Plataforma de Control en el Ámbito de la Televisión Digital Interactiva.

LENIN DAVID GÓMEZ MERA.
ANDRÉS GUERRERO ROSERO.

Anexos

Universidad del Cauca.
Facultad de Ingeniería Electrónica y Telecomunicaciones.
Departamento de Telemática.
Servicios Avanzados de Telecomunicaciones.
Popayán, Septiembre de 2010.

TABLA DE CONTENIDO

ANEXO A. MODELO DE APLICACIÓN *DVB-J*

Una aplicación *DVB-J* o simplemente *Xlet*, son aplicaciones programadas en lenguaje *Java* para el entorno de televisión sobre la interfaz *API MHP*, básicamente son un conjunto de clases que son trasmitidas junto a un servicio de televisión. Estas aplicaciones, a diferencia de las convencionales, pueden estar varias ejecutándose al mismo tiempo y no tienen que iniciarse a través de línea de comandos, muy similar al funcionamiento de los *applets* [1] [2].

La interfaz *Xlet* permite a una fuente externa (como lo puede ser un Navegador de Internet en el caso del los *applets* en el mundo web, para el mundo de la televisión es el gestor de aplicaciones (*application manager*[1]) incluido en el receptor de televisión digital) iniciar y terminar las aplicaciones, al igual que hacer el control de otros estados propios del *Xlet* [2].

Una de las características que incorpora los *Xlets*, es la capacidad que tiene el gestor de aplicaciones de pausar y resumir la ejecución de estas, ya que por encontrarse en un entorno de escasos recursos, solo una aplicación puede ser visible a la vez y las demás deben estar pausadas para dejar disponibles los recursos a la aplicación que los solicite [2].

El flujo de vida de un *Xlet* es bastante sencillo, como se puede observar en la siguiente figura [60]:

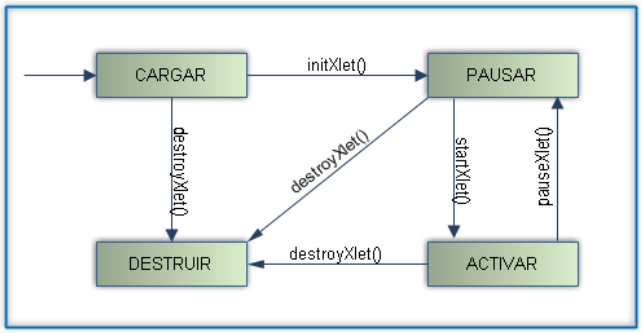

Fig.A.1. Ciclo de vida de una aplicación Xlet, tomado de [1].

La interfaz que debe implementar está en total concordancia con ciclo de vida del Xlet [2]:

```
Public interface Xlet
{
public void initXlet(XletContext ctx)
throws XletStateChangeException;

public void startXlet()
throws XletStateChangeException;

public void pauseXlet();
public void destroyXlet(boolean unconditional)
throws XletStateChangeException;
}
```

[1] *Application Manager*: Se encarga de manejar los estados de cada Aplicación y hacer de puente a los recursos del Set-top Box

Una aplicación entra en el estado *Loaded* cuando es cargada bien sea desde *Set-top Box* o bien desde el Carrusel de datos y se hace una invocación de la clase constructora para crear una instancia de la clase principal del *Xlet* por parte del gestor de aplicaciones en el receptor de television digital. El *Xlet* puede ser iniciado de tres formas [2]:

- Por indicación directa del usuario,
- Invocado por otro *Xlet*,
- Automático por configuración en la Tabla de información de las aplicaciones *(AIT*).

Cuando sucede esto, el gestor de aplicaciones del receptor invoca al método *initXlet(),* pasando un nuevo objeto de tipo *XletContext*[2] para el *Xlet* . El *Xlet* puede usar este *XletContext* para inicializarse así mismo, y recargar cualquier cantidad de recursos como imágenes que pueden requerir cierto tiempo de carga desde el carrusel de objetos. Cuando la inicialización está completa, el estado del *Xlet* está en *Paused* y esperando para iniciarse inmediatamente [2].

Una vez el método *initXlet()* se ejecute, el gestor de aplicaciones hace un llamado al método *startXlet(),* este cambia el estado del *Xlet* de *Paused* a *Active* e inmediatamente el *Xlet* estará disponible para interactuar con el usuario [2].

Durante la ejecución del *Xlet*, el gestor de aplicaciones puede llamar al método *pauseXlet()*, esto hará que la aplicación cambie de el estado *Active* que se encuentra a *Paused*. La aplicación puede volver al estado *Active* llamando al método *startXlet()* nuevamente. Esto sucede muchas veces durante el ciclo de vida del *Xlet* [2].

Al final del ciclo de vida del *Xlet*, el gestor de aplicaciones llamara al método *destroyXlet()*, el cual llevara al *Xlet* al estado *Destoy* e inmediatamente libera todos los recursos previamente asignados. Después de este punto, esta instancia del *Xlet* no puede volver ser iniciada [2].

Para concluir, es de suma importancia tener claro que un *Xlet* no es una aplicación Java convencional, se acerca más conceptualmente a lo que significa un *applet*. Como un *applet*, hay más de un *Xlet* ejecutándose al tiempo, lo que indica que estos no deberían invocar ciertas acciones que afectarían de manera global al funcionamiento de la maquina virtual *Java*, como por ejemplo: un *Xlet* nunca debería llamar al método `System.exit()`, ya que no es solamente esa aplicación que esta ejecutándose sobre esa máquina virtual [2].

Bibliografía

[1]. **B. CUBAS.** (2008). HTML: "*Articulo: Introducción a los Xlets*". [En línea]. Disponible en: http://www.cesnavarra.net/cesdigital/Lists/Noticias%20CESDigital/DispFormCES.aspx?List=5ec0dfc7-7911-470b-8b6b-71ba72783fdd&ID=23 [Consulta: Enero 2010].

[2]. ***S. MORRIS.*** (2008). HTML: "*An Introduction To Xlets*". [En línea]. Disponible en: http://www.interactivetvweb.org/tutorials/mhp/xlet_introduction [Consulta: Enero 2010].

[2] XletContext: Actúa como puente entre el Xlet y el Application Manager. Permite que el Application Manager sea correctamente notificado de cualquier cambio en los estados de los Xlet's.

ANEXO B. DESCRIPCIÓN DEL PROCEDIMIENTO DE REGISTRO DE USUARIO EN *IMS*

El nivel de autenticación de *IMS* es básicamente el procedimiento por el cual la red autentica al usuario y posteriormente lo autoriza para acceder a los servicios *IMS*.

El registro IMS permite al terminal de usuario disponer de los servicios ofrecidos por *IMS* siempre y cuando exista soporte para una conexión *IP* y descubrir un punto de entrada a *IMS* (por ejemplo *P-CSCF*).

El proceso de registro contiene dos fases de acuerdo como se observa en las figuras B.1 y B.2.

- La primera fase – cómo la red desafía al equipo terminal de usuario.
- La segunda fase – cómo el terminal de usuario responde al desafío y completa el registro.

Primera fase:

1. El terminal de usuario envía una petición *SIP REGISTER* al *P-CSCF* descubierto. Esta petición puede contener por ejemplo, una identidad para ser registrada y un nombre de dominio del home *(dirección del I-CSCF).*
2. El *P-CSCF procesa* la peticion de registro y usa la dirección de dominio del home provista para resolver la dirección *IP* del *I-CSCF.*
3. El *I-CSCF,* a su vez, contacta al *HSS (Home Subscriber Server)* para encontrar al *S-CSCF* requerido.
4. Después de la selección del *S-CSCF*, el *I-CSCF* reenvía la petición de registro al *S-CSCF*.
5. El *S-CSCF* verifica que el usuario no es autorizado y, por lo tanto, devuelve automáticamente información de autenticación desde el HSS
6. y desafía al usuario con una respuesta *401 Unauthorized.*

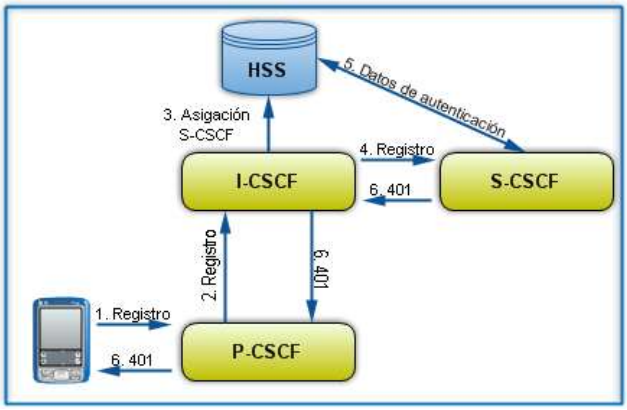

Fig.B.1 Primera fase de registro en IMS tomado de [1].

Segunda fase:

7. El equipo terminal calcula una respuesta al desafío y envía otra petición de registro al *P-CSCF*.

8. De nuevo, el *P-CSCF* encuentra al *I-CSCF*

9. y el *I-CSCF* de vuelta, encuentra al *S-CSCF*.

10. Finalmente, el *S-CSCF* verifica la respuesta

11. y, si esta es correcta, descarga información del perfil del usuario desde el *HSS*

12. y acepta el registro con una respuesta *200 OK.*

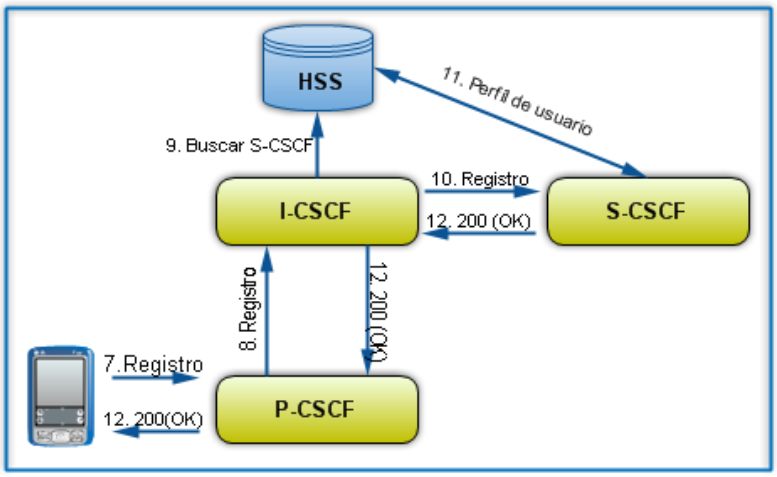

Fig.B.2 Segunda fase de registro en IMS tomado de [1].

Una vez el terminal de usuario se encuentre exitosamente autorizado, podrá iniciar y recibir sesiones. Durante el procedimiento de registro ambos, el terminal de usuario y el *P-CSCF* aprenden del *S-CSCF* de la red al cual está sirviendo al terminal de usuario.

Es responsabilidad del terminal de usuario mantener el registro activo y periódicamente actualizarlo. Si el terminal de usuario no actualiza el registro, el *S-CSCF* removerá este registro cuando el lapso de tiempo en el temporizador de registro sea demasiado largo. Cuando el terminal de usuario quiere terminar el registro en *IMS*, fija el temporizador de registro a 0 y envía una petición *REGISTER.*

Bibliografía

[1]. **D. BUSTOS.** (----). HTML: "*IMS Tutorial Examples based in SDS Ericsson tool Basic level course*". [En línea]. Disponible en: http://www.docstoc.com/docs/16501893/IMS-tutorial-sds-ericsson-examples/ [Consulta: Marzo 2010].

ANEXO C. CONFIGURACIÓN DE LA HERRAMIENTA *SDS* DE *ERICSSON*

C.1 Requisitos

Para la instalación de la solución *SDS* de *Ericsson*, se requiere tener instalado el sistema operativo ***Windows XP Professional Edition*** además de otras aplicaciones que pueden obtenerse de manera gratuita en Internet, por lo que se recomienda tener una conexión a internet al momento de la instalación de la herramienta y seguir detalladamente los pasos que se describen a continuación.

Inicialmente la herramienta *SDS* de *Ericsson* se obtiene del siguiente enlace: http://www.ericsson.com/developer/sub/open/technologies/ims_poc/tools/sds_40, donde se descarga la versión 4.1.

C.1.1 Requisitos Hardware

La computadora a utilizar, en donde se instala tanto el sistema operativo Windows XP como la herramienta *SDS* de *Ericsson* debe cumplir como mínimo con los siguientes requisitos:

Requisito	Mínimo Recomendado
Procesador	AMD 64 Athlon 3500+ o superior
Memoria RAM	1 Gb de capacidad o superior.
Espacio Libre en Disco Duro	De 2 a 3Gb de espacio
Red	Adaptador de Red 10/100 Mbps o 10/100/1000 Mbps
Tarjeta de Sonido	Estéreo.

Tabla C. 1 Requisitos Hardware.

C:2.2 Requisitos Software

Requisito	Mínimo Recomendado
Sistema Operativo	*Windows XP Professional SP3 32Bits, Windows Vista 32Bits (las versiones de 64Bits no son oficialmente soportadas y no fueron probadas)*
SDS	*4.1 FD1* únicamente *(4.2 no soporta desarrollo móvil)*
Java SDK	JDK 1.6_10 o superior.
DirectX	9.0c (manejo de video en simulación)
Visual C	*Visual C Redistributable 2005 (mantiene a IMS como servicio de Windows)*
Gestor de Base de Datos	*PostgreSQL 8.4*

Tabla C. 2 Requisitos Software.

Adicionalmente, al momento de ejecutar el asistente de instalación del *SDS* de *Ericsson*, se recomienda cerrar todas las aplicaciones que se encuentran ejecutándose sobre la maquina.

C.2 Instalación de *SDS 4.1 FD1*

Se ejecuta el instalador autoejecutable de *SDS* en modo Administrador, este instalador ofrece todas las aplicaciones necesarias para el funcionamiento de la plataforma *IMS*.

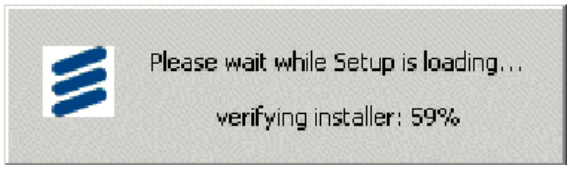

Figura C. 1 Instalación de la solución SDS de Ericsson.

Se muestra la ventana de bienvenida para la instalación de la solución.

Figura C. 2 Instalación de la solución SDS de Ericsson.

Si no se tiene maquina virtual *Java* instalada, el asistente intentara descargar el instalador *JDK 6 update 10* desde Internet y automáticamente la instalara, ya que es un requisito fundamental para la ejecución de la solución.

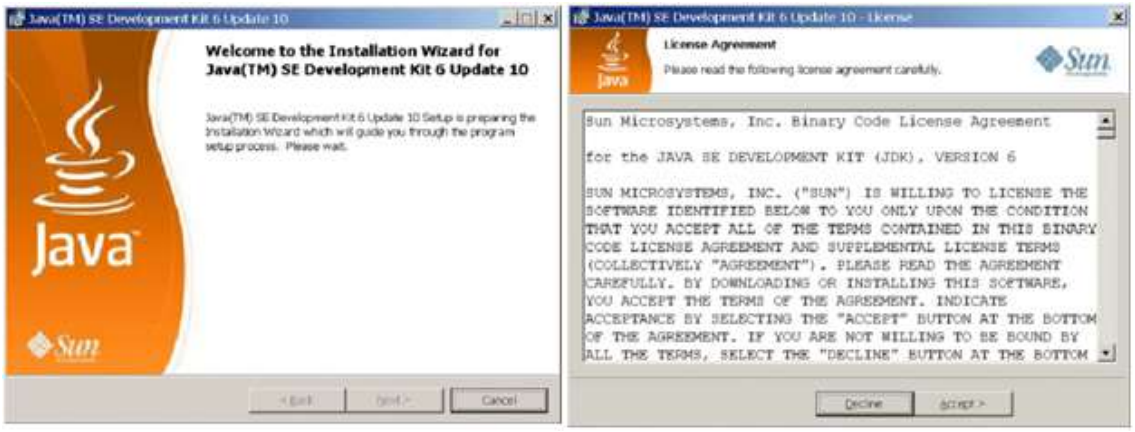

Figura C. 3 Instalación de la Maquina Virtual Java.

Luego de instalar la maquina virtual Java, se continúa con la ejecución del asistente de *SDS*, donde se hace una serie de preguntas como se muestra a continuación:

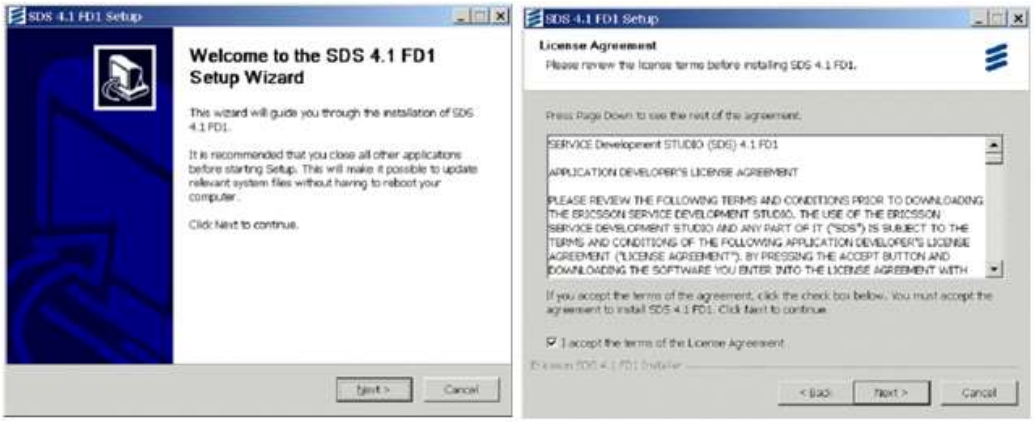

Figura C. 4 Instalación de la solución SDS de Ericsson.

Para la grafica C.5, se debe introducir la clave de activación del producto. Esta clave de prueba es proporcionada por *Ericsson* y llega a la cuenta de correo electrónico de quien descargo el instalador, ya que se tuvo que hacer con anterioridad una subscripción para poder descargar el archivo.

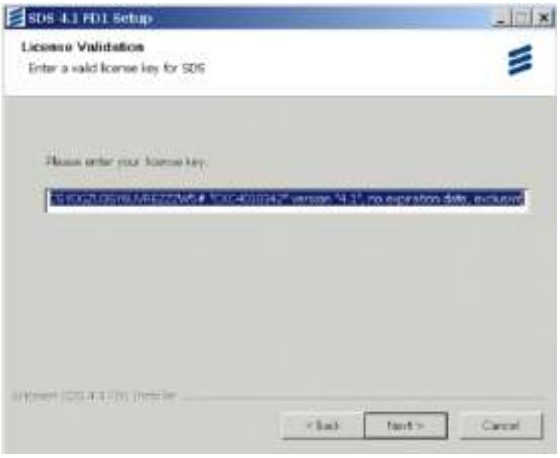

Figura C. 5 Instalación de la solución SDS de Ericsson.

Una vez se tiene el producto activado, se procede a seleccionar todos los componentes y la ruta de instalación (preferiblemente se deja por defecto). No se debe dejar espacios en los nombres de las carpetas o del sistema y se debe proporcionar nombres cortos que no superen el límite permitido para algunos formatos ya que pueden existir fallas debido al comportamiento distinto de estos símbolos en diferentes sistemas operativos.

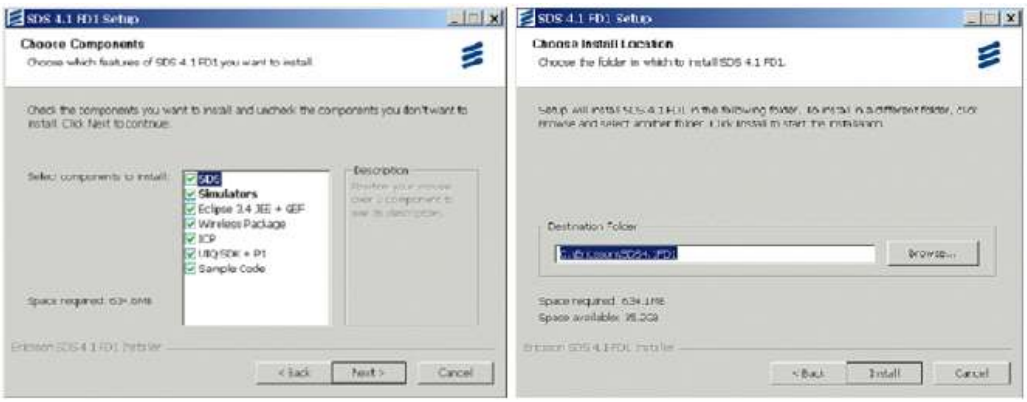

Figura C. 6 Instalación de la solución SDS de Ericsson.

El instalador extrae el contenido en una carpeta temporal con multitarea ocupando mucho del procesamiento de la *CPU*, por esta razón es importante tener cerrado cualquier programa residente en memoria o antivirus que pueda por error truncar alguno de los muchos paquetes que se descomprimen (Este error se presento en este proyecto debido a que el antivirus sospecha de los archivos empaquetados, y mientras escaneaba el proceso de instalación no permitió el acceso de algunos archivos necesarios, como resultado se tiene una instalación corrupta con la falta de archivos necesarios para la ejecución de los diferentes servicios).

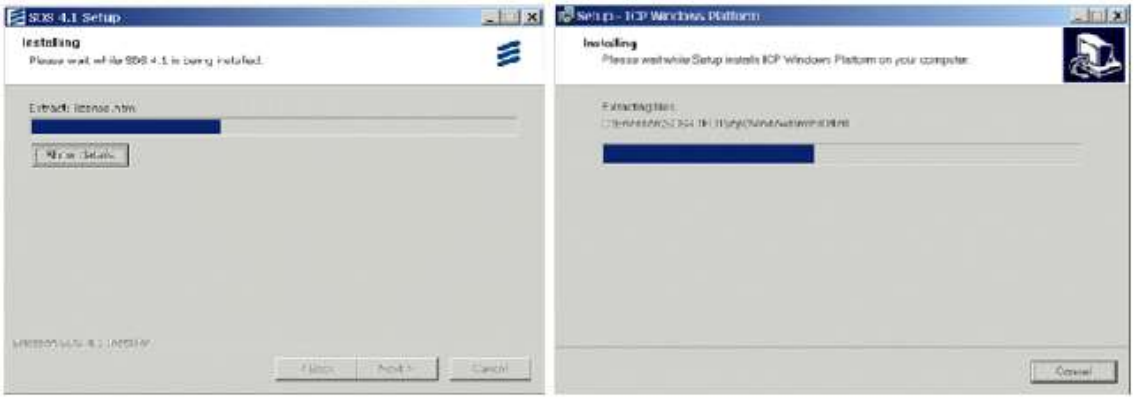

Figura C. 7 Instalación de la solución SDS de Ericsson.

Cuando termina la extracción de los archivos, se procede a la instalación de la plataforma de desarrollo *UIQ* y sus componentes para el desarrollo de aplicaciones sobre dispositivos móviles.

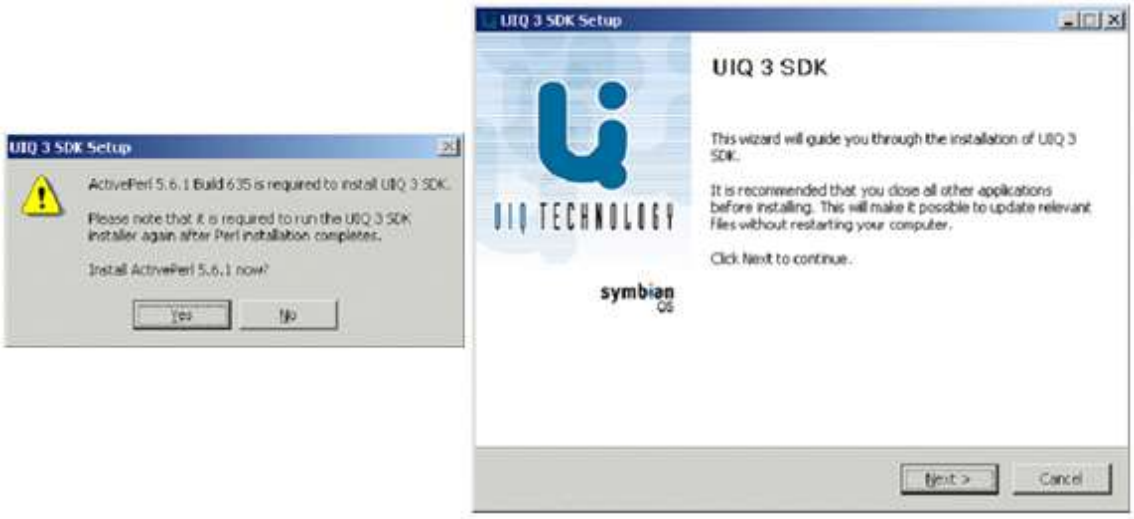

Figura C. 8 Instalación de la solución SDS de Ericsson.

Se establece a este *SDK* como el entorno principal para tenerse en cuenta dentro de la simulación de entornos móviles.

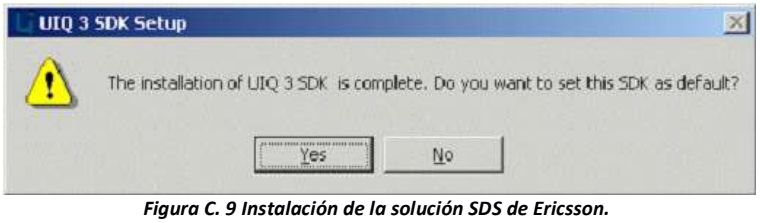

Figura C. 9 Instalación de la solución SDS de Ericsson.

Se instalan las aplicaciones necesarias para la ejecución del *UIQ SDK.*

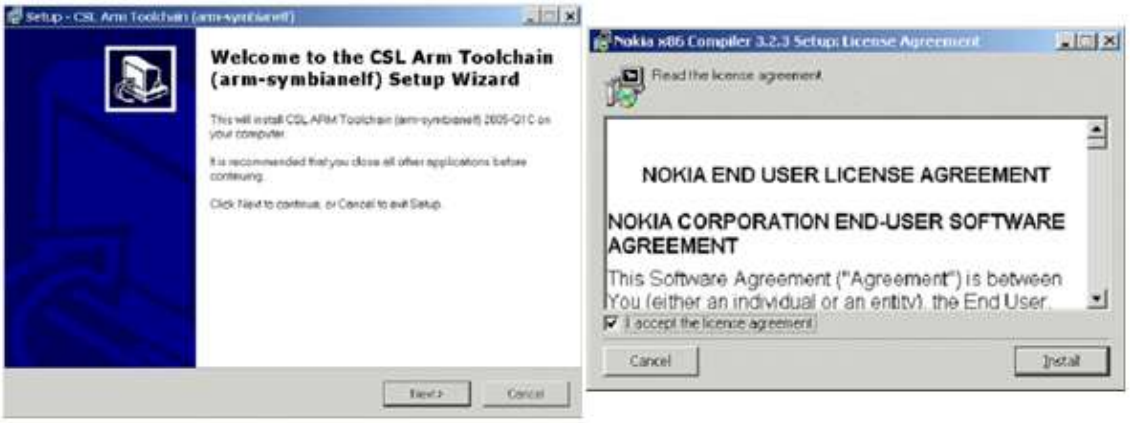

Figura C. 10 Instalación de la solución SDS de Ericsson.

Se completa la instalación y aparece automáticamente la pantalla de configuración del *UIQ 3 SDK.*

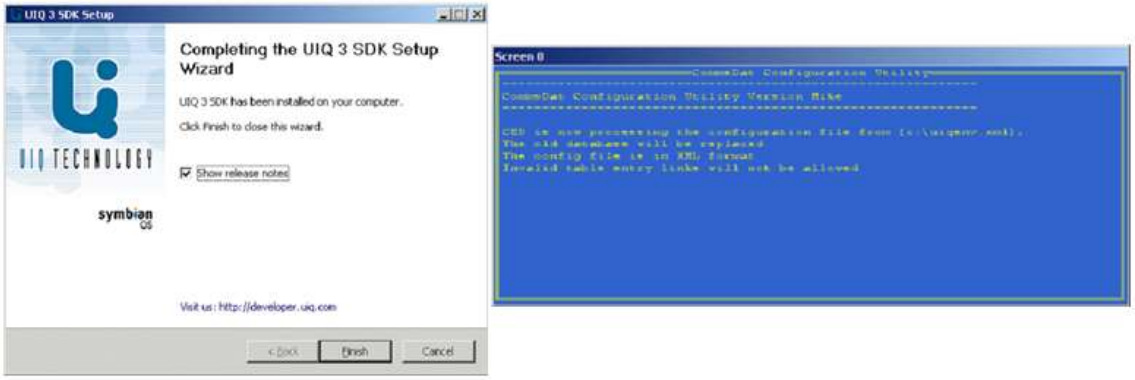

Figura C. 11 Instalación de la solución SDS de Ericsson.

Se instala el complementario para el soporte de dispositivos móviles de *Sony Ericsson*.

Figura C. 12 Instalación de la solución SDS de Ericsson.

Ahora se instala el servidor de aplicaciones con capacidades de manejo de mensajes *SIP Sailfin* configurado con el domino *ericsson.com* el cual viene por defecto.

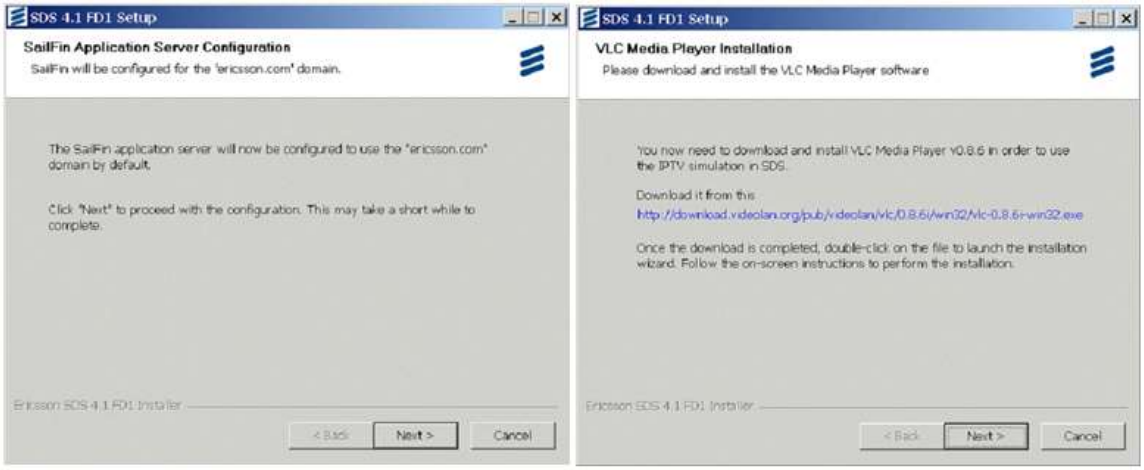

Figura C. 13 Instalación de la solución SDS de Ericsson.

Teniendo completa la instalación como se muestra en la figura C.14, se reinicia el equipo para que *IMS* pueda iniciar como servicio de *Windows.* Este permite tener acceso a *IMS* y a la base de datos *HSS* por fuera del entorno además, sirve para limitar la versión únicamente para los clientes que están registrados y con privilegios para acceder a los servicios de *IMS,* haciendo una simulación del acceso de la base de datos para evitar hacer un registro de manera dinámica.

Figura C. 14 Instalación de la solución SDS de Ericsson.

Una vez reiniciada la maquina, se observa en la figura C.15 la pantalla inicial del *SDS* de *Ericsson SDS* 4.1 y su entorno para desarrollo en *Java* basado en *Eclipse IDE*.

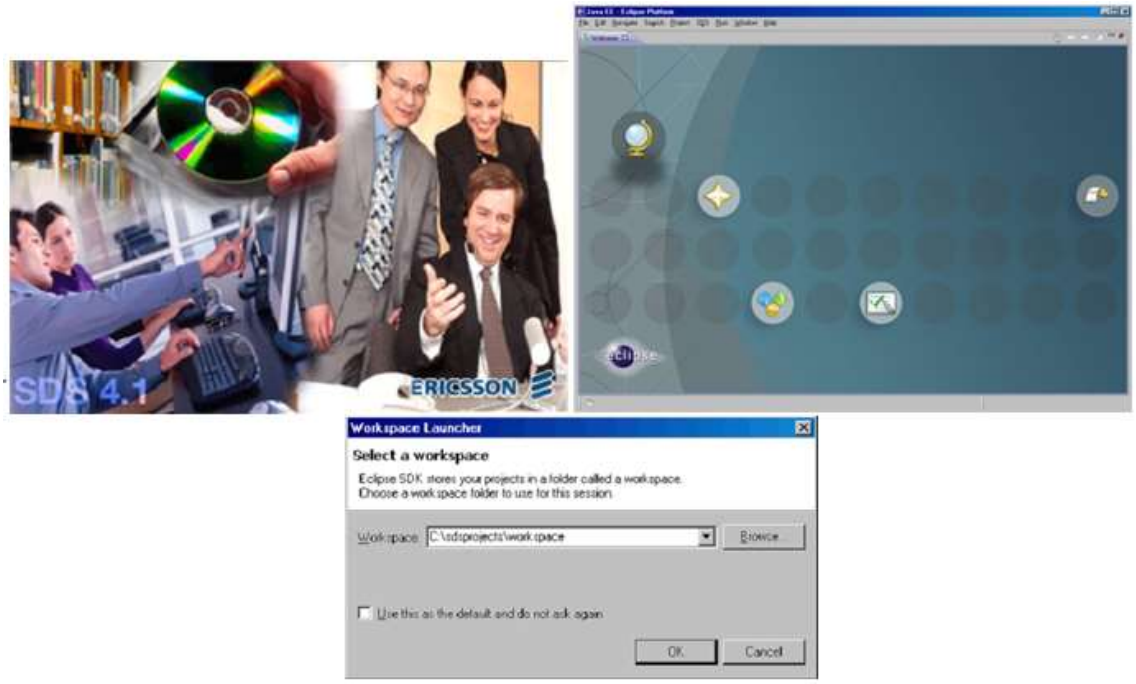

Figura C. 15 Instalación de la solución SDS de Ericsson.

Como se observa en la figura C.16, el entorno de trabajo Eclipse IDE se le adicionan funcionalidades propias de *Ericsson* que permiten crear un proyecto *SIP* paso a paso.

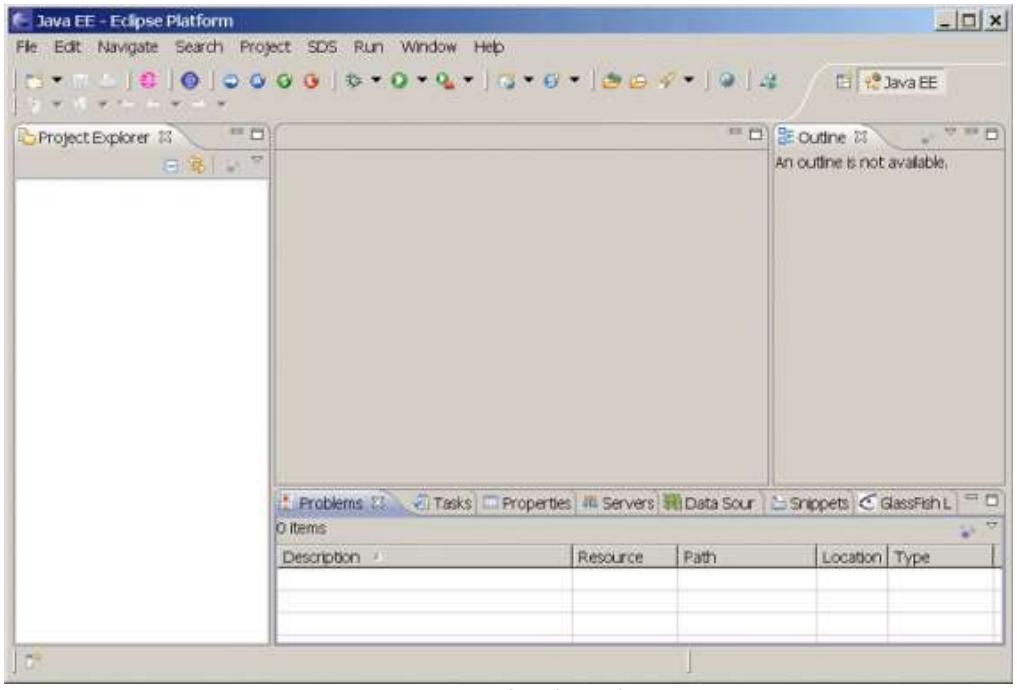

Figura C. 16 Entorno de trabajo Eclipse IDE.

Posteriormente se configura el *JRE* para el entorno de desarrollo *Eclipse IDE* agregando los archivos necesarios *Java* instalados previamente. Se agrega una maquina virtual estándar buscando el directorio raíz donde se instaló el JDK: *Archivos de Programa* o *Program Files* dependiendo del idioma de instalación del sistema operativo y la versión del mismo.

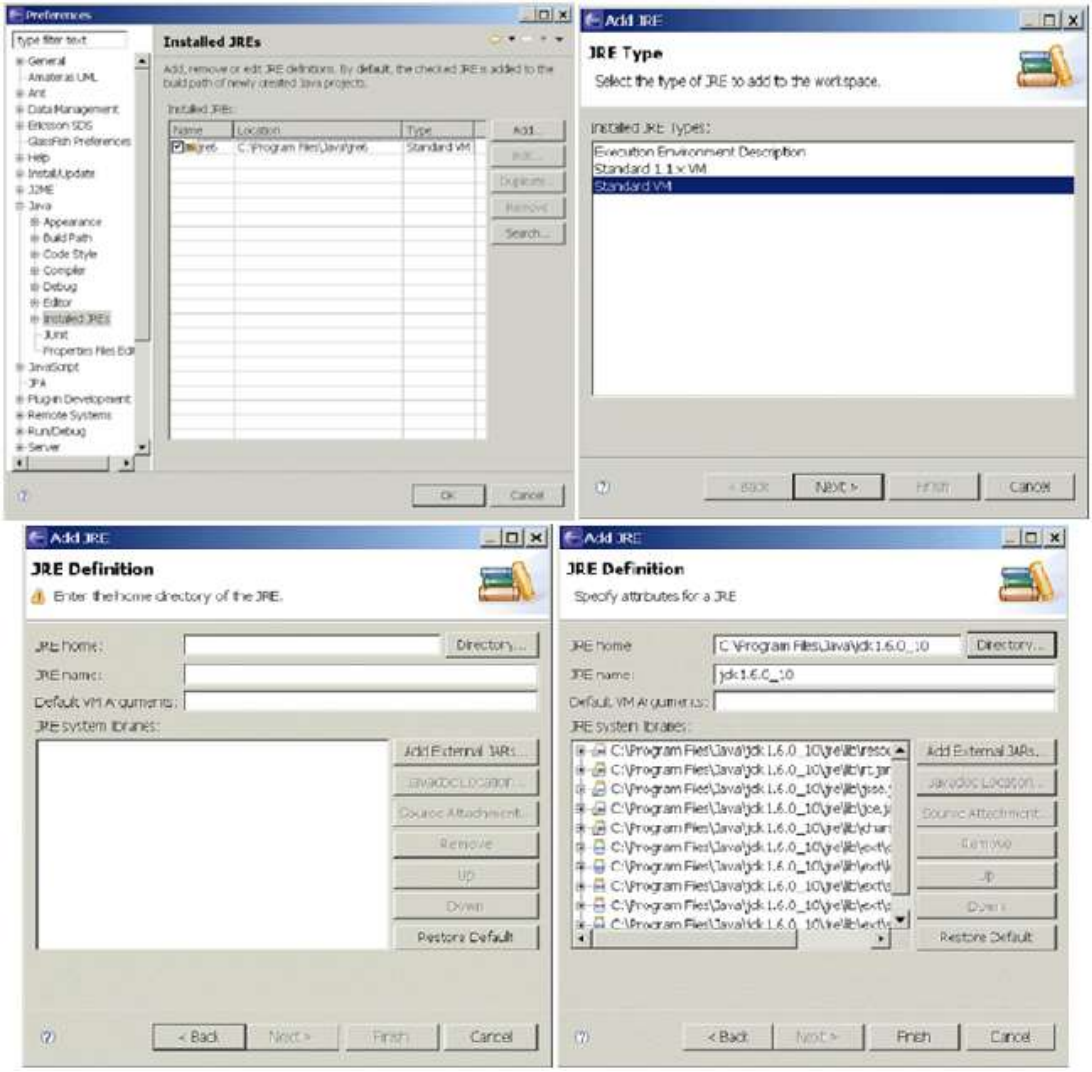

Figura C. 17 Configuración de la maquina virtual en Eclipse IDE.

Para finalizar, la figura C.18 muestra el entorno preparado para trabajar con el simulador de IMS, donde se puede iniciar con la implementación de lo que el proyecto busca presentar.

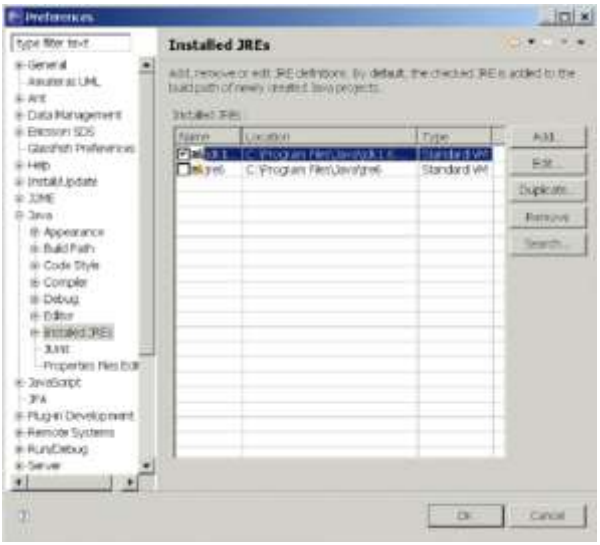

Figura C. 18 Configuración de la maquina virtual en Eclipse IDE.

C.3 Configuración de los servicios y Aprovisionamiento de Datos

En la pestaña de *Provisioning – DNS,* se configuran los dominios de los servicios como el servidor de aplicaciones. Para este caso, se crea el dominio en el *DNS* como ***sip:placa.ericsson*** y se definen los puertos donde el servidor atiende las peticiones a través del *CSCF* y además el tipo de transporte configurado por defecto como *UDP*.

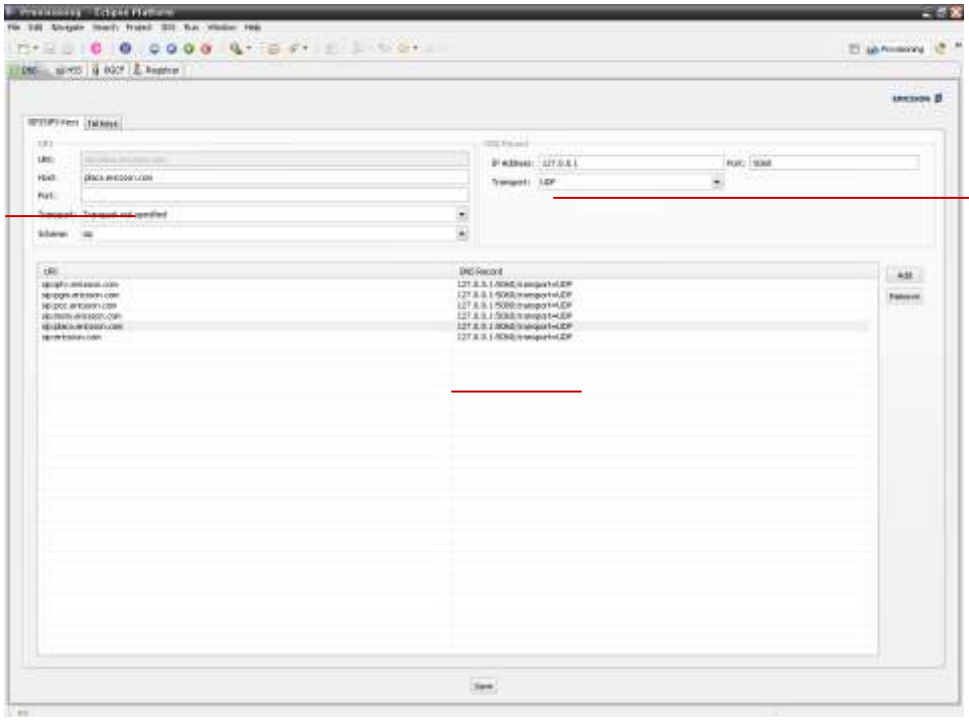

Figura C. 19 Ventana Provisioning-DNS.

Posteriormente se configura al *HSS* por medio de la ventana *Provisioning,* donde se hace la creación y configuración de los servicios que pueden existir dentro de la arquitectura y que son asignados a los usuarios dependiendo de los privilegios que tengan asignados de acuerdo al perfil de cada usuario.

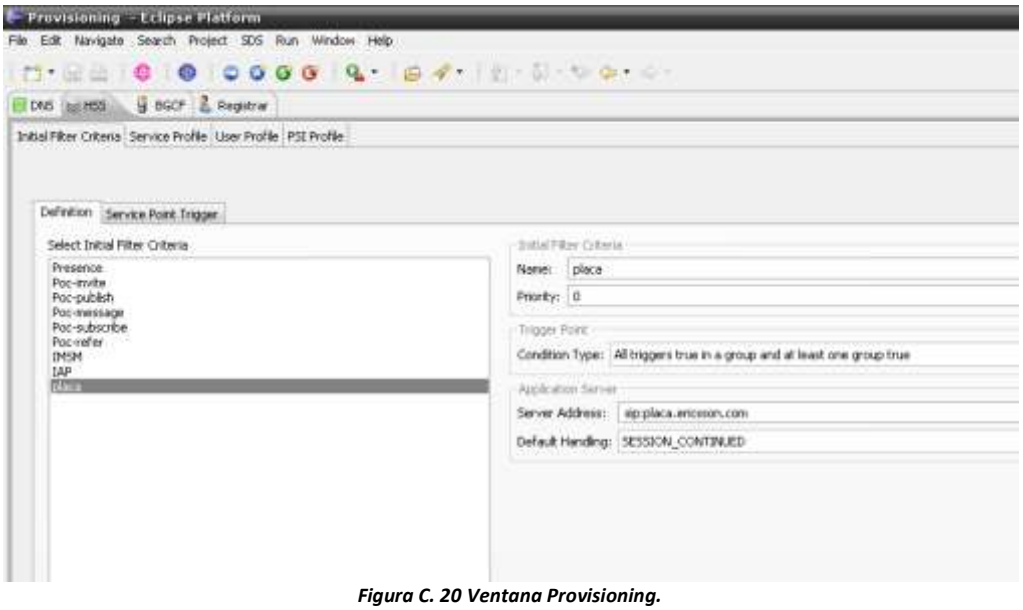

Figura C. 20 Ventana Provisioning.

Como se observa en la figura C.20, se configura inicialmente con la asignación de un nombre al servicio creado, luego la prioridad entre el rango de 0 a 255, la condición de los disparadores del servicio, la dirección *IP* del servidor en donde se tiene al *Servlet* y por último el tipo de manejo que se hace cuando se invoca al *Servlet.*

Ahora se necesita configurar a los diversos disparadores del servicio, en la imagen anterior se observa la elección como condición que todos los disparadores del grupo sean verdad y al menos un grupo sea verdadero, por lo tanto esta condición se toma como disparador.

Figura C. 21 Configuración de los disparadores para el servicio.

Figura C. 22 Configuración de los disparadores para el servicio.

Los disparadores que se agrupan en el **grupo 0** están relacionado para el caso de un mensaje *SIP.* Para el caso que la dirección de envío del mensaje sea ***sip:registrar@ericsson.com***, se diseñan las opciones lógicas que permitan crear las normas de los disparadores del servicio.

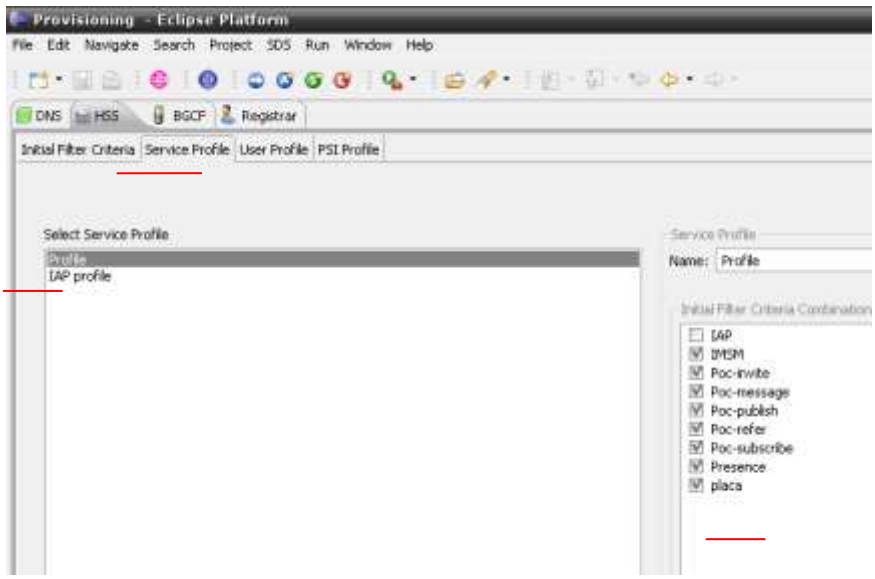

Figura C. 23 Configuración del perfil del servicio.

En perfil de servicio se crea un perfil y a este se le agrega los servicios, para este caso se agrego el servicio creado.

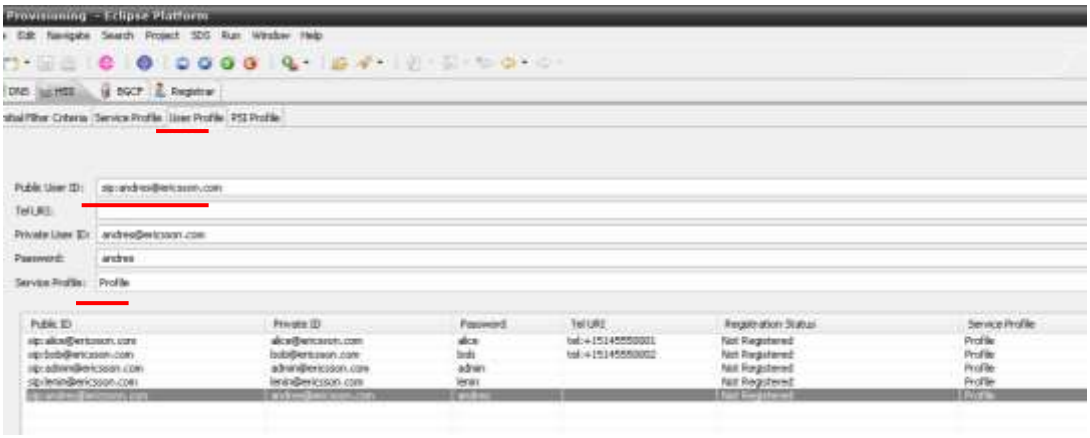

Figura C. 24 Configuración del perfil del usuario.

Se puede crear un ficha de usuario o **perfil de usuario** agregándole un **perfil de servicio**, para este caso se agrega el perfil creado con los servicios que se adicionaron a este, entre ellos encontramos el servicio **placa**.

Una vez terminado los pasos anteriores, se puede crear un proyecto *SIP/WEB* para el *Servlet* que atenderá las peticiones *SIP*, para esto se necesita del asistente mostrado en la figura C.25.

Figura C. 25 Asistente para la creación del proyecto.

El asistente proporciona la ayuda necesaria para la configuración del *Servlet*. Se elige la versión del *SIP Servlet API 1.0* correspondiente a la compilación *JSR-116,* debido a que la configuración de los disparos se hace dentro del *Servlet* de manera manual y controlada lo que permite un total control sobre lo que se quiere ejecutar, caso contrario si se elige a la compilación *JSR-289*, donde la configuración de los disparos de los servicios se hace directamente sobre el servidor donde existe incompatibilidad con los disparadores predefinidos que hasta el momento, no hay soporte para el entorno de la solución *SDS*, dando como resultado servicios que no se disparan de la forma definida ya que esta compilación no requiere crear las reglas de disparo dentro del archivo *sip.xml* del *Servlet.*

Figura C. 26 Asistente para la creación del proyecto.

Se crea dentro del proyecto un *SIP Servlet*, el cual automáticamente toma la configuración de la compilación *JSR-116* ya que esta fue la predefinida para el proyecto.

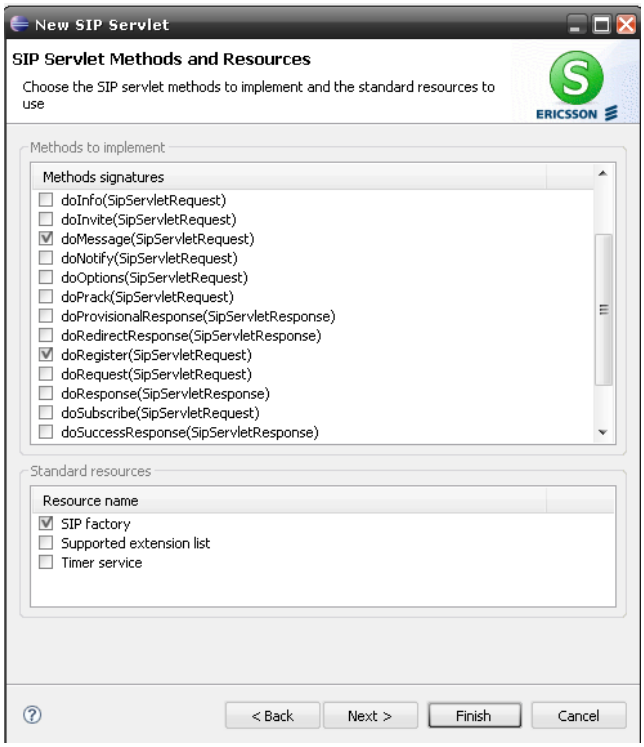

Figura C. 27 Capacidades de SIP-Factory para el SIP-Servlet.

Una vez creado el *SIP Servlet*, elegimos las acciones las cuales van a realizar además, del tipo de recurso que este usara. Se añade a *SIP-Factory* el cual es una librería que permite extraer información de mensajes *SIP* y crear diferentes tipos de respuestas *SIP* de manera correcta.

Figura C. 28 Configuración de las reglas para el Sip Servlet.

A continuación el asistente pide establecer las condiciones en las que actuará el *Servlet*, estas reglas deberán ser coherentes con las reglas que se establecieron en la creación del perfil de servicio en la *HSS*.

En caso de que existen otros servicios creados mediante *SDS,* se debe poner un nombre al *Servlet.* Este nombre debe superar en valor alfanumérico al paquete de servicios provisto por *SDS*, este paquete tiene como nombre *services.sar*.

El servidor de aplicaciones *Sailfin* busca entre los servicios desplegados el que cumpla con las capacidades descritas por el *sip.xml* y elige al que se adecue con las peticiones que recibe. Ya que las comparaciones son secuenciales y el servicio creado tiene como objetivo atender peticiones *SIP*, elegimos un nombre alfanuméricamente inferior a *services.sar* para que el servicio siempre sea invocado para responder a un número reducido de direcciones *SIP.* La forma de elegir un servicio es ordenando cual de los servicios tiene la capacidad de responder a mensajes y se elige al primero del orden alfabético de la A a la Z y posteriormente de la a a la z, por eso se elige un *Servlet* de nombre *services1-z* para que no sea elegido inicialmente y espere hasta que los disparadores secundarios se encarguen de escogerlo. Para este caso, el *Servlet* se llamó *zPlacaServlet116,* usando como convención el número que informa la versión de compilación del mismo.

Con esto, se hace la codificación de las funciones que va a tener el *Servlet*, para el proyecto se usa la base de datos para el servicio que se encarga de registrar la hora de ingreso y el tiempo de conexión destinado a servicios de televisión digital prepagada.

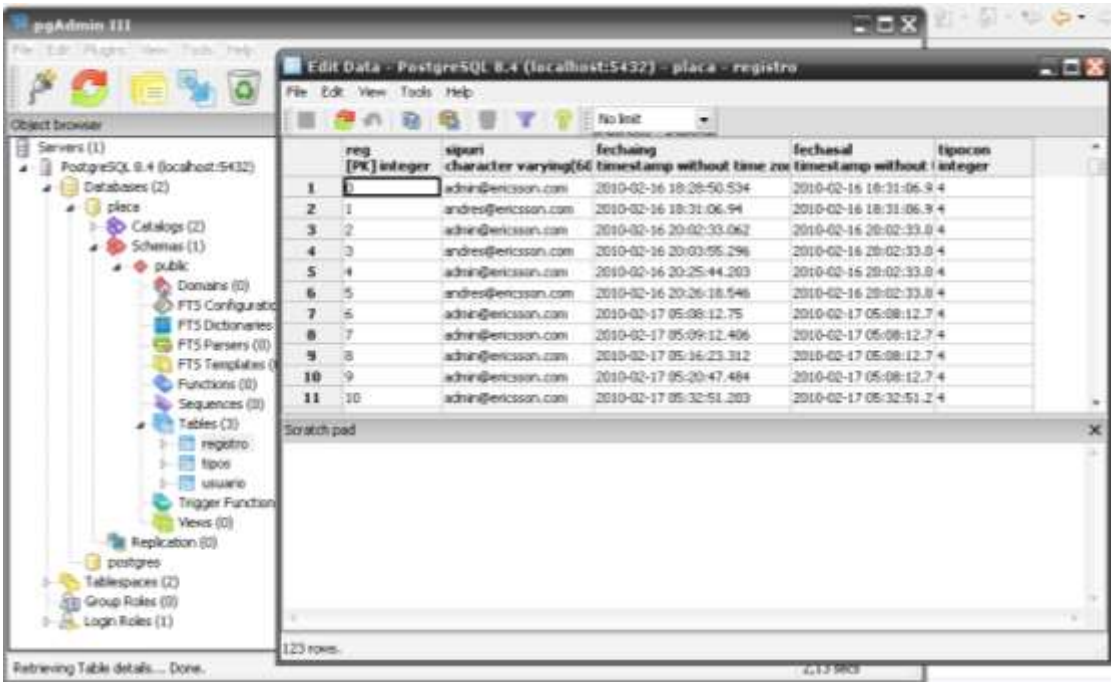

Figura C. 29 Base de datos placa.

La figura C.29 muestra ka base de datos placa donde existen 3 tablas: la tabla registro donde se tiene los registros para el tiempo de entrada y de salida de los usuarios, la tabla usuarios donde se tiene a los usuarios permitidos a usar el *servicio placa,* y la tabla tipo donde identifica al tipo de usuario que ha ingresado en el sistema, este tipo de dato fue pensado para diferenciar entre datos del simulador y del televisor con el *STB*.

Con esta información se pueden crear informes de estadísticas de ingreso para un ambiente en donde al sistema accedan una cantidad elevada de usuarios, para este proyecto estos datos no son relevantes debido al ambiente de pruebas y pocos usuarios que se tiene.

Para la capa de acceso a datos se hace uso de las librerías de *Hibernate* 2.5 y se mapean las tablas de forma objeto relacional. De esta manera se optimiza el acceso a la base de datos y se asegura que los puertos y las sesiones se mantengan estrictamente controlados y soporten múltiples usuarios sin afectar a la integridad de los puertos y del servidor de base de datos.

Algo muy importante al momento de hacer el despliegue del *Servlet* sobre el servidor de aplicaciones, es exportar la librería de acceso a la base de datos provista por la instalación del gestor de base de datos incluyéndola en la carpeta del proyecto además, cuando se intente desplegar al *Servlet* sobre el servidor este preguntara si se quiere adjuntar la librería para ser desplegada, aceptamos y de este modo el *Servlet* tendrá acceso a la base de datos de manera correcta.

ANEXO D. VERIFICACIÓN DEL PROTOTIPO DESPLEGADO SOBRE LA PLATAFORMA DE CONTROL PROPUESTA

Una vez la plataforma de control se encuentre en ejecución y esperando por *clientes STB* como se evidenció en el numeral 4.2 de la monografía, se procede a la ejecución del prototipo implementado y desplegado sobre el emulador *XleTView* como lo muestra la figura D.1, y al mismo tiempo se visualizara los procesos generados en el historial del servidor *Sailfin* que se muestra en la figura D.2.

Cuando se hace la ejecución del prototipo en el emulador de televisión digital interactiva, la aplicación intenta conectarse con el servidor placa, este a su vez recibe esta petición y determina que es un mensaje indicando que un *cliente STB* se ha conectado.

En la figura D.2, el historial muestra al servidor placa en espera por algún *cliente STB*, una vez establecida una conexión, este muestra un mensaje "*Entró un usuario!!",* en donde posteriormente a modo de prueba, se procede a cerrar el *hiloServidor* para evitar el consumo de recursos y el *servidor placa* ahora espera por el respectivo *login* y *password* para la autenticación del *cliente STB*.

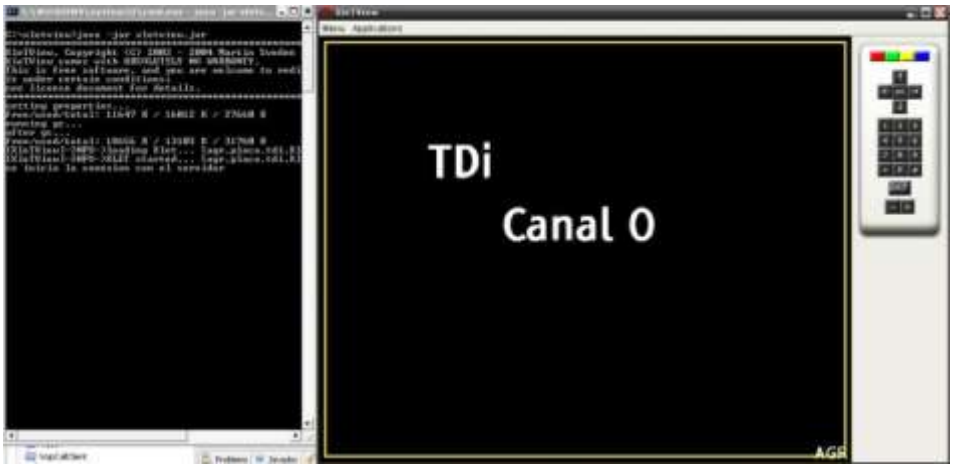

Fig. D.1Ejecución del prototipo usando XleTView.

```
Servidor (1) [Java Application] C:\Archivos de programa\Java\jdk1.6.0_19\bin\javaw.exe (14/05/2010 12:12:22)
PHANTOM - CLIENTE [STB] Ha Recibido un mensaje de: <sip:registrar@ericsson.com>
de tipo: text/plain
con el contenido: SIP SERVLET: Registrado:<sip:admin@ericsson.com>
 desde: Cliente SCP
PHANTOM - CLIENTE [STB] Se registró al servidor exitosamente!!
 tiempo de registro 880 ms
PHANTOM - CLIENTE [SERVIDOR] Esperando usuarios.........
PHANTOM - CLIENTE [SERVIDOR] Entró un usuario!!!
DEBUG: con solo uno es suficiente para probar cerrando Servidor de escucha!
PHANTOM - CLIENTE [STB] esperando login y password
```

Fig. D.2. Historial durante la ejecución del prototipo.

Se presenta a continuación en la figura D.3, la pantalla relacionada con la validación del *cliente STB*, donde se espera por el ingreso de estos datos y posteriormente por una respuesta y en caso de ser exitosa, el *cliente STB* tendrá acceso a los recursos y servicios a los cuales tiene privilegio.

Como se puede observar en la figura D.3 y D.4, el proceso de validación inicia con el ingreso de los datos del *cliente STB*, estos datos son enviados al *servidor placa* donde la clase *admin* los toma y los envía al *servlet placa,* se espera por la respuesta a esta petición de validación y para este caso como la validación es exitosa, se crea al *cliente fantasma* quien es el encargado de mediar e interactuar con los mensajes de servicios de *Chat* y de *Presencia* y otros recursos que ofrece *IMS*.

Una vez el usuario se encuentre validado, se autoriza para que pueda acceder a los recursos y servicios a los cuales tiene privilegio, como la figura D.5 muestra la pantalla de gestión para el *cliente STB*, donde se ofrece el servicio de *Chat* apoyado con el servicio de *Presencia* en la opcion *amigos,* los canales de televisión digital iteractiva en la opcion *T.V,* y la opcion *App* implementa un agregador *para web feed* donde se ofrece información actualizada de manera frecuente a los suscriptores.

Fig.D.3. Pantalla de validación del prototipo implementado.

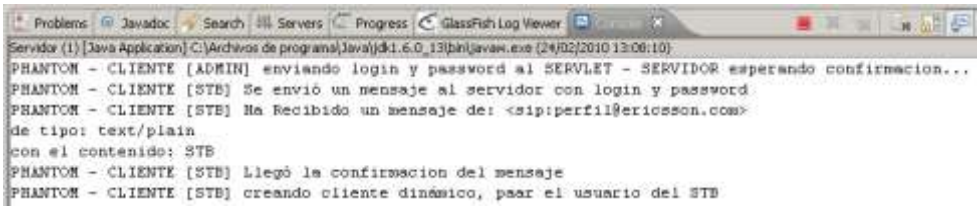

Fig.D.4. Historial generado durante el proceso de validación y autorización del cliente STB.

En el momento del acceso del *cliente STB* a la pantalla de gestión del prototipo, se observa en la figura D.6 el historial del *servidor placa* durante este proceso. En este caso, andres es el usuario que ha sido validado y autorizado para acceder a la aplicación, cuando esto sucede se consulta a los respectivos *buddies* los cuales tiene inscritos para *Chatear.* Por otro lado, se modifica el estado de la presencias a *true,* indicando que andres se encuentra disponible y visible para otros *clientes STB* como para *clientes IMS.* Luego espera por cualquier cambio en el estado de los *buddies* suscritos para ser notificado.

Fig.D.5. Pantalla de gestión para el cliente STB.

```
PHANTOM - CLIENTE [STB] Ha Recibido un mensaje de: <sip:registrar@ericsson.com>
de tipo: text/plain
con el contenido: SIP SERVLET: Registrado:<sip:andres@ericsson.com>
 desde: Cliente SCP
PHANTOM - CLIENTE [STB] Se registró al servidor exitosamente!!
 tiempo de registro 70 ms
sip:lenin@ericsson.com
processStateChanged
processUpdatePresentityResult
processSubscribeResult
Status: true Remote: sip:andres@ericsson.com;list=Others Event: presence
processSubscribeResult
Status: true Remote: sip:andres@ericsson.com;list=Todos Event: presence
Entró por ELSE Lenin esta ahora offline
PHANTOM - CLIENTE [STB] se esta enviando 1 registro|s| de contactos
processSubscribeNotification
```

Fig.D.6. Historial durante el proceso de validación y autorización del cliente STB.

OPCIÓN AMIGOS

La opción amigos permite establecer una sesión de *Chat* con los *buddies* a los cuales el *cliente STB* tiene inscritos. En este caso, tenemos al cliente *lenin* quien se registra en *IMS* y accede mediante un *cliente Windows* a *IMS* como se muestra en la figura D.7.

Fig.D.7. Cliente IMS

Una vez registrado *lenin,* su estado pasa de *offline* a *online,* indicándoles a los usuarios que lo tienen dentro de su lista de contactos, que a partir de ese momento se encuentra disponible.

Es así que el *cliente STB andres* en ese momento es informado mediante una notificación que *lenin* pasó a un estado *online* y por consiguiente está disponible para establecer una sesión de *Chat*, como se observa en la figura D.8*.*

Durante el anterior proceso, el historial de la figura D.9 muestra la notificación del cambio de estado de *lenin* y el envío a la aplicación *DVB-J* de los registros relacionados con este suceso.

Ahora el cliente *lenin* envía un mensaje por la sesión de *Chat* establecida con destino al *cliente STB andres* como se observa en la figura D.10. Posteriormente se visualiza en la aplicación de *TDi* la llegada de mensaje enviado por la sesión de *Chat* y se confirma en el historial de la figura D.11 el inicio de la sesión de *Chat* y el envío del mensaje por esta sesión del *cliente IMS lenin* hacia el *cliente STB andres.*

Ahora el cliente *STB andres*, responde a este mensaje desde la aplicación *DVB-J* en el entorno de la *TDi* hacia el *cliente IMS lenin* como lo muestra la figura D.12. Se visualiza en la figura D.13 el historial relacionado con este proceso, que el enrutamiento del mensaje hacia el *cliente IMS lenin* está a cargo del *cliente fantasma,* quien es el que busca a la sesión de *Chat establecida* para enviar los mensajes con destino al cliente *IMS lenin.*

Fig.D.8. Notificación del estado de los buddies.

```
Status: true Remote: sip:andres@ericsson.com;list=Others Event: presence
Entró por ELSE Lenin esta ahora offline
PHANTOM - CLIENTE [STB] se esta enviando 1 registro(s) de contactos
processSubscribeNotification
processSubscribeResult
Status: true Remote: sip:andres@ericsson.com;list=Todos Event: presence
Lenin esta ahora Online
PHANTOM - CLIENTE [STB] se esta enviando 1 registro(s) de contactos
processSubscribeNotification
```

Fig.D.9. Historial correspondiente a la notificación del estado.

Fig.D.10. Envío del mensaje por la sesión de chat al cliente STB.

```
processInvitation
<init>
Una sesion de chat proveniente de tu contacto --> Lenin
addChatSessionListener
processStarted
processText
processSubscribeResult
Status: false Remote: sip:andres@ericsson.com;list=Todos Event: presence
```

Fig.D.11. Historial correspondiente al inicio de la sesión de Chat.

Fig.D.12. Envió de mensaje por la sesión de Chat del cliente IMS al cliente STB.

```
processInvitation
<init>
Una sesion de chat proveniente de tu contacto --> Lenin
addChatSessionListener
processStarted
processText
Se esta mandando un msg a Lenin con el contenido: qqqq
Se encontro una instancia de chat: Lenin
Se envio datos al STB
```

Fig.D.13. Historial correspondiente al envío del mensaje al cliente STB.

OPCIÓN *T.V*

La idea principal de la opción *T.V* que ofrece el prototipo, es simplemente la existencia de cierto número de canales a los cuales el cliente tiene privilegio. Estos canales para este

prototipo, se han definido como una serie de películas las cuales están a disposición gracias al entorno de la *TDi.*

Como se puede observar en las figuras D.14, D.15, D.16, D.17, la opción *T.V* ofrece inicialmente los diferentes canales a los cuales el *cliente STB* tiene privilegio. Una vez elegido el canal de preferencia, el menú que se muestra en la pantalla puede ocultarse en cualquier momento dependiendo de la elección del cliente, además de poder hacer el cambio de canal al momento de una emisión de un canal existente.

Fig. D.14. Canales disponibles en el prototipo.

Fig.D.15. Elección de un canal con el menú desplegado.

Fig.D.16. Emisión de un canal con el menú de gestión de la aplicación.

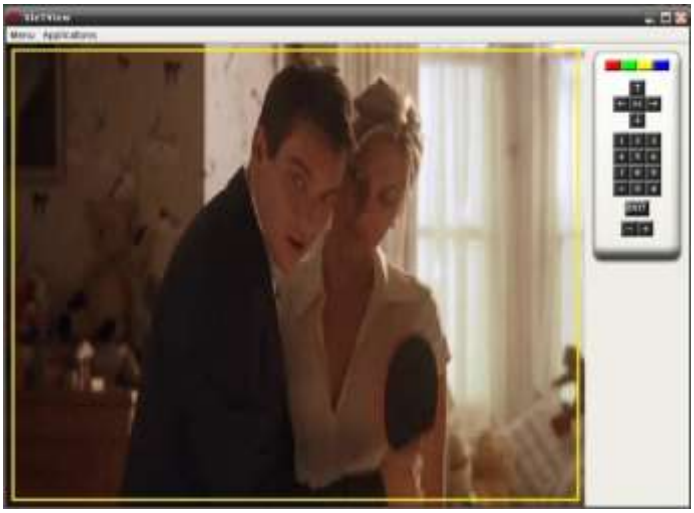

Fig. D.17. Emisión del canal en pantalla completa.

OPCIÓN APP

La opción *APP* que ofrece el prototipo, presenta las aplicaciones a las cuales el cliente tiene acceso. Para este proyecto se definió la implementación de un agregador W*eb feed,* donde se obtiene el contenido difundido por una página web relacionado con las novedades que publica el sitio.

Como lo muestra la figura D.18, se observa una lista de servicios dinámica a las cual el *cliente STB* está inscrito, solamente se tiene implementado un servicio para recibir actualizaciones via RSS[3] y solo es necesarios agregar los *feeds* para comenzar a recibir el contenido web disponible.

[3] *RSS es una grupo de formato de fuentes web codificados en XML, usados para difundir información actualizada frecuentemente a los usuarios suscritos a la fuente de contenidos.*

Fig. D.18. Lista de servicios dinámica para el cliente STB.

La figura D.19 y D.21 muestra la suscripción al *web feed* proporcionado por el diario El Tiempo de Santafé de Bogotá y la emisora la W de la misma ciudad. Por otra parte las figuras D.20 y D.22 muestran los historiales en donde se muestra la descarga del contenido web de los *feed* a los cuales el agregador se encuentra suscrito.

Fig.D.19. Contenidos proporcionados por el feed del diario el Tiempo.

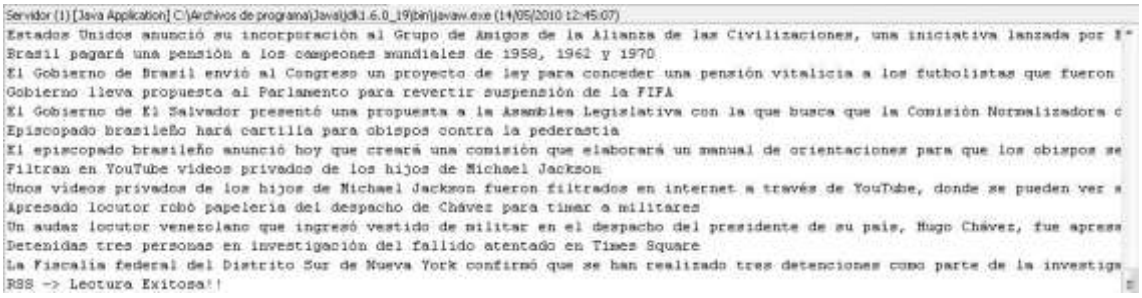

Fig.D.20. Historial correspondiente a la descarga del contenido web del diario El Tiempo.

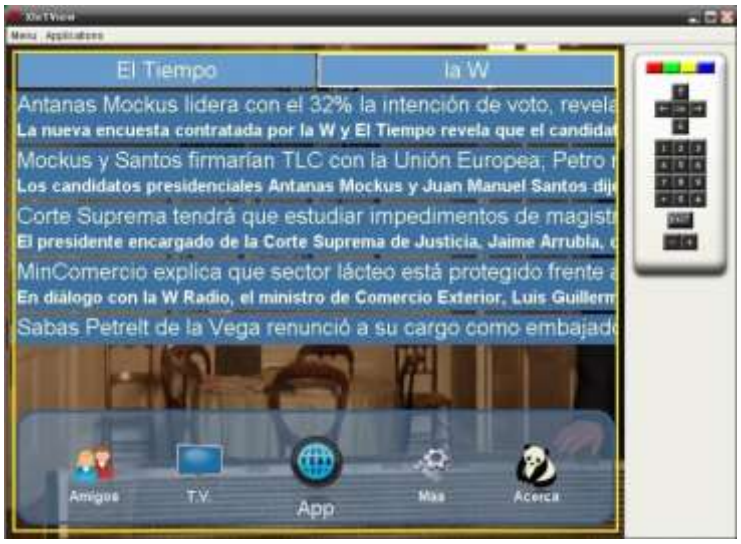

Fig.D.21. Contenidos proporcionados por el feed de la emisora la W.

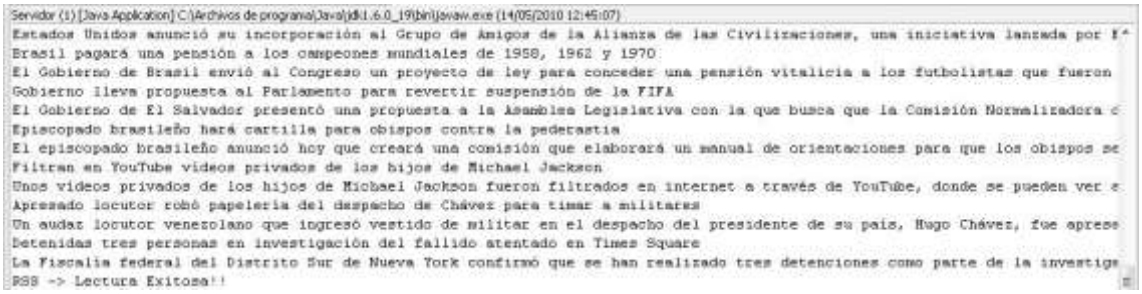

Fig.D.22. Historial correspondiente a la descarga del contenido web de la emisora la W.

Como se observa, la organización del agregador *web feed* esta diferenciado por tres campos:

- El nombre del *feed.*
- Titulo del contenido.
- Descripción del contenido.