entorno tiene que ser cargada por el shell de sistema y por el shell de los usuarios ("**bash**", o similar). La forma más fácil de hacer esto es salir del sistema (*logout*) y volver a entrar (*login*). Otros métodos dependientes de la distribución incluyen recargar el archivo de configuración del shell con el comando "`source $HOME/.bashrc && source /etc/profile.local`".

Notar que esta preparación del entorno es necesario llevarla a cabo para cualquier usuario que pueda ejecutar Java; por tanto, hemos usado el archivo de perfiles "`profile.local`", que es independiente de la distribución y que es cargado para todos los usuarios excepto *root*; si fuese necesario, para el usuario `root` es necesario volver a ejecutar los pasos por separado[1].

Apache/Jakarta TOMCAT

Tomcat es un servidor de aplicaciones web.

1 Pero en estricto rigor, el usuario 'root' no debe necesitar hacer uso del shell o de Java. Si lo hace, es seña de una vulnerabilidad grave en el sistema.

Dependencias

Apache Tomcat (también Jakarta Tomcat) viene con dos distribuciones para Linux: una completa, y una básica. Sin embargo, la completa no es tan completa como parece ya que necesita algunas librerias para llevar a cabo la manipulación de XML. Para instalar estas librerias basta con ejecutar en el servidor:

```
sudo apt-get install libxml2 libxml2-devel docbook-xsl-pages
```

(el nombre del paquete docbook puede cambiar drásticamente entre distribuciones y versiones; una estrategia más adecuada es uilizar el Gestor de Paquetes de la distribución para instalar los paquetes requeridos para generar documentos XML)

Con una conexión a red decente, la descarga toma ~3 min.

I.- Descargando e Instalando

A continuación, nos dirigimos al sitio web de Apache Tomcat, y buscamos la version 4.1.36 en sus archivos. Podemos abrir la siguiente página en el navegador para guiarnos: http://tomcat.apache.org/download-41.cgi

Imagen 3: — Página de descarga de Tomcat.

Descargamos el paquete de la distribución completa (`apache-tomcat-4.1.36.zip`) y lo almacenamos en un directorio del sistema, en nuestro caso el directorio de programas locales. Notar que para hacer esto necesitamos ser un usuario con privilegios, y utilizar *sudo*:

```
[user@server ~] cd /usr/local
[user@server /usr/local] sudo unzip /ruta/hacia/apache-tomcat-4.1.36.zip
```

Esto creará un directorio "apache-tomcat-4.1.36" en nuestro `/usr/local`, y es ahí donde va a quedar instalado el servidor Tomcat. Cambiemos el nombre (con "`mv`") para eliminar el

numero de versión:

```
sudo mv /usr/local/apache-tomcat-4.1.36 /usr/local/apache-tomcat
```

De ahora en adelante nos referiremos a esa ruta como ***{apache_tomcat.home}*** en el documento.

{***apache_tomcat.home***}= /usr/local/apache-tomcat

Preparación del Usuario y el Entorno

Debemos exportar tres variables de entorno para poder hacer uso cómodamente de esta aplicación. La primera es *JAVA_HOME* que hemos exportado antes, las otras dos son `APACHE_TOMCAT` y `CATALINA_HOME`. Además agregaremos la ruta del programa a las rutas del sistema:

```
sudo echo "APACHE_TOMCAT={apache_tomcat.home} >> /etc/profile.local
sudo echo "CATALINA_HOME={apache_tomcat.home} >> /etc/profile.local
sudo echo "{apache_tomcat.home}/lib >> /etc/ld.so.conf
sudo ldconfig
```

Nuevamente, conviene hacer *logout* y *login* para que se recarguen las rutas del shell.

Agregaremos una cuenta de usuario que va a ser la responsable de gestionar Tomcat. De esta forma evitamos a Tomcat tener acceso a privilegios de nivel superior si fuéramos a ejecutarlo con una cuenta de administración, o como 'root'. El primer paso es crear una nueva cuenta de usuario (las opciones cambian de distribución en distribución) y asignarla como propietario de la instalación de Tomcat:

```
sudo useradd tomcat
sudo chown -R tomcat {apache_tomcat.home}
```

Y luego necesitamos crear un script que gestione el inicio y parada del servicio de Tomcat. Este script es ejecutado por 'root' durante el inicio y parada del sistema, y ejecuta tomcat cediéndole los permisos de ejecución al usuario que hemos creado. El script puede ser encontrado en el Anexo I y se guarda bajo `/etc/init.d/tomcat`.

Iniciando y Deteniendo el Servidor

Estos son comandos de sistema, por lo cual deben ser ejeuctados desde una cuenta con privilegios administrativos y usando '`sudo`':

Iniciar:
`/etc/init.d/tomcat start`

Detener:
```
/etc/init.d/tomcat stop
```

Eclipse

Descargando e Instalando

(Por Completar)

Jakarta Struts

II.- Descargando e Instalando

Para instalar **Jakarta Struts**, se descarga del sitio de Apache Tomcat la versión escogida que es la 1.2 (en estricto rigor). Luego se descromprime en un directorio de tal manera que construya su propia entrada de directorio, en nuestro caso, en **$HOME**`/plataformas/struts`.

```
[lmachuca@server ~ ]> mkdir plataformas; cd plataformas;
[lmachuca@server ~/platafromas]> unzip struts-1.3.8.zip
```

No es necesario completar variables de entorno, pero sí es necesario mantener nota de la ubicación donde hemos descomprimido los archivos porque el servidor Tomcat y Eclipse necesitarán referirse a ellos después. Por tanto, asignaremos el nombre *{struts.archivo}* a la ruta que hemos indicado en el párrafo anterior, a lo largo de este documento.

Nota referente a EasyStruts for Eclipse

Vinculando con Apache Tomcat

(Por Completar)
- Crear los vínculos simbólicos
- Crear la estructura de directorios modelo
-

Eclipse Europa + JDK

(Por Completar):

- Instalar Eclipse
- Instalar Sysdo Plugin for Tomcat
- Instalar StrutsStudio
-

III.- Comprobando que Todo Está Bien

Para comprobar que todo está bien, la forma más adecuada es simplemente intentar un "*Hola Mundo*" en Struts y hacerlo correr bajo Apache-Tomcat.

"*Hola Mundo" en Struts*

1. Detener el servidor Apache Tomcat.
2. Tomar la carpeta de struts que hemos descomprimido en {*struts.archivo*}.
3. La carpeta tiene varios ejemplos; escogeremos uno (*ch03app*) y lo copiamos al directorio de aplicaciones web *{apache_tomcat.apps}*, tomando el cuidado de cambiar los permisos para que el servidor tomcat pueda ejecutarlos. Esto significa cambiar el propietario al usuario "`tomcat`".

```
[lmachuca@servidor {struts.archivo}]> su tomcat -c "cp -R ch03app
{apache_tomcat.apps}"
```

Otra forma de hacerlo (en dos pasos) sería:

```
[lmachuca@server {struts.archivo}]> sudo cp -R ch03app {apache_tomcat.apps}
[lmachuca@server {struts.archivo}]> sudo chown tomcat -R
{apache_tomcat.apps}/ch03app
```

4. Hacer partir el servidor Apache Tomcat con las instrucciones en "Iniciando y Deteniendo el Servidor".
5. Navegar a la dirección provista por Apache Tomcat (por defecto: la URL del servidor con puerto 8080, p.ej.: `http://informatica.temuco.udelmar.cl:8080/`)
6.

Si todo ha salido bien, lo que debemos ver en nuestro Navegador Web sería una pantalla como la siguiente:

Igualmente, si revisamos los mensajes de acceso al Servidor nos podremos encontrar con algo parecido a lo siguiente:

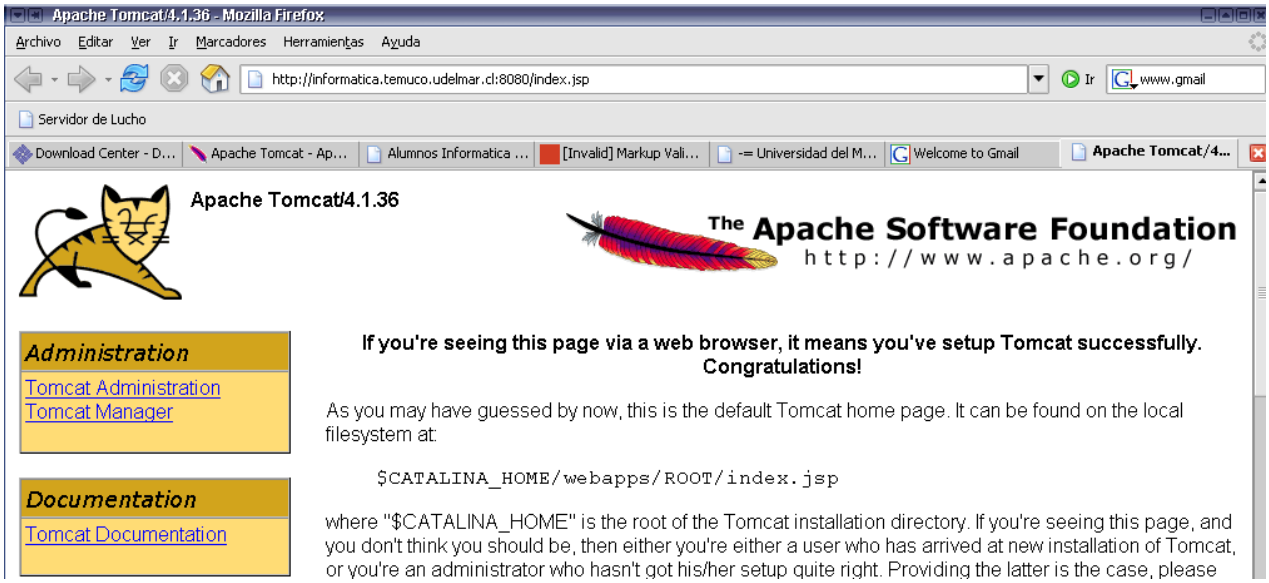

Imagen 4: Pantalla de Bienvenida de Apache Tomcat en la Universidad del Mar

IV.- Creando una Aplicación Web

A continuación, tomaremos los primeros pasos para crear una aplicación web básica, y realizaremos las modificaciones para que esta aplicación sea desarrollada desde `Eclipse` con Struts.

El primer paso es crear la estructura de archivos de la aplicación web, la cual sigue la misma forma que las aplicaciones web de Tomcat.

La estructura de las aplicaciones web es bastante sencilla: creamos una carpeta, la cual en este caso particular se llama "luchoapp", pero, en general nos referiremos a ella como *{miapp.carpeta}*. Luego creamos la siguiente estructura de directorios:

{miapp.carpeta}

- WEB-INF
  - classes
  - lib
- pages

A continuación, creamos el archivo `web.xml`, el cual contiene la configuración base de nuestra aplicación. Este archivo lo podemos copiar de cualquier otra aplicación disponible en el servidor, pero lo más adecuado es tomar el archivo de ejemplo provisto con Struts:

```
cp {struts.carpeta}/apps/struts-blank/web.xml {miapp.carpeta}
```

En la carpeta `WEB-INF/lib`, van los archivos JAVA (*.jar*) que componen la aplicación y que no estén o difieran respecto de la instalación estándar de Tomcat. Ejemplo: nuestra aplicación, conectores a bases de datos, librerias importadas, etc. Estos archivos deben ser importados de forma apropiada, por ejemplo con un paquete *.war*, y después de hacer cualquier modificación en este directorio es necesario reiniciar Tomcat para que reconozca los cambios.

Finalmente, para hacer que el servidor Tomcat reconozca nuestra aplicación es necesario copiar esta carpeta dentro del directorio de aplicaciones web de Tomcat, es decir, de *{apache_tomcat.apps}*.

Iniciando el Desarrollo

aa

Anexos

Anexo I — Script de Servicio Tomcat <tomcat >

El siguiente script ha sido publicado por **Stefan Gybas**, de una contribución original de Miquel van Smoorenburg quien trabajó en el desarrollo de utilidades para Tomcat. Este script ha sido ligeramente modificado para nuestra versión de Ubuntu y han sido agregadas y configuradas las variables que se refieren a nuestra instalación particular.

Para que este script funcione debe estar bajo `/etc/init.d` y con permisos de ejecución. Para agregarlo como servicio del sistema puee usarse "`chkconfig`" o "`services`" dependiendo de la distribución.

```
#! /bin/sh -e
#
# /etc/init.d/tomcat -- startup script for the Tomcat 5.0 servlet engine
#
# Written by Miquel van Smoorenburg .
# Modified for Debian GNU/Linux by Ian Murdock .
# Modified for Tomcat by Stefan Gybas .

#PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# directorio base de Tomcat
TOMCAT_HOME=/usr/local/apache-tomcat
CATALINA_HOME=/usr/local/apache-tomcat
# usuario que corre el servidor
TOMCAT_USER=tomcat
TOMCAT_APPS=$CATALINA_HOME/webapps
# scripts de inicio y parada de la distribucion
TOMCAT_STARTUP=$TOMCAT_HOME/bin/startup.sh
TOMCAT_SHUTDOWN=$TOMCAT_HOME/bin/shutdown.sh
# identificacion del servicio
TOMCAT_NAME="tomcat"
TOMCAT_DESC="Motor Tomcat"

# instalacion de Java
JAVA_HOME=/opt/java/j2sdk
export JAVA_HOME

# Otras variables para uso del gestor de servicios
TOMCAT_PIDFILE="/var/run/$NAME-local.pid"
TOMCAT_LOGFILE="$CATALINA_HOME/run.log"
export CATALINA_PID="$PIDFILE"

# vemos con que parametros somos invocados
# start (iniciar el servicio)
# stop (detener el servicio)
case "$1" in
  start)
        echo -n "Starting $TOMCAT_DESC:"
        #if start-stop-daemon --test --start --pidfile "$PIDFILE" \
        #        --user $TOMCAT_USER --startas "$DAEMON" >/dev/null; then
        #        # -p preserves the environment (for $JAVA_HOME etc.)
        #        # -s is required because tomcat5's login shell is /bin/false

       # usamos "su" para iniciar como el usuario tomcat
       su -p -s /bin/sh $TOMCAT_USER \
         -c "$TOMCAT_STARTUP" >> $TOMCAT_LOGFILE 2>&1
```

```
        sleep 1
        # chequeo visual que el proceso ha iniciado
        ps -ef | grep $TOMCAT_STARTUP
        echo "$NAME."
        ;;
  stop)
        echo -n "Stopping $TOMCATDESC: "
        #if start-stop-daemon --test --start --pidfile "$PIDFILE" \
        #        --user $TOMCAT5_USER --startas "$DAEMON" >/dev/null; then
        #        echo "(not running)."
        #else
        #
        # usamos "su" para detener el proceso
        su -p -s /bin/sh $TOMCAT_USER \
          -c "$TOMCAT_SHUTDOWN" >/dev/null 2>&1 || true
        rm -f "$PIDFILE"
        echo "$NAME."
        ;;
  restart|force-reload)
        $0 stop
        sleep 1
        $0 start
        ;;
  *)
        echo "Usage: /etc/init.d/tomcat {start|stop|restart|force-reload}" >&2
        exit 1
        ;;
esac

exit 0
```

Anexo II — Configuración de la aplicación de Struts <web.xml>

(Por publicar)

Anexo III — Falencias y Requerimientos de Operación

Aún cuando Java es un buen lenguaje de programación, sufre de dos malestares que me han afectado particularmente: está tremendamente sobrecargado, y sus complementos están complejamente enrevesados. Una situación que es más o menos análoga a los “infiernos de dependencias” en Linux. Lamentablemente, esto es producto de su evolución reciente: “Java 1”, la primera versión, no era nada pesada y podía ser ejecutada sin mayores problemas en los PC estándar de la época.

El hecho que tanto Tomcat como Eclipse requieran Java, me ha significado una limitante de importancia para continuar el proyecto. En condiciones normales he tenido acceso a tres máquinas:

- Una **máquin en casa**, AMD Sempron 2.4 GHz y 256 MiB de RAM. El espacio en disco no es problema (200 + 320 GiB). No cuenta con acceso a Internet.
- Un **notebook** Olidata refaccionado. VIA Nemeiah 1.6 GHz y 256 MiB de RAM. Dado que es un notebook, tremendamente limitado para desarrollar en Java. Es usualmente el sistema que utilizo en mis cursos de informática en la sede, exceptuando el curso actual de Taller WWW-2 debido a estas limitaciones.

- El **servidor de alumnos de Informática**. Buena máquina (Intel Pentium IV 2GHz, 384 MiB) pero no destinada al desarrollo.

En las tres máquinas puedo correr sin problemas aplicaciones simples de Java. Sin embargo, pasando a un SDK o a un entorno de desarrollo las cosas se complican drásticamente, dada la poca memoria RAM disponible.

Pese a que la máquina en casa usa un escritorio Linux liviano, al partir con Eclipse para Java el rendimiento cae a un nivel tan bajo que la operación normal se vuelve casi indistinguible de un sistema infectado o en modo "zombie". Con Eclipse para C++ nunca tuve estos problemas, aunque tampoco era ligero como una pluma, y podía generar y compilar proyectos.

En el notebook la situación es más drástica, con Eclipse quedando completamente pegado al tratar siquiera de cargas sus componentes básicos. Tampoco puedo correr aplicaciones en Java sencillas sin degradar el rendimiento del sistema, tema no menor en un notebook, aunque la degradación no es muy severa.

El servidor de nuestro proyecto puede correr Eclipse para Java con algo más de facilidad, pero aún así, esto es al precio de mantener el sistema sobrecargado (necesidad de Java, entorno gráfico, Eclipse y sus complementos), en circunstancias que su tarea principal no es ser una consola de desarrollo sino un servidor web. Dado que no nos podemos preciar esa baja de rendimiento ni la sobrecarga (no servimos sólo a nosotros, sino a otros proyectos como la Revista Microverso), no es posible continuar el desarrollo en esta máquina, pese a sus recursos.

Lamentablemente, la única forma de progresar más allá de la teoría con este proyecto es que la Universidad ponga los medios necesarios para continuar. Existen dos opciones: pedir a la Universidad los recursos para actualizar mi sistema, o pedir que se disponga una estación en la sede con todos los recursos necesarios. La primera opción sería mucho más viable para mí ya que, por ejemplo, si la Universidad pudiera donarme una memoria RAM extra, podría aumentar la capacidad del sistema y trabajar desde casa, sin sobrecargar el entorno del servidor trabajando localmente, y sin depender de la disponibilidad de un equipo en laboratorios en la sede, donde usualmente, se ve a los alumnos chateando con Messenger o usando las máquinas para jugar Battlefield II (que no es un mal juego); según ellos, "*eso*" es Informática.

En cuanto a los recursos, teorizo que bastaría con una expansión de la memoria RAM de 256 a 512 MiB. Esto se lograría con la simple compra de un módulo de 256 MiB DDR PC3200, de los cuales supongo que ya no quedarán muchos en el mercado, siendo los más fáciles de encontrar los de 512 MiB. En consecuencia, bastaría con que la Universidad pudiera colaborar con **parte del costo** que bordea los CL$30000. Tal ayuda económica debiera ser vista como una inversión a largo plazo en el futuro del alumno, dado que le permite la capacidad necesaria para poder adaptarse y utilizar tecnologías más recientes, cada vez más hambrientas de recursos a menudo innecesariamente; pudiendo así aprender y mejorar su currículo, y eventualmente optando a un trabajo más realista (*¿qué empresa está desarrollando o sirviendo ahora aplicaciones de consola con C++*?), que le permita acceder a una buena calidad de vida, mayor que la del ciudadano del Tercer Mundo promedio.

En circunstancia, desde una perspectiva de evaluación de proyectos, estimo el costo de continuación de este proyecto en la suma de CL$30000, liquidos, al valor del dólar el 27 de Septiembre de 2007. Además, tal inversión tendría que ser realizada antes que suba el precio del dólar y, en consecuencia, las partes y piezas para PC. Una petición al Sr Marcos Almonacid, rector de nuestra Sede, ya ha sido emitida en la búsqueda de este apoyo económico.

Conclusiones

En este primer proceso de desarrollo y en la preparación de este informe me han quedado claros varios elementos, paradigmas y decisiones que se han tomado a lo largo de los años.

El proyecto en sí es muy interesante desde el punto de vista funcional, ya que me permite probar directamente una tecnología nueva para mí (no he pasado por Java desde al menos el año 2003) en un ambiente controlado (tenemos el servidor para nosotros) y para realizar una tarea menos que trivial.

Estoy consciente que muchos tutoriales y manuales para este tipo de desarrollos, vienen, por necesidad de diseño, enfocados a sistemas Windows. No es menos cierto que la preparación de un sistema como éste en Linux requiere mucho más que "*Siguiente, Siguiente...*". Y tampoco es menos cierto que la documentación de procesos es un área en la cual tengo experiencia. De modo que hasta ahora el proyecto me ha venido como "anillo al dedo" para poner al día mis capacidades.

El hecho de estar usando Java se ha convertido en la principal (y esencialmente la única) limitante. Cuando probé Java el año 2003, era un lenguaje sencillo y liviano, y no tuve problemas para desarrollar una aplicación un poco menos que simple. Sin embargo, estamos en el año 2007, donde Java se ha convertido en un *sistema tan sobrecargado* que aún con un computador medianamente decente (procesador de 2GHz y 256 MiB de RAM), no me es posible hacer convivir el desarrollo en Java con otras aplicaciones que corran a la vez; noi siquiera me es posible correr las más simples aplicaciones de Java sin sufrir un deterioro serio de la funcionalidad de mi equipo.

Me vienen a la mente los comentarios de Exel Silva, mi compañero en el proyecto de servidor de alumnos de Informática, en cuanto a que Java se ha convertido en un infierno. Esto daña particularmente mi capacidad de avanzar en este proyecto en cuanto a que dos de las aplicaciones usadas en él (Tomcat y Eclipse) están basadas en Java. Considerando que yo vengo de la *practicabilidad, elegancia y compatibilidad* de C++, ver que Java hace lo mismo pero mucho más pesado y con mucho más riesgo y daño para mis sistemas, me ha hecho recordar por qué lo abandoné la primera vez: Java es un lenguaje de programación genial, muy adaptable, y es más que perfectamente usable, pero no en éste, el *Tercer Mundo*, esencialmente una colonia gringa que a lo más, recibe la basura electrónica e informática del Primer Mundo como sistemas de primera línea para su población.

Apache Tomcat ha sido un nuevo paradigma a tratar en este proyecto. Hasta la fecha, sólo había trabajado con servidores web (sistemas LAMP) o con aplicaciones de servicios punto a punto (programas usando sockets o puertos para usar simples comandos). Tomcat me parece una excelente nueva forma de servir aplicaciones al mundo, sirviendo ya no simplemente contenido dinámico, sino "*comportamiento dinámico*", capacidades de programación real para llevar a cabo tareas complejas vía web. Además enfrentarme al uso de Tomcat desde su instalación y configuración ha sido particularmente agradable por la documentación existente y las facilidades en nuestro servidor.

Eclipse, la plataforma de desarrollo integrado, ha sido, por mucho, el mejor plus en el tratamiento de este proyecto. Tengo escasa experiencia en Eclipse para C++ y me asombra el nivel de "humanidad" al que se llevan las opciones y acciones que provee este programa. La disponibilidad, la facilidad de uso, una vez configurado, y la extensibilidad simplemente asombrosa. La base de usuarios y colaboradores que provee también es importante. Desafortunadamente, al tratar con Eclipse para Java nuevamente me vi cercado por las capacidades de mis actuales sistemas, con una máquina que llegaba casi al punto del *halting* al tratar de cargar todos los elementos que requiere la plataforma de desarrollo para Java. ¡Ni pensar entonces en agregar un módulo para trabajar con Tomcat o Struts! ¡Y ni menos cambiarme a una plataforma tan pesada como un CD completo, como NetBeans!

Struts ha sido algo completamente nuevo e interesante, definitivamente lo más novedoso para mí en estos dos años. Desde mi tiempos como programador en C++ y sirviendo como administrador subrogante en la Universidad de La Frontera, mi enfoque a la programación y el servicio de aplicaciones siempre ha sido el de simplemente "cliente-servidor": una aplicación hacía el trabajo, y la otra le hacía encargos. Pero Struts promueve una vista fundamentalmente distinta: la MVC, donde ya la comunicación entre estos actores está aislada y por tanto la interoperabilidad es más segura de desarrollar. Como siempre, agregar un *nivel extra de abstracción* para mejorar las capacidades de un sistema desde una perspectiva macro.

Desde el punto de vista de aprendizaje, entonces, estoy muy contento con las cosas que este proyecto me ha planteado (en particular Tomcat). Pero de momento el ser un simple ciudadano del Tercer Mundo me impide llevar el desarrollo de forma permanente y siquiera a un buen ritmo, debiendo adaptarme con el paradigma de "*no poder caminar y comer chicle a la vez*". Razón por la cual he establecido un anexo con las capacidades necesarias para poder continuar operando, en la esperanza de poder obtener de alguna forma estos recursos. De lo contrario, en las condiciones actuales, el desarrollo del proyecto tendrá que avanzar a un paso cancino que me impedirá continuarlo en casa, donde los requisitos de mi familia son otros.