3. UN ENTORNO DE DESARROLLO: EL SDS DE © ERICSSON

3.1 Herramientas de desarrollo de aplicaciones y servicios IMS: ¿Por qué el SDS?

Como se verá en el capítulo 5, existen soluciones IMS concretas que ofrecen algunos proveedores y operadores de telecomunicaciones como Ericsson, Nokia, Vodafone o Telefónica Móviles. Sin embargo, la mayoría de estas iniciativas consisten en actuaciones dedicadas y a medida del cliente, o a medida de un operador concreto. Sin embargo, IMS pretende aportar más servicios de los que puede aportar el propio *core*, su objetivo es proporcionar un marco estándar para que la amplia comunidad de desarrolladores existente en Internet pueda adaptarse rápidamente a IMS y aprovechar sus ventajas proponiendo servicios novedosos cada vez más centrados en el usuario. Para lograr este objetivo, es preciso disponer de las herramientas de desarrollo que faciliten la programación de servicios y aplicaciones IMS. Los principales entornos — más escasos en número que las soluciones IMS a medida — se describen brevemente a continuación aportando los datos suficientes para un estudio comparativo posterior que proporcione una idea justificada de la elección de una herramienta frente a otras.

3.1.1 RADVISION IMS Developer Suite

RADVISION es una compañía dedicada durante más de una década a soluciones de voz y vídeo sobre IP y experta en mercados 3G. Para IMS, RADVISION ofrece una herramienta de desarrollo de aplicaciones formada por bloques constituyentes que son a su vez herramientas de desarrollo para los protocolos de señalización y de medios de IMS. En la figura 3.1 se observa mejor la heterogeneidad de la solución de RADVISION.

Esta herramienta se compone pues de cuatro entornos de desarrollo para protocolos, a saber: SIP (IMS SIP *Toolkit*), MeGaCo (IMS MEGACO/H.248 *Toolkit*), DIAMETER (IMS DIAMETER *Toolkit*), RTP/RTCP (IMS Advanced-RTP/RTCP *Toolkit*). Además, para el soporte de estos desarrollos, la *suite* de RADVISION aporta plataformas de servidor, cliente y testeo: las plataformas IMS SIP *Server Platform*, IMS *Client Suite* y ProLab IMS *Test Solution*.

Los *toolkits* de desarrollo facilitan la programación sobre los protocolos de

Figura 3.1.: Esquema general de los bloques que componen el *IMS Developer Suite*. Fuente: RADVISION

IMS, asistiendo la creación de mensajes de protocolo y estableciendo modelos de comunicación de cliente-servidor. Contienen documentación, aplicaciones de muestra y guías tutoradas. Además, la programación está basada en el cumplimiento de los protocolos uno a uno, por lo que permite cubrir prácticamente todas las interfaces de IMS. El desarrollador ha de aprender los asistentes para la creación de mensajes de protocolo, inserción de cabeceras, etc.

Tanto la plataforma cliente como la del servidor ofrecen soporte a las aplicaciones creadas con los *toolkits* e incorporan los bloques necesarios para el despliegue de aplicaciones cliente y servidor. La plataforma servidora proporciona los servidores SIP de IMS, así como funcionalidades de los servidores de aplicaciones (B2BUA, *Redirect*,...). La plataforma cliente es un *framework* independiente del sistema operativo — por lo que es multiplataforma — y *mapea* las capacidades IMS del lado del cliente a través de APIs de alto nivel propietarias.

Por último, la *suite* de testeo está diseñada para perfeccionar los servicios creados y completar el ciclo de desarrollo. Consiste en un paquete de *scripts* embebidos en la herramienta y *plug-and-play*, archivos de medios de pruebas, simulación avanzada del *User Equipment* (UE), señalización *online* y análisis de los medios y su calidad. Todo ello integrado en una plataforma de testeo del equipo de usuario (UE).

3.1.2 Nokia Siemens Networks IMS Developer Program

La solución de la multinacional formada por los gigantes Nokia y Siemens cambia la visión de los desarrollos con respecto a la solución nativa en el protocolo de RADVISION. Nokia Siemens apuesta por el desarrollo de APIs en lenguajes archiconocidos como Java o .NET y por la integración en un entorno de desarrollo o *Integrated Development Environment* (IDE) que agrupe de forma homogénea todas las funcionalidades que aporta. Así, dentro de **Nokia Siemens Networks** se ha aprovechado su experiencia en desarrollos IMS a medida [19, 20] para desarrollar una herramienta para la creación de servicios IMS integrable con las herramientas de desarrollo, IDEs, emuladores y sistemas operativos favoritos del desarrollador.

Para el desarrollo del servidor la solución de la herramienta de Nokia Siemens es el **IMS *Network Emulator***, un simulador del núcleo de la red de IMS (CSCF, HSS) y simulador del servicio de presencia de la *Open Mobile Alliance* (OMA) y un servidor *XML Data Management*. El IMS *Network Emulator* es en sí una aplicación Java, por tanto *multiplataforma*. Incluye una interfaz gráfica *Graphical User Interface* (GUI) para las configuraciones locales, y una interfaz Web para las remotas.

Del lado cliente un desarrollador puede hacer uso del **Java ME IMS SDK** si elige programar para teléfonos con soporte Java, o bien **.NET IMS SDK** para el desarrollo de clientes Windows o Windows Mobile.

El **Java ME IMS SDK** proporciona el entorno de desarrollo para los terminales con soporte Java ME. Este entorno incluye APIs de alto nivel para servicios SIP e IMS. Este hecho reduce el tiempo de desarrollo, al ahorrar la implementación de una pila SIP en el dispositivo y la elaboración de todos los mensajes de este protocolo de cara a cubrir las necesidades del servicio que se desee desarrollar. Este *Software Development Kit* (SDK) hace uso del borrador de la especificación JSR-281 (pre-JSR-281), la API de servicios IMS. En este sentido es preciso apuntar que existen dos versiones de la librería Java ME IMS SDK: la versión para dispositivos más limitados (dispositivos con soporte MIDP 2.0/CLDC 1.1) que incluye una pila SIP y compatibilidad con aplicaciones Java ME ligeras (CLDC). Asimismo, existe otra versión del SDK dirigida a dispositivos con más capacidades como aquellos con sistema operativo Symbian S60 3rd edition, que ofrecen un soporte mucho más potente al protocolo SIP a través de la API JSR-180 para Java ME.

3.1.3 Ericsson Service Development Studio (SDS) y comparativa

En este apartado se argumenta la elección de esta herramienta de desarrollo de Ericsson AB apoyados por una breve descripción como elemento comparativo con respecto a las dos plataformas antes descritas. En secciones posteriores se estudia con profusión la solución elegida y la metodología de creación y despliegue de servicios IMS.

La compañía sueca Ericsson ha liderado durante los últimos años el desarrollo tanto de *hardware* IMS como de soluciones *software*. El *Service Development Studio* (SDS) es la herramienta de desarrollo de aplicaciones y servicios IMS y es parte del conjunto de productos IMS de Ericsson. Por otro lado, Ericsson ha copado la tarea de la especificación de las APIs para IMS como son la API de servicios IMS JSR-281 y la API de *enablers* de la comunicación, la API JSR-325; y participado activamente en los desarrollos para el servidor (especificaciones JSR-116 y JSR-289) y APIs para el control de medios (JSR-309). Esta experiencia en el desarrollo de las librerías para IMS queda patente en el SDS, que al igual que la solución de Nokia Siemens proporciona APIs de alto nivel para ocultar la complejidad de la red y del dipositivo para el desarrollador. Asimismo, incluye una mayor cantidad de plantillas, tutoriales, códigos de ejemplo y asistentes para disminuir el tiempo de desarrollo.

El SDS está completamente integrado en el IDE Eclipse, siendo este factor una

ventaja considerable al ofrecer un entorno conocido para los desarrolladores, pero también una restricción al ligar el entorno de desarrollo a un IDE o plataforma concreta. Tanto la solución de RADVISION como la de Nokia Siemens eran multiplataforma. En contra del SDS juega su compatibilidad única con Windows en la actualidad. No obstante, están contempladas más plataformas en su planificación o *roadmap*.

Ofrecer APIs a los programadores en un lenguaje ampliamente extendido es un factor clave para la proliferación de aplicaciones, por lo que este factor juega en contra de la solución de RADVISION, que además implica una curva de aprendizaje mucho más lenta.

A grandes rasgos, el SDS consiste en una plataforma que permite la creación de aplicaciones y servicios tanto cliente como servidor extremo a extremo haciendo uso de un simulador del *core* de IMS, soporte para la mayoría de *enablers* de servicios estandarizados por la OMA (presencia, gestión de grupos, *Push-To-Talk over Cellular* (PoC), mensajería IMS), APIs y plataformas del cliente (dispositivos Symbian UIQ/S60 y CLDC), emuladores de dispositivos y servidores SIP. Con esta breve enumeración ya es posible apreciar la mayor completitud de la solución de Ericsson, dado que cuenta con un mayor soporte para dispositivos y para los *enablers*. Y, sobre todo, la inclusión de un servidor de aplicaciones SIP potente, conforme al estándar del 3GPP y a la API JSR-116 (y en un futuro la JSR-289) y basado en tecnologías Java y centradas en Internet.

Sin embargo, la mayor ventaja es quizás la versatilidad para configurar la herramienta como un simulador del *core* IMS en redes locales, remotas, con componentes distribuidos, etc. Así como la capacidad de conexión con entornos reales y servicios remotos de Ericsson[1] como el *Ericsson's Remote IMS* (RIMS) o el *IMS Expert Centre*.

Con el estudio de las diferentes herramientas de desarrollo de aplicacione IMS ha sido posible realizar una breve comparativa. Sin embargo, las ventajas del SDS, así como su problemática o sus limitaciones, se perciben con un análisis más pormenorizado a lo largo del presente capítulo y el capítulo 4.

3.2 ¿Qué es el SDS?

El ***Service Development Studio*** **(SDS)** es un conjunto de herramientas *software* que permiten el desarrollo y testeo de aplicaciones que proporcionan servicios extremo a extremo o *end-to-end* (*e-2-e*) basados en *IP Multimedia Subsystem* (IMS). El SDS permite el desarrollo, simulación y testeo tanto del lado cliente como servidor de las aplicaciones IMS[2].

[1] En realidad de cualquier otra compañía, puesto que IMS es un estándar, pero se indica Ericsson ya que sus productos están desarrollados y son conocidos.

[2] Una vez se está tratando el entorno de desarrollo *software*, el concepto de **aplicación** va ligado al de **servicio** ya que los programas o aplicaciones desarrollados con el SDS tienen la máxima de proporcionar servicios IMS al usuario. No obstante, el concepto de servicio es aún más amplio, dado que un servicio puede ser una combinación de aplicaciones que proporcionen servicios de

El SDS proporciona APIs del servidor basadas en el estándar, así como una arquitectura emulada del *core* IMS y APIs de alto nivel para la parte cliente y para los servicios de comunicaciones.

Con la combinación de un "contenedor" SIP (o SIP *container*) real y sus herramientas de desarrollo, el SDS permite a los operadores de red fijos y móviles y a los proveedores de servicios desarrollar de una manera más sencilla y rápida servicios de valor añadido y de gran calidad. Estos servicios han de ser los que catapulten IMS para poder satisfacer todas las posibilidades que presentan las redes de nueva generación.

El *Service Development Studio* (SDS) es un producto de ©Ericsson perteneciente al *Ericsson IMS Solution*, el conjunto de soluciones tecnológicas tanto *hardware* como *software* para IMS en las que Ericsson es líder mundial [21]. Por lo tanto, Ericsson ha diseñado todas sus herramientas de acuerdo al estándar y compatibles entre sí, con la idea de estar presente en todo el proceso ligado a una aplicación, desde la idea y requerimientos iniciales, hasta el despliegue en una red real. Las primeras fases se completarían con el SDS, y a partir de ahí, el *software* del servicio desarrollado podría desplegarse de forma transparente en el entorno real IMS *Common System* de Ericsson. En concreto en los servidores de aplicación – como por ejemplo es el Oracle (BEA[3]) Web Logic SIP Server (BEA WLSS) [22] o el polivalente *Oracle Communications Converged Application Server* (OCCAS) –. De esta manera, el SDS puede configurarse para simular el núcleo IMS completo e incluso funcionar como un efectivo (y prácticamente gratuito) servidor JEE/SIP[4] como banco de pruebas real de un servicio. Asimismo, es posible interconectar el SDS con entornos reales para verificar eficientemente los servicios y realizar las pruebas pertinentes sin repercutir gravemente en el coste.

La herramienta de desarrollo SDS viene integrada como un conjunto de *plugins* del *framework* **Eclipse *Integrated Development Environment* (IDE)**[5]. Es decir, las capacidades del SDS se presentan como un conjunto de *plugins* dentro de la interfaz de usuario del Eclipse IDE. En la figura 3.28 se muestra el entorno de trabajo de Eclipse, con la herramienta de desarrollo de aplicaciones y servicios de IMS, **SDS**, completamente integrada como parte del *workbench*.

---

poco valor añadido por sí mismos pero que combinados ofrezcan una solución más atractiva.

[3]Oracle adquirió BEA Systems en 2008, así como BEA Systems adquirió Weblogic, Inc. en 1998.

[4]El SDS está completamente integrado, a partir de su versión 4.0 con el **Sailfin JEE/SIP *Application Server*.**

[5]Eclipse, o Eclipse Platform formalmente, es en sí una amalgama de *plugins*. A pesar de tener muchas funcionalidades nativas, son a veces muy genéricas. Se necesitan herramientas adicionales para extender la plataforma para trabajar con nuevos tipos de contenidos, llevar a cabo nuevas capacidades o "especializar" la herramienta en un ámbito específico. La plataforma Eclipse trae un mecanismo de descubrimiento, integración y ejecución de módulos llamados *plugins*. Un proveedor de una herramientas las desarrolla como un módulo por separado pero para que opere con los archivos del *workspace* y superponga sus elementos de interfaz de usuario en este mismo espacio de trabajo o *workspace*. Cuando Eclipse se arranca, el usuario percibe un entorno de desarrollo integrado (IDE) compuesto de un conjunto de *plugins* disponibles.

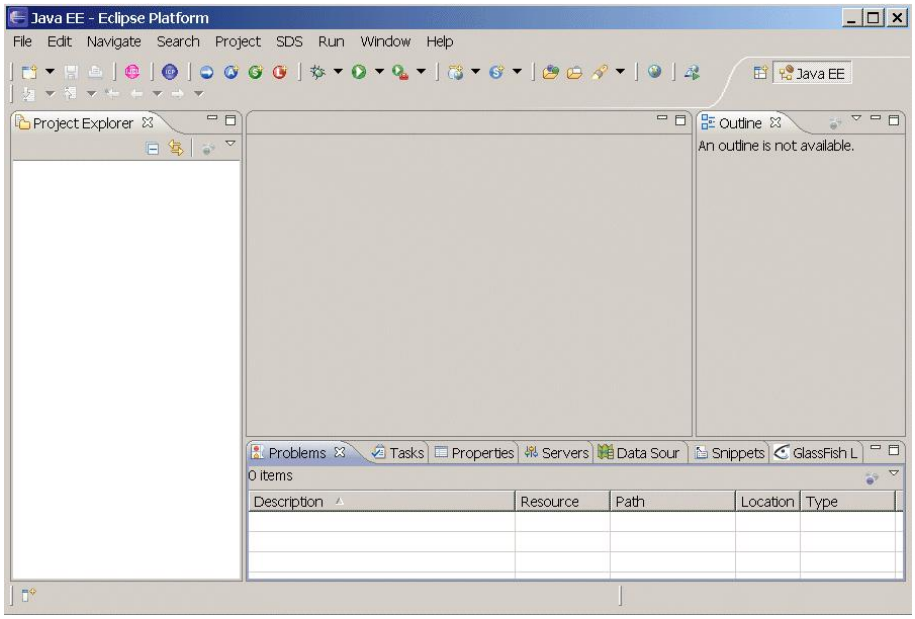

Figura 3.2.: Ventana del *workbench* del SDS

3.2.1 Beneficios del SDS

El SDS se ejecuta en un PC con entorno ®Windows y soporta, como ya se ha introducido, la creación aplicaciones IMS cliente y servidor y permite su testeo y validación extremo a extremo (*end-to-end*). Para ello, simula[6] el *core* de la red IMS, los *enablers* de servicios, el dispositivo móvil y los servidores JEE/SIP. Todo ello ofreciendo una solución intuitiva y orientada a los estándares para el desarrollo de nuevos servicios y aplicaciones.

El SDS proporciona APIs para ocultar la complejidad de la red y del dispositivo al programador, e incluye multitud de plantillas y asistentes (*wizards*) para agilizar las primeras fases de los desarrollos.

Con el SDS, los desarrolladores pueden emplear estas APIs para controlar y acceder a capacidades avanzadas como el PGM, VoIP, o el PoC (o PTT) descritas en el capítulo 2 e incluso en su versión 4.1 ya incorpora un simulador de *Internet Protocol Television* (IPTV) y en entregas posteriores se añadirá soporte para servicios adicionales como la telefonía IMS (*IP Multimedia Telephony*). Además, se verá en la sección 3.3 la hoja de ruta o *roadmap* que plantea Ericsson para su herramienta SDS. En esta planificación vienen detalladas las funcionalidades que incorpora a cada versión, vislumbrándose soporte para la gestión del *Business and Operations Support Systems* (BSS/OSS) por parte del operador: configuración de dispositivos,

[6] Aunque en este documento se utiliza a menudo el término de "emulador" se considera que, en rigor, **el SDS simula núcleo de la red IMS**. Si fuera un emulador, trataría de imitar el funcionamiento del *core* igualando su comportamiento e incluso mejorando aspectos del propio entorno real. Esto no es así, porque como se verá más adelante, la herramienta presenta limitaciones y en ningún caso iguala las funcionalidades de una red IMS real, en cualquier caso las simplifica.

gestión de clientes, provisión, tarificación y operación y mantenimiento (*Operations and Maintenance* (O&M)).

De igual manera, es posible configurar el SDS — aparte de como un emulador del núcleo de red IMS — como un entorno de ejecución de un servidor JEE/SIP rentable para ensayos y pruebas de servicios IMS.

3.2.2 Subsistemas del SDS y características esenciales

El SDS 4.1 FD1, su última versión aparecida durante la redacción de este proyecto, consta de los siguientes subsistemas, indicando entre paréntesis su denominación en la interfaz de la herramienta:

- Entorno de diseño (*Design Environment*)
- Entorno visual (*Visual Network*)
- *Framework* de cliente IMS (ICP y IJCU) (*IMS Client Framework*)
- Plataforma de cliente IMS (ICP) (*IMS Client Platform*)
- Utilidad de cliente IMS JME (IJCU) (*IMS JME Client Utility*)
- Emulador de dispositivo móvil (*Handheld Device Emulator*)
- Emulador de servidor (*Server Emulator*)
- Emulador del núcleo IMS (*IMS Core Network Emulator*)
- Emulador de presencia y gestión de grupos (PGM) (*Presence and Group Management Enabler Emulator*)
- Emulador de *push-to-talk* (PoC) (*Push-to-Talk Enabler Emulator*)
- Emulador de mensajería IMS (*IMS Messaging Enabler Emulator*)
- Emulador de televisión IP (*Internet Protocol Television (IPTV) Emulator*)
- API de servicios para el cliente y el servidor (*Service APIs* (*Server and Client*))
- Entorno de testeo o pruebas (*Test Environment*)

3.3 Futuras versiones y *roadmap*

El objeto de estudio es el *Service Development Studio* (SDS) 4.1, aunque no hay que olvidar que fue concebido con la planificación o *roadmap* que se muestra en la figura 3.3. Aunque con retrasos y modificaciones en su cronograma (la versión 4.1 FD1 ha salido cuando en el *roadmap* está planificada la 5.0), Ericsson ha sido fiel a las capacidades que incluirá en sus versiones y que se ilustran en la figura 3.3.

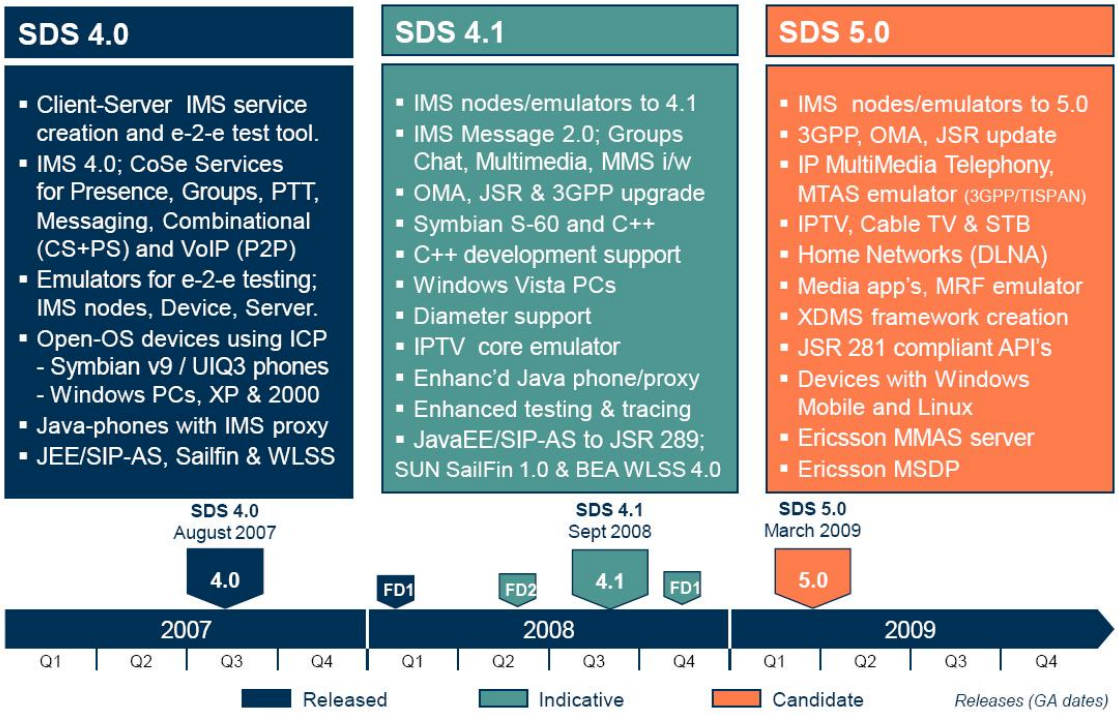

Figura 3.3.: Planificación de las distintas versiones del SDS junto con sus capacidades y estado de *release*. Fuente: Ericsson.

3.4 Subsistemas del SDS y su modo de uso

3.4.1 Entorno de diseño (*Design Environment*)

El entorno de diseño del SDS se ha diseñado para reutilizar prácticas comunes y estándares de la comunidad de desarrollo de Java. Por consiguiente, el entorno de diseño está basado en el IDE gráfico de la plataforma de desarrollo Eclipse [23]. Este entorno para desarrolladores en *Java 2 Platform Enterprise Edition* (J2EE) está constituido por Eclipse 3.4 y la plataforma de herramientas web *Web Tools Platform* (WTP) 3.0, que incluye numerosas mejoras respecto a versiones anteriores. Prácticamente, este IDE contiene todo lo que un programador de Java necesita para desarrollar aplicaciones Java y J2EE. Ericsson escogió Eclipse y las *Web Tools* para su SDS auspiciada por su éxito y por ser considerada por muchos la mejor plataforma de desarrollo en Java.

Algunas de las **características** más importantes proporcionadas por el IDE **Eclipse** se encuentran en los asistentes de proyectos o *wizards*, asistente de codificación, edición Java con compilación incremental, soporte Java EE, editor gráfico de lenguajes web HTML, JSP y JSF, herramientas de gestión de bases de datos, soporte para servidores de aplicación, depuración, testeo, etc. son El SDS ofrece un conjunto de herramientas de última generación que se integran en Eclipse y lo extienden con las siguientes características:

- Asistentes de proyectos SIP/Web dinámicos (*Dynamic SIP/Web Project Wizard*)
- Asistente de creación de *Servlets* SIP (*SIP Servlet Wizard*)
- Asistente de *listeners* SIP (*SIP Listener Wizard*)
- Editor del descriptor de despliegue SIP (*SIP Deployment Descriptor Editor*)
- Editor del descriptor de despliegue HTTP (*Web Deployment Descriptor Editor*) (web.xml) (Característica Eclipse)
- *Debugger* o depurador (Característica Eclipse)
- La construcción y el despliegue (deploying) o *repliegue* (*undeploying*) del proyecto (Característica WTP)
- *Launcher* del conversor de aplicaciones SIP heredadas (*Converting a Legacy SIP Application Simulation Environment Launcher*)
- Visor del flujo de tráfico (Visual Traffic Flow)
- Vista de red (*Visual Network*)
- Entorno de testeos automatizados (*Automated Testing Framework*)
- Agente SIP de testeo (*SIP Test Agent*)
- Vista de cliente IPTV (*IPTV Client View*)
- Asistente de creación de cliente sobre ICP (*ICP Client Application Wizard*)
- *Plugin* para Nokia Carbide C++ para desarrollo de clientes sobre ICP (*ICP Client Development Plug-in for Nokia Carbide.c++ Express*)
- Creación de clientes sobre ICP en Microsoft Visual C++ 2008 Express Edition (*ICP Client Development on Microsoft Visual C++ 2008 Express Edition*)
- Instalación de ICP sobre emulador Symbian (*Install ICP in Symbian Emulator*)
- Instalación del fichero instalador (SIS) (ICP o aplicación cliente) en un dispositivo Symbian (*Install SIS file in Symbian Device (ICP or Client Application)*)
- Lanzar emulador Symbian (*Start Symbian Emulator*)
- Instalar cliente en Windows (*Install Client in Windows*)
- Herramienta de depuración para Windows (*Windows OS Client Debugger*)
- Instalar o desinstalar (*Install (Uninstall) Client in Symbian Emulator*)
- Crear un fichero SIS (*Create SIS File*)
- Herramienta de creación de proyectos IJCU (*IJCU Project Creator*)
- Despliegue de aplicaciones IJCU en el emulador (*IJCU Application Deployment on Emulator*)
- Despliegue de aplicaciones IJCU en teléfonos concretos (*IJCU Application Deployment on Feature Phones*)

- *Javadoc* de la API de la ICP (*ICP API Javadoc*)
- *Javadoc* de la API para el IJCU (*IJCU API Javadoc*)
- *Cppdoc* para la API de la ICP (*ICP API Cppdoc*)

El SDS diferencia entre el entorno de diseño (***Design Environment***) y el entorno de ejecución (***Execution Environment***). El entorno de diseño es la herramienta gráfica con la que el usuario interactúa, mientras que el entorno de ejecución es el conjunto de procesos que, en segundo plano, simulan la red IMS, los servidores de aplicación y los servicios que se alojan en estos. Cuando el entorno de ejecución se instala en un PC con Windows, éste se controla con el entorno de diseño del SDS.

El entorno de diseño se divide en **diferentes perspectivas** o ***vistas*** acordes a los procesos del usuario y sus actividades de diseño:

- *Development Perspective* (un conjunto de perspectivas Java y JavaEE)
- *Provisioning Perspective*
- *Visual Network Perspective*
- *Automated Testing Framework* (ATF) *Perspective*
- *Visual Traffic Flow* (VTF) *Perspective*
- *Debugging Perspective*

El usuario puede cambiar de perspectiva fácilmente desde la ventana de Eclipse.

En el SDS, las aplicaciones IMS de la parte servidora – comúnmente denominado el "servidor" de una aplicación– se crean:

1. Abriendo un *Dynamic SIP/Web Project*
2. Creando un *SIP Servlet*[7]
3. Añadiendo un *SIP Listener* si es preciso
4. Editando el *Deployment Descriptor* si es preciso
5. Desplegando la aplicación en el servidor de aplicaciones[8].

Es posible desarrollar aplicaciones en el SDS acordes a las especificaciones *Java Specification Request* (JSR) 116 [24] y/o JSR 289 [25] definidas por el *Java Community Process* (JCP). La especificación JSR 116 define la API del SIP *Servlet*, mientras que la JSR 289 actualiza las especificaciones de la JSR 116 y define un modelo estándar para la programación de aplicaciones para aunar SIP *Servlets* y componentes Java EE. Para ello, la JSR 289 extiende las funcionalidades de la JSR 116 e introduce nuevos mecanismos de "disparo" o *triggering* basados en un *Application Router* [26] (frente al uso único de las reglas de asociación de los Servlets, las *Servlet Mapping Rules*).

[7] Recordar que un **Servlet** es, *grosso modo* un programa o aplicación que se ejecuta en un servidor (se entiende un servidor de aplicaciones, o AS).

[8] El *Application Server* (AS) del SDS 4.1 de estudio es el servidor de aplicaciones **SailFin**, integrado con el AS Java EE de código abierto **Glassfish**. Es decir, SailFin extiende el servidor de aplicaciones Java Glassfish para que soporte funcionalidades del protocolo SIP. SailFin se basa en un código abierto donado por Sun, Oracle y Ericsson.

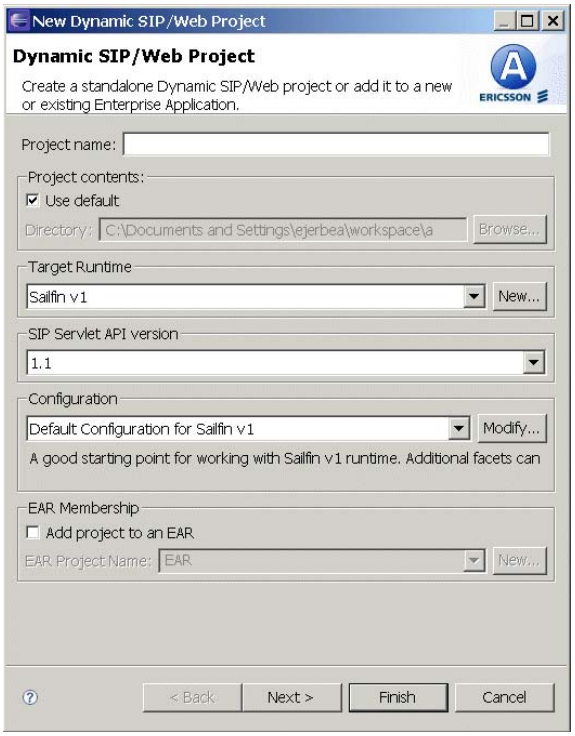

Figura 3.4.: Ventana del *Dynamic SIP/Web Project Wizard*

Las reglas para el *triggering* de aplicaciones descritas en la JSR 289 se encuentran en el *router* de aplicaciones, de tal forma que quedan externas a la aplicación SIP. Con el *router* de aplicaciones, las *mapping rules* pierden peso en la JSR 289. Además, no hay que olvidar que la JSR 289 hace uso de *annotations*, que son tipos de metadatos recogidos en tiempo real.

■ El asistente o *wizard* para un proyecto SIP/Web dinámico: *Dynamic SIP/Web Project Wizard*

En el SDS, las aplicaciones se crean por medio de asistentes que integran la plataforma WTP. De este modo, haciendo *click* en el icono de *Dynamic SIP/Web Project Wizard* de la barra de herramientas del SDS, aparecería la ventana de la figura 3.4 del asistente para la creación de un proyecto SIP/Web dinámico.

Como se observa en la figura 3.4, en el asistente es posible seleccionar la API para seguir las especificaciones de la JSR 116 (opción SIPServlet API v1.0 Converged Project) o la JSR 289 (opciones SIPServlet API v1.1 y Default Configuration for SailFin v1).

El campo de ***target runtime*** contiene la selección del servidor de ejecución donde las aplicaciones se ejecutarán, en este caso, como se ha comentado previamente, el AS por defecto es el SailFin v 1.0.

A través del botón *modify* se puede seleccionar las *facets* de la *Web Tools Platform* (WTP). Estas *facets* son extensiones de la plataforma de herramientas web WTP

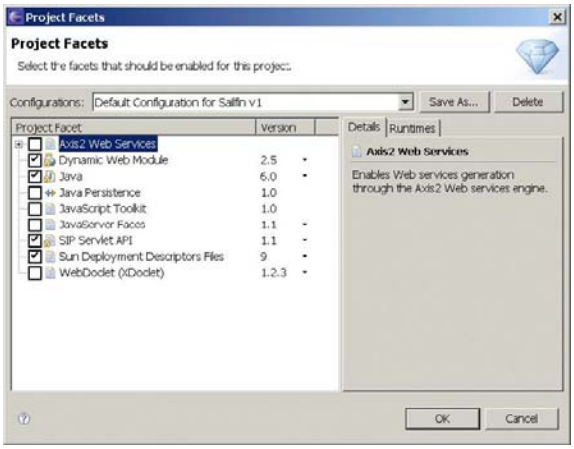

Figura 3.5.: Ventana de *facets* del proyecto

para añadir o restar funcionalidades al proyecto. Existen *facets* para J2EE, web y algunos otros tipos de aplicación. El SDS define, además, una *facet* adicional para SIP. La ventana *Project Facets window* muestra la configuración actual y las *facets* disponibles para el proyecto. Para proyectos SIP como los de IMS, las *facets The Dynamic Web Module*, Java, y SIP *Servlet* API son obligatorias, y como se ilustra en la figura 3.5 están seleccionadas por defecto.

Finalmente, los proyectos se empaquetan automáticamente en ficheros *SIP ARchive* **.sar**, pero es posible compilarlos en ficheros *Enterprise ARchive* **.ear** para ejecutarlos como aplicaciones (empresariales) Java.

■ El asistente o *wizard* para crear un SIP *Servlet*

Para crear un SIP *Servlet* es necesario haber creado previamente un proyecto SIP/Web dinámico, tal y como se describe en el paso anterior. A partir de ahí, se selecciona el proyecto y haciendo click en el icono del *wizard* se pasa al asistente para crear un SIP *Servlet*

Este asistente genera un esqueleto de un SIP *Servlet* basado en las necesidades del usuario. A partir de este esqueleto, el usuario añade la lógica asociada al servicio que desee implementar. El *servlet* generado se añade automáticamente al proyecto que se creó previamente.

El usuario hará uso de la página de métodos (*methods*) y recursos (*resources*) para seleccionar los métodos que desee incluir en el *servlet* y los recursos estándar que utilizará el *servlet*. Además, el usuario podrá definir las condiciones bajo las cuales se ejecutará el *servlet* haciendo uso de las **SIP *Servlet Mapping Rules***.

■ El asistente o *wizard* para crear un SIP *Listener*

El *wizard* para la creación de SIP *Listeners* permite, al igual que en el caso de los *servlets*, generar una clase esqueleto para manejar los eventos que sucedan

Figura 3.6.: Definición del SIP *Servlet*

durante la ejecución de las aplicaciones, como por ejemplo eventos de sesión, errores o temporizadores. Para lanzar este *wizard* hay que tener seleccionado el proyecto concreto y hacer *click* en el icono de la barra de herramientas del SDS correspondiente a este *wizard* para *listeners*, ilustrado en la figura 3.7.

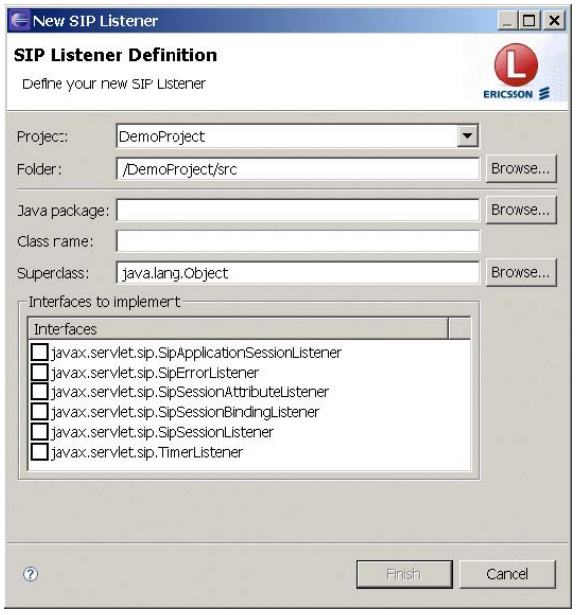

Figura 3.7.: Definición del SIP *Listener*

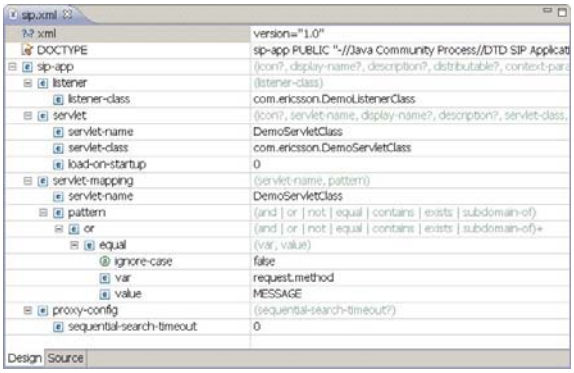

Figura 3.8.: Definición del fichero de despliegue SIP en su *design window*

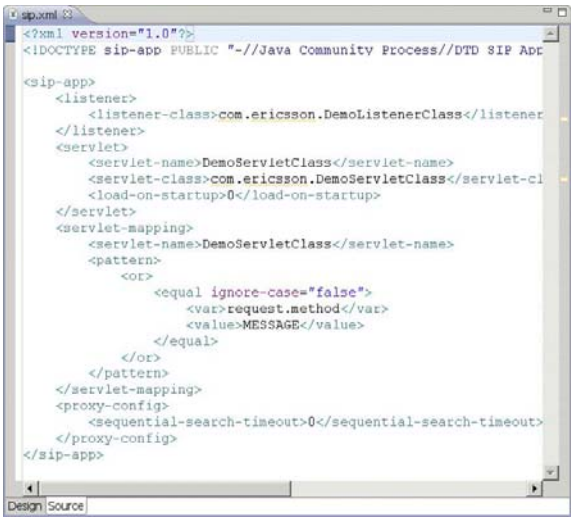

Figura 3.9.: *Source Window*

■ Editor del descriptor de despliegue (sip.xml) (*Deployment Descriptor Editor*)

El editor del *deployment descriptor* permite al programador editar o actualizar el descriptor de despliegue[9] (fichero **sip.xml**) accediendo a éste directamente como fichero de texto. El fichero puede visualizarse bajo la vista de diseño del SDS, donde los elementos del fichero XML se introducen en una estructura que asegura su validez XML (ver figura 3.8). O bien bajo la vista de fuente o *source view*, que permite editar el fichero con total libertad (ver figura 3.9).

[9]La principal función de un contenedor de *servlets* es llevar a cabo el enrutado de las peticiones hacia las aplicaciones. Para permitir esto, hay una serie de reglas asociadas a cada aplicación y especificadas en un fichero XML llamado el **descriptor de despliegue** de la aplicación. Contiene detalles del: nombre del *servlet*, mapeo del *servlet*, parámetros iniciales, condiciones de seguridad y detalles de *login*. Se usan para *servlets* SIP (sip.xml) y HTTP (web.xml).

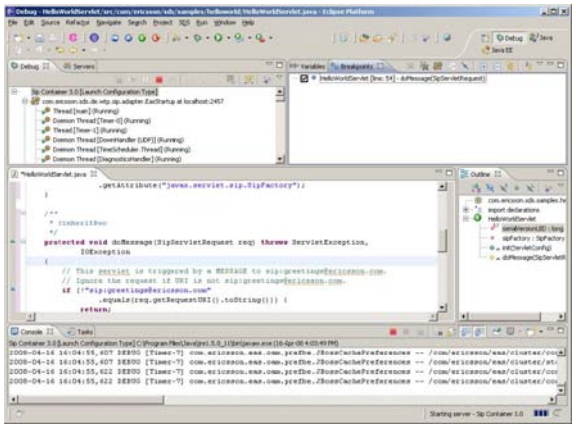

Figura 3.10.: Ventana del *debugger*

■ Editor del descriptor web (web.xml) (*Web Deployment Descriptor Editor*)

Este editor es análogo al anterior, pero referente al despliegue web descrito en el fichero **web.xml**.

■ Depurador (*Debugger*)

El *debugger* permite al desarrollador establecer puntos de ruptura o *breakpoints* en el código fuente de las aplicaciones Java o SIP. La ejecución se detiene al llegar a estos puntos y en ese momento se puede examinar la estructura interna de los datos así como controlar la ejecución de la lógica de servicio paso a paso. El *debugger* es común a todos los IDEs y viene integrado con Eclipse (ver figura 3.10).

■ La construcción y el despliegue (deploying) o *repliegue* (*undeploying*) del proyecto

Para desplegar y ejecutar un proyecto en el servidor de aplicaciones (AS) del SDS (por defecto SailFin) es preciso seleccionar el proyecto con el botón secundario y en el menú desplegado, seleccionar Run As >Run on Server[10]. Se verá más adelante las implicaciones de este hecho en la arquitectura de IMS.

También es posible *replegar* el proyecto desplegado a través de la interfaz gráfica de *servers view*.

■ Convertir aplicaciones SIP heredadas de otras versiones

Debido a que esta versión de estudio, la 4.1 FD1 y más reciente hasta la fecha utiliza WTP como sustrato del entorno de diseño, las aplicaciones SIP creadas con versiones previas del SDS (SDS 4.0 o SDS 4.0 FD1) pueden ser convertidas para su manejo

[10]Es necesario anotar en este punto que es posible lanzar otro servidor JEE/SIP de terceros haciendo uso de la pestaña ***WTP's Server***.

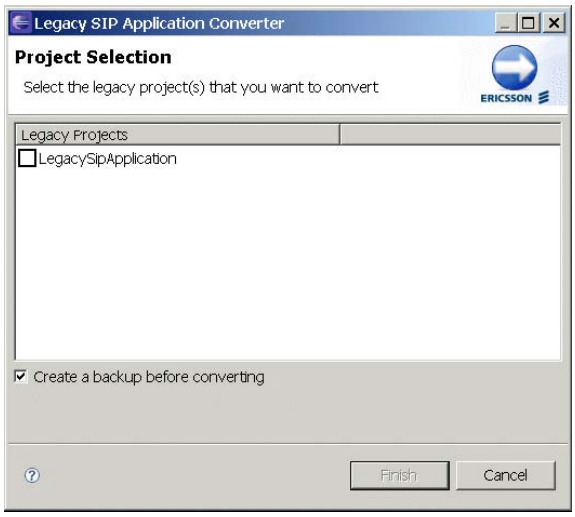

Figura 3.11.: Ventana del conversor de aplicaciones heredadas de otras versiones

en la versión 4.1. Esta conversión se lleva a cabo con el *Legacy SIP Application Converter Wizard* o asistente para la conversión de aplicaciones SIP heredadas[11]. Una vez realizada la conversión, el proyecto no es compatible hacia atrás, de forma que no se puede volver a utilizar en la versión previa donde fue creado[12]. Este *wizard* de conversión está completamente integrado en el IDE con un botón propio en la barra de herramientas que da acceso a la ventana del asistente 3.11.

■ *Launcher* del entorno de simulación y visualización de red (*Visual Network*)

El entorno de simulación (de la red IMS) consiste en el CSCF (integrando el P-CSCF, I-CSCF y S-CSCF, e incluyendo el HSS y el BGCF), un DNS y un servidor de *Push-To-Talk over Cellular* (PoC). Cada uno de estos elementos de la red IMS simulada pueden inicializarse o detenerse individualmente desde el menú "SDS", como ilustra la figura 3.12.

En el modo de visualización de red, *Visual Network* o *Network View* también es posible iniciar o detener la actividad de estas funciones de la red simulada haciendo click con el botón secundario y seleccionando *Start*. Esta perspectiva de visualización consiste en una representación gráfica del entorno de simulación (CSCF, DNS y PoC), así como del emulador de Symbian. Está basada en una *Graphical User Interface* (GUI) que permite seleccionar qué nodos simulados representar, e interactuar con los elementos de manera que es posible cambiar su estado de ejecución y acceder a sus paneles de configuración simplemente haciendo doble click y seleccionando la opción deseada.

[11] La conversión de aplicaciones IMS desarrolladas con la versión 3.1 del SDS no es inmediata ni existe ningún asistente para ello. Para poder reutilizarlas y añadir las funcionalidades de versiones más modernas es preciso reeditar el código teniendo en cuenta la nueva arquitectura de clases del cliente que se verá más adelante.

[12] Para evitar posibles pérdidas a la hora de convertir un proyecto, existe la opción de hacer una copia de seguridad o *backup* antes de completar el proceso de conversión.

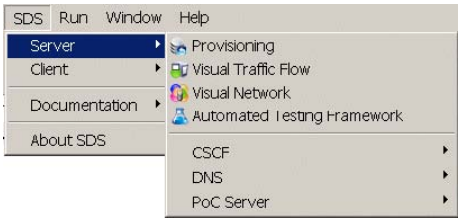

Figura 3.12.: *Launcher* de los elementos de la red para inicializarlos o detenerlos en el entorno de simulación.

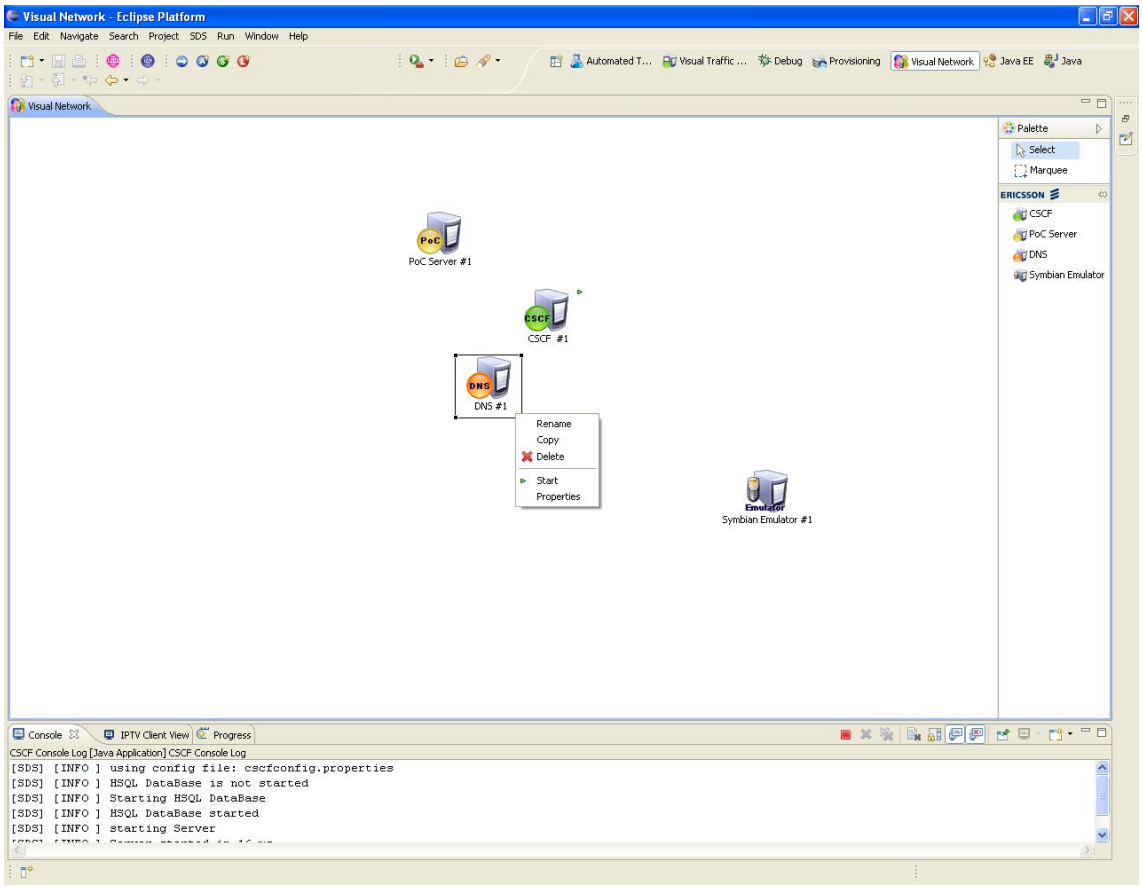

Figura 3.13.: *Visual Network* con todos sus nodos tanto de red como de cliente (el emulador de Symbian) y el menú de interacción gráfica.

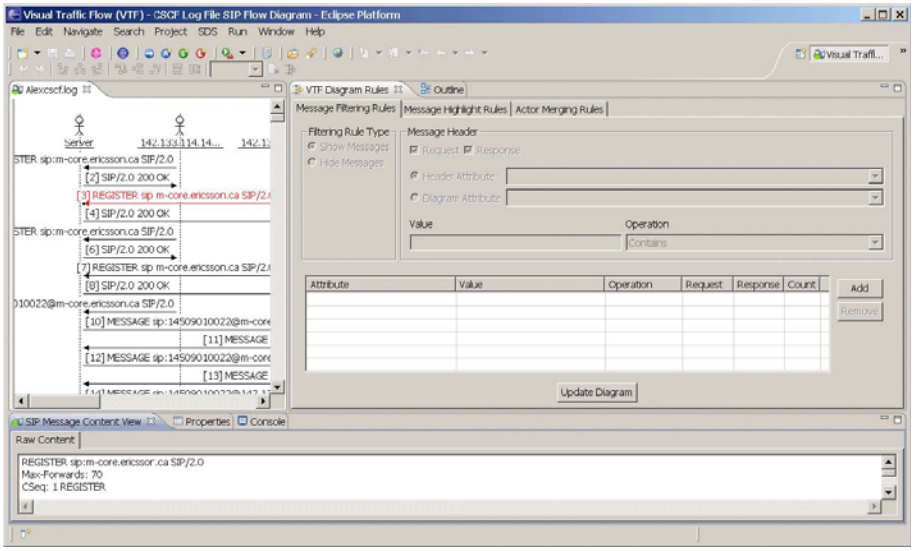

Figura 3.14.: *Visual Traffic Flow*

■ Visualización del flujo de tráfico: diagramas de paso de mensajes

El *Visual Traffic Flow* (VTF) es una herramienta que genera diagramas de secuencias de mensajes SIP a partir de datos almacenados en un archivo de *log* o bien a partir del tratamiento de datos en tiempo real obtenidos del CSCF simulado del SDS. A día de hoy, el VTF sólo procesa tráfico SIP sobre UDP a través del CSCF, por lo que no se soporta aún señalización SIP sobre TCP. Cuando los *logs* se analizan los elementos, mensajes y entradas se muestran como usuarios, diagramas de paso de mensajes y descripciones, tal y como se muestra en el ejemplo de la figura 3.14.

■ El entorno de testeo

En el SDS, desde sus orígenes, venían incorporadas dos entornos para el testeo de las aplicaciones IMS: el SIP *Test Agent* y el *Automated Testing Framework* (ATF). El primero consiste, *grosso modo*, en una interfaz para probar el comportamiento de las aplicaciones ante mensajes SIP concretos elaborados por el programador. El segundo es un entorno que permite simular escenarios concretos de funcionamiento en la red IMS, facilitando la realización de baterías de pruebas. Ambos *frameworks* de testeo se lanzan desde el menú cliente del SDS, ilustrado en la figura 3.15.

El SIP *Test Agent* El SIP *Test Agent* permite testar el comportamiento de una aplicación ante cualquier tipo de mensaje. Proporciona un mecanismo para crear cualquier tipo de mensaje real. Este hecho da lugar a la posibilidad de que el desarrollador teste las aplicaciones y analice sus respuestas a mensajes específicos, ya sean válidos o no válidos. Esta herramienta es flexible para numerosos escenarios. Es posible crear uno o más clientes (*test clients*) que envíen, reciban o respondan a determinadas peticiones SIP. Para crear mensajes SIP correctos, es posible utilizar

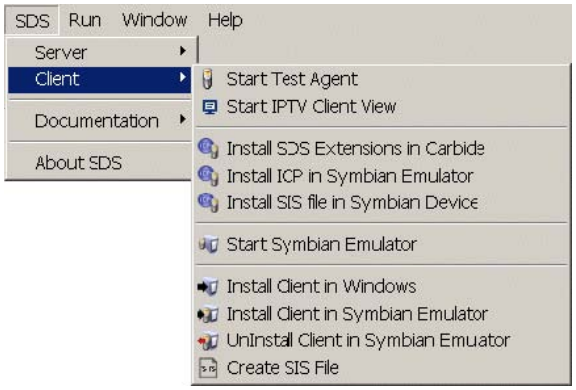

Figura 3.15.: Menú cliente del SDS, con el acceso al SIP *Test Agent* y el ATF.

un generador de mensajes, consistente en un asistente para facilitar la tarea de la creación del mensaje inicial y subsecuentes. Este asistente se encuentra en el botón "New Request" de la parte izquierda del *Test Agent* de la figura 3.16.

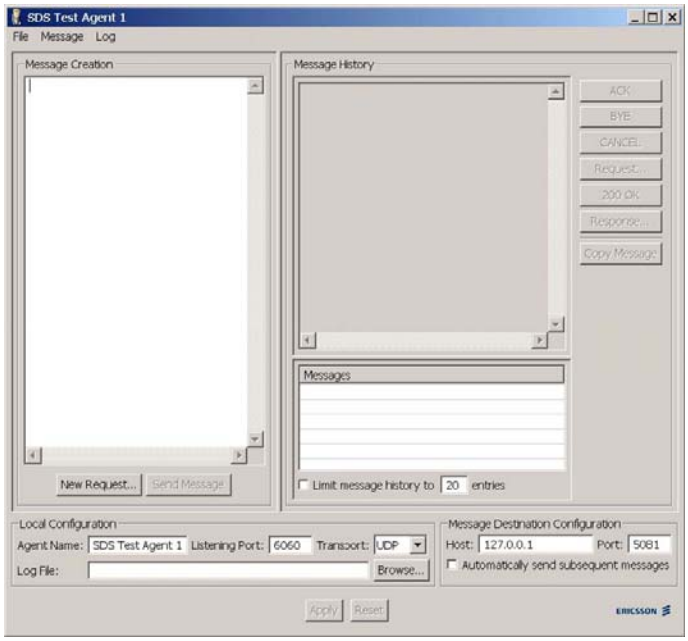

Figura 3.16.: SIP *Test Agent*

El *Automated Testing Framework* (ATF) El ATF se utiliza para crear, editar y ejecutar escenarios de testeo descritos en *scripts* para automatizar la fase de pruebas de las aplicaciones desarrolladas. Simula el comportamiento del cliente, sin la necesidad de programar una aplicación cliente. Queda ilustrado en la figura 3.17.

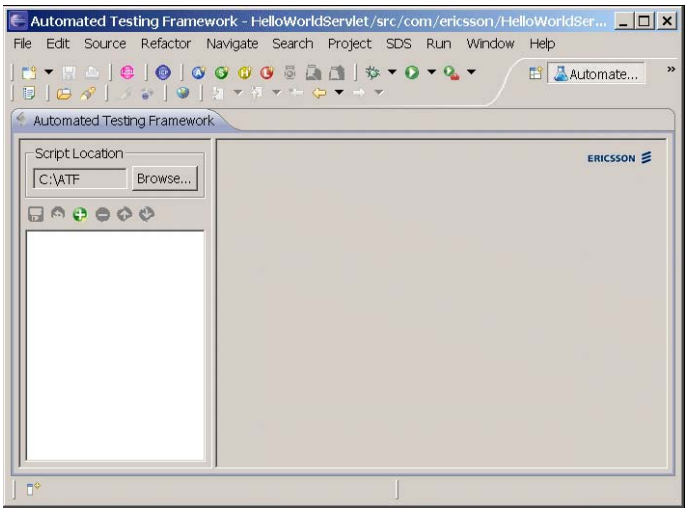

Figura 3.17.: El *Automated Testing Framework* (ATF)

■ Cliente de *Internet Protocol Television* (IPTV)

El SDS, a partir de su versión 4.1 FD1 pone a disposición de los desarrolladores de IPTV el IPTV *Simulator*, un *middleware* que hace efectiva la interacción de aplicaciones de IPTV con la red IMS.

■ El asistente o *wizard* para aplicaciones cliente sobre ICP

El ICP *Client Application Wizard* genera un proyecto marco (*framework* para una aplicación cliente). Para crear la aplicación cliente sobre ICP, es preciso disponer y seleccionar un proyecto sobre el que se quieren añadir capacidades de la ICP en el *Package Explorer*. Posteriormente, se ejecuta el *wizard* 3.18, que proporciona un asistente paso a paso para que el desarrollador seleccione las propiedades que estime oportuno para el cliente IMS que desarrolla.

A lo largo de cada paso del asistente es posible configurar los siguientes atributos:

- Tipo de servicio IMS (esto es, PGM, VoIP, PoC, IMS *Messaging*, servicios integrados o "combinacionales")
- Dispositivo al que está orientada la aplicación (basado en Symbian o Windows)
- Nombre del cliente y su ubicación
- Otros atributos

El asistente de creación de aplicaciones cliente IMS genera la estructura de una aplicación cliente IMS. Los **pasos para la creación de un cliente IMS** pueden resumirse en:

- Crear un proyecto ICP y añadir las capacidades de la plataforma ICP
- Codificar, compilar y depurar

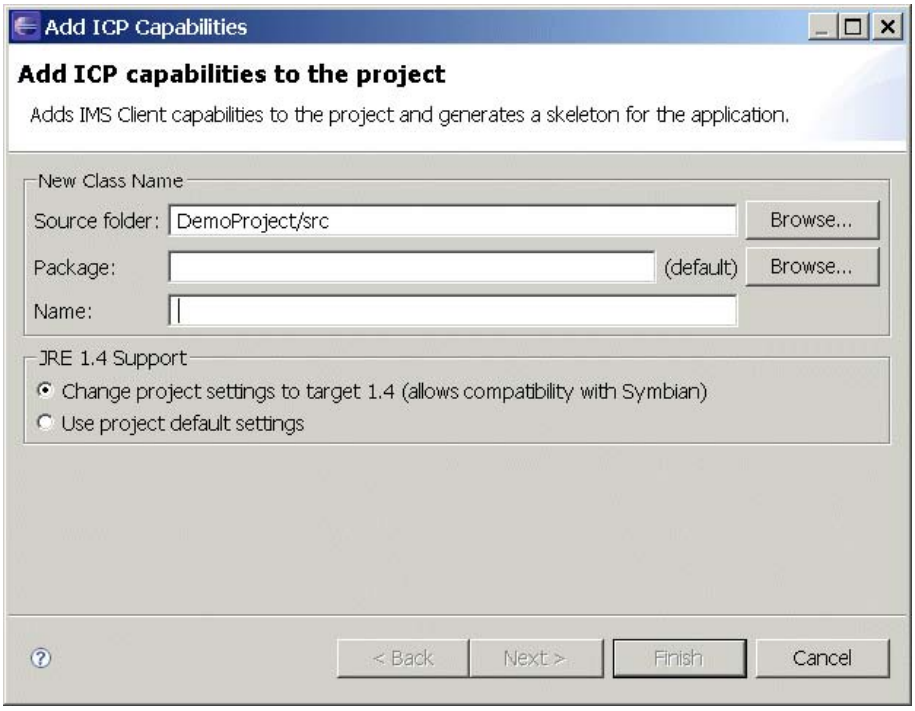

Figura 3.18.: El asistente o *wizard* para aplicaciones cliente sobre ICP

- Instalar la ICP y la interfaz gráfica (UIQ o S60) que mostrará el dispositivo cliente en el entorno objetivo (emulador o dispositivo real)
- Inicializar el entorno de Ejecución: *core*, *Application Server*, *Enablers* de servicio
- Configurar el perfil de usuario (como se vio en el capítulo 2 en lo referente a los *inital Filter Criteria*) de la ICP
- Ejecutar la aplicación
- Llevar a cabo pruebas o testeos extremo a extremo haciendo uso de los nodos IMS del SDS, el servidor y los simuladores de servicios
- Solucionar problemas (*troubleshooting*)

■ *Plugin* para el desarrollo de aplicaciones cliente ICP para Nokia Carbide C++ Express

El SDS proporciona una extensión (un ICP *plugin*) que permite desarrollar aplicaciones IMS sobre la herramienta de desarrollo Nokia Carbide C++ (Express v1.3). Carbide C++ es una herramienta de desarrollo basada en Eclipse para la interfaz gráfica para móviles de Nokia, la **plataforma Nokia S60**. Esta extensión puede añadirse seleccionando SDS >Client >Install SDS Extensions in Carbide.

Esto instalará el ICP *plugin* sobre Carbide y permitirá la creación de aplicaciones IMS para móviles basados en la plataforma S60. Los pasos para crear estos clientes

IMS usando SDS/ICP sobre Nokia Carbide C++ para móviles basados en S60 son los siguientes:

- Tener instalado el SDS
- Registrarse en la web del Forum Nokia y descargar e instalar Carbide.c++ Express v1.3
- Descargar e instalar el SDK S60 para Symbian: S60 3rd edition, FP1 (2006-10-11)
- Ejecutar el SDS, y del menú seleccionar SDS >Client >Install SDS Extensions in Carbide. Esto permite proporcionar la ruta del Carbide instalado, e instala el ICP *plugin* sobre éste.
- Iniciar Carbide integrado en el entorno de desarrollo del SDS (ver figura 3.19). Esta integración añade características como: icono para iniciar un proyecto ICP, un menú para desplegar aplicaciones (*build/deploy*), etc.
- Codificar, compilar y depurar
- Inicializar el entorno de Ejecución: *core*, *Application Server*, *Enablers* de servicio
- Configurar el perfil de usuario (como se vio en el capítulo 2 en lo referente a los *inital Filter Criteria*) de la ICP
- Ejecutar la aplicación
- Llevar a cabo pruebas o testeos extremo a extremo haciendo uso de los nodos IMS del SDS, el servidor y los simuladores de servicios
- Solucionar problemas (*troubleshooting*)

■ Instalar ICP en el emulador de Symbian

Antes de instalar ninguna aplicación cliente es necesario tener instalado y en ejecución la plataforma para clientes IMS (ICP) en el dispositivo de destino. Este entorno de ejecución puede instalarse en el emulador Symbian integrado en el paquete del SDS[13]. Para ello, en el Eclipse IDE hay que seleccionar SDS >Client >Install ICP in Symbian Emulator en el menú que proporciona el SDS dentro del IDE.

■ Instalar un fichero de instalación SIS en un dispositivo Symbian

Los ficheros .sis son ficheros de instalación para dispositivos Symbian, por lo que tanto la ICP como las aplicaciones cliente deberán estar empaquetadas en un paquete de instalación. Las aplicaciones creadas con el entorno de diseño de clientes del SDS pueden desplegarse fácilmente en dispositivos Symbian. Para ello, es preciso generar el fichero .sis para que lance el paquete de instalación, y así posibilitar la instalación en el dispositivo. Esta opción se encuentra en el menú SDS: SDS >Client >Install SIS

[13]La ICP se instala en el sistema operativo Windows con la propia instalación del SDS en el PC.

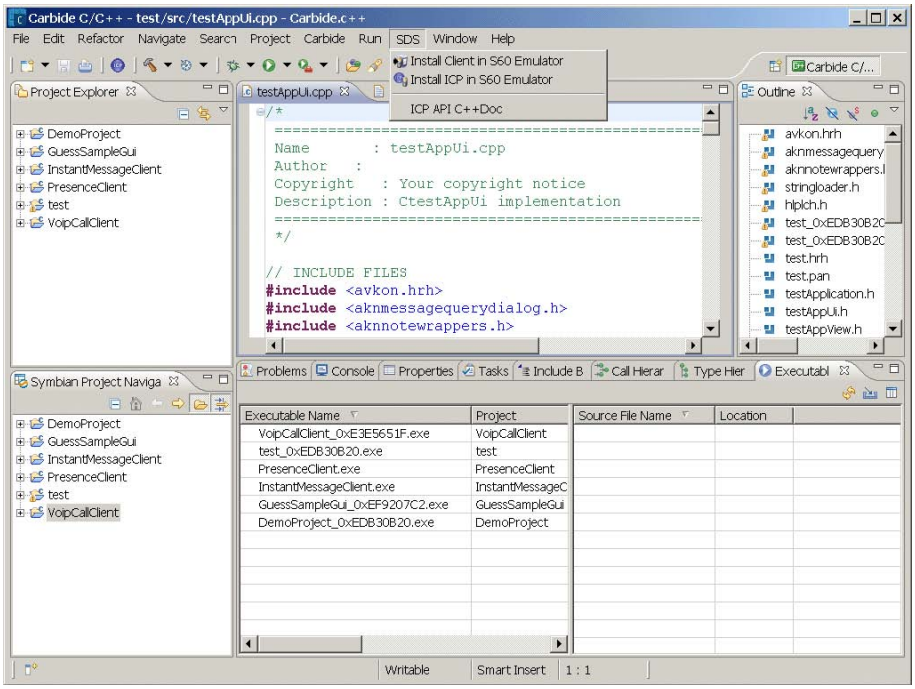

Figura 3.19.: El entorno de desarrollo del SDS con el S60 C++ Carbide integrado.

file in Symbian Device. El despliegue de las aplicaciones en los terminales requiere las *suites* de escritorio propias del terminal. En estas *suites* se presentan una serie de menús, interfaces, asistentes y aplicaciones integradas para facilitar la comunicación con el terminal a través del ordenador para todo tipo de propósitos. En conclusión, el programador ha de instalar la *suite* de comunicación con el terminal, instalar el fichero .sis de la ICP en el dispositivo, para luego instalar el .sis correspondiente a la aplicación cliente.

■ Inicializar el emulador de Symbian

Esta función proporciona un acceso directo incluido en el menú de Eclipse a la opción de inicialización del emulador de Symbian: SDS >Client >Start Symbian Emulator.

El programador puede lanzar el emlador de Symbian para acceder al proyecto cliente que esté desarrollando en ese momento. El emulador escogido es el Symbian UIQ, y se usa para ejecutar y testear las aplicaciones IMS clientes antes de pasar a desplegarlas en un terminal real. Alternativamente, puede accederse al emulador de Symbian desde la *Visual Network*, con doble click en el menú emergente disponible en su icono (ver figura 3.13).

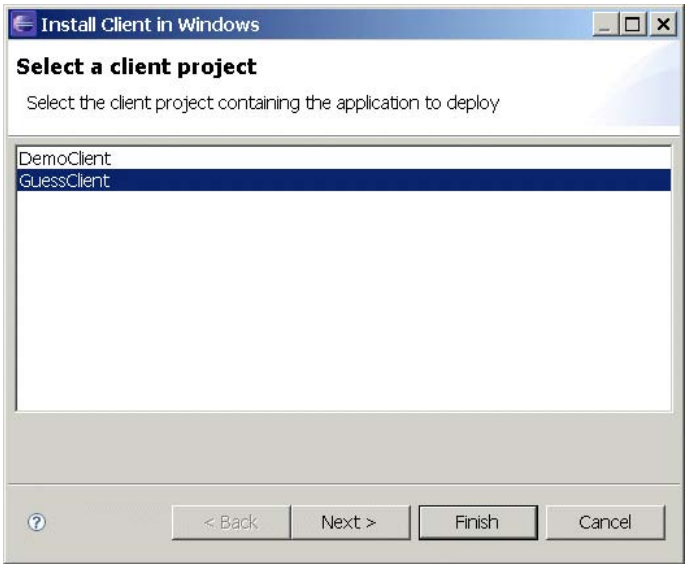

Figura 3.20.: Asistente para la instalación de la aplicación cliente en Windows.

■ Instalar el cliente IMS en Windows

Los desarrolladores de aplicaciones IMS pueden hacer uso de esta opción para ejecutar clientes sobre el sistema operativo Windows. Desde el menú: SDS >Client >Install Client in Windows. En la ejecución del cliente, se puede seleccionar la opción de no ejecutarlo en modo de depuración (modo *non-debug*), para ahorrar recursos del sistema, ya que la opción de depuración puede ralentizar los PCs con menos prestaciones. En la figura 3.20 se ilustra el asistente para la instalación de la aplicación cliente en Windows.

■ *Debugger* para clientes sobre Windows

El "depurador" o *debugger* está habilitado durante toda la fase de desarrollo para los clientes IMS para Windows. Los programadores pueden establecer puntos de ruptura (*breakpoints*), hacerle seguimiento a una determinada variable, etc. Esto proporciona una gran flexibilidad y ahorro de tiempo a los desarrolladores, ya que pueden atacar los problemas y arreglar determinados problemas como *software bugs* de una manera rápida y directa. La aplicación Java que se ejecuta en el emulador de Symbian está completamente aislada del sistema operativo Windows; de ahí que la depuración se soporte en los clientes para Symbian, pero requiere una configuración especial, y no tan directa como la de los clientes en Windows.

■ Instalación o desinstalación de clientes en el emulador Symbian

La opción en el menú para proceder a la instalación o desinstalación de los clientes es SDS >Client >Install or Uninstall Client in Symbian Emulator. El *Client launcher* lanza las aplicaciones IMS al emulador, siempre y cuando la ICP esté activa. El

mismo proceso se lleva a cabo para la desinstalación en el emulador.

Creación de proyectos IJCU

El asistente para la creación de proyectos *IMS JME Client Utility* (IJCU) permite crear aplicaciones IJCU haciendo uso de **Eclipse Micro Edition** (Eclipse ME). En la barra de herramientas del SDS, es posible seleccionar el botón de IJCU para lanzar el *wizard* de nuevo proyecto *Java 2 Micro Edition* (J2ME) IJCU.

Una vez lanzado el asistente, los siguientes pasos para la creación de una aplicación IJCU son:

1. Decidir en qué dispositivo J2ME emulado se va a desplegar la aplicación; configurarlo y cargarle los parámetros Java.
2. Crear un nuevo J2ME Midlet para el proyecto. Se genera un esqueleto para un proyecto IJCU, y se le añaden las utilidades IJCU a la ruta de las clases Java. Existen dos tipos de ficheros .jar disponibles para proyectos IJCU:
   - ijcu-core-4.1.1-me.jar: este paquete *core* implementa la APIs JSR 180 y JSR 281. Es especialmente compacto (500KB) para permitir a dispositivos limitados en recursos enviar y recibir mensajes SIP.
   - ijcu-presence-4.1.1-me.jar: este paquete (700KB) incluye el paquete *core*, así como el soporte para una funcionalidad avanzada de presencia.
3. Una vez generado el proyecto IJCU, pero antes de implementarlo, es preciso usar la clase IjcuUserManagement para configurar parámetros como domainURI, outboundProxyURI, publicUserId, "reino" de claves (*realm*), privateUserId, contraseña (*password* y plataforma (*platform*) [27].
4. Codificar el *midlet* IJCU haciendo uso de los documentos de la API descrita en el IJCU *Javadoc*.

Despliegue de aplicaciones IJCU

Existen dos opciones para desplegar aplicaciones IJCU:

- Despliegue de aplicaciones IJCU en emuladores de dispositivos JavaME: Si se elige esta opción, el despliegue se lleva a cabo haciendo click con el botón derecho sobre la clase del *midlet* IJCU, y del menú contextual seleccionar Run AS >Emulated J2ME Midlet.

  De este modo, se lanzará el emulador por defecto con la aplicación ejecutándose sobre éste.
- Despliegue de aplicaciones IJCU en terminales: El despliegue del *midlet* IJCU varía según qué terminal se esté usando. Un modelo de teléfono específico tendrá que incluir la información necesaria. Como prerrequisito, la red de paquetes que se utilice tendrá que configurarse. Entonces, se seguirán los siguientes pasos para el despliegue de la aplicación:

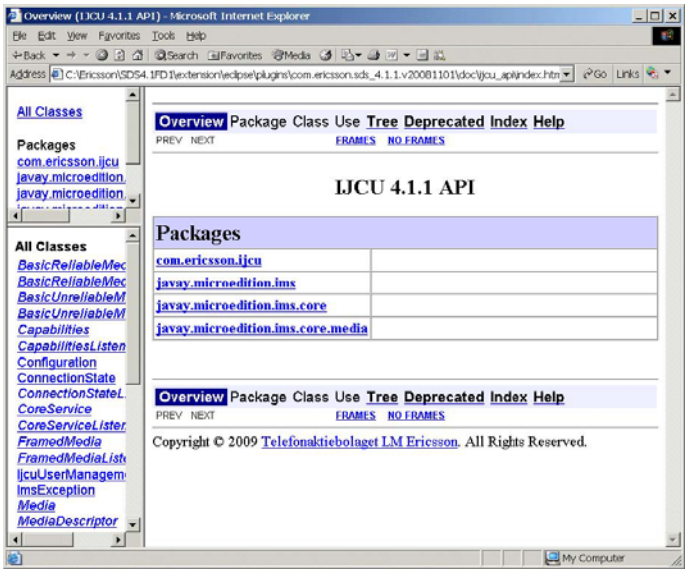

Figura 3.21.: Un ejemplo de Javadoc: La API para clientes IMS sobre IJCU basados en Java.

1. "Subir" el *midlet* (ficheros JAD y JAR) al teléfono, por ejemplo mediante USB o Bluetooth.
2. Instalar el *midlet* seleccionando el fichero JAD o JAR.
3. Ejecutar el *midlet* en el teléfono.

Es posible que haya que firmar la aplicación, dependiendo del modelo de terminal.

■ ICP/IJCU API Javadoc y Cppdoc

Desde el menú del SDS puede accederse a la documentación (SDS >Documentation menu) de las APIs para el desarrollo de aplicaciones cliente sobre ICP o IJCU tanto en Java (el documento se denomina Javadoc) como en C++ (Cppdoc). La documentación consiste en en un manual basado en HTML (ver ejemplo de la figura 3.21) donde se describen todas las interfaces y métodos disponibles para la creación de clientes IMS.

3.4.2 *Framework* del cliente IMS: plataforma (ICP/IJCU) y aplicaciones

■ Origen

El éxito de IMS, como el de otras tecnologías, depende de la disposición de nuevas aplicaciones en el dispositivo final. En el caso de IMS la importancia de los terminales y dispositivos es aún mayor por las siguientes razones:

- Al pasar del tradicional dominio de conmutación de circuitos CS al de

conmutación de paquetes PS, cada aplicación define qué parte se establece en el terminal de usuario, y qué parte en la infraestructura de red. **Un cliente siempre será necesario**.

- Con la evolución tecnológica, los dispositivos son cada vez más potentes y son capaces de ejecutar una multitud de aplicaciones avanzadas. Esto es debido a que cada vez se disponen de procesadores más pequeños, con un menor consumo y una mayor potencia, baterías más compactas y de mayor duración, y una enorme integración de accesorios (cámara, radio, GPS,...) y plataformas de acceso (WCDMA, WLAN, HSPA, Bluetooth, etc.)

En términos empresariales, se requiere una solución madura para asegurar rápidamente unos *Time To Market* (TTM) y *InterOperability Testing* (IOT) (si la aplicación pretende desplegarse en distintos dispositivos), así como, en términos tecnológicos, una implementación en la red y en definitiva generar una aplicación IMS *e2e* atractiva y competitiva.

Por otro lado, el desarrollo y la implementación de aplicaciones IMS en los dispositivos finales no es una tarea fácil. El desarrollador tiene que afrontar la creación de aplicaciones desde **dos perspectivas**:

- **Perspectiva tecnológica**, con requerimientos en:
  - El número de protocolos de comunicaciones a implementar, incluyendo: SIP, SDP, RTP, RTCP, MSRP, XML, XCAP, HTTP
  - Los estándares IMS del 3GPP, IETF y OMA
  - Ejecución en tiempo real
  - Gestión de tareas de bajo nivel
  - Gestión de los recursos del dispositivo (consumo de baterías, tarificación, ...)
  - La propia coexistencia y competencia entre aplicaciones
- El punto de vista del usuario o ***experiencia de usuario***, con una demanda, en general, en:
  - Interfaz (GUI) atractiva, competitiva e intuitiva para el usuario
  - Interacción con el usuario simple y atractiva

Con respecto a estas dos perspectivas, existen a su vez dos problemas básicos:

- La ausencia de desarrolladores que dominen ambas perspectivas
- La interoperabilidad requiere una cantidad considerable de tiempo y recursos, debido a que cada aplicación IMS ya desarrollada incluye su implementación propia de la tecnología IMS

Sin embargo, la interoperabilidad de las aplicaciones IMS es clave para el éxito de IMS, por lo que estas cuestiones han de ser planteadas y resueltas. En este proyecto se ha llegado a la conclusión de que Ericsson es una compañía que ha sabido afrontar estos problemas con la **división de las aplicaciones en dos capas**:

- **Plataforma**[14] **de aplicaciones cliente (ICP/IJCU)**: dominio de los proveedores de plataformas de dispositivos.
- Aplicaciones cliente IMS, los **clientes IMS**[15]: dominio de los desarrolladores de aplicaciones.

El objetivo para esta división fue ocultar todos los detalles tecnológicos de IMS detrás de la plataforma del cliente, y presentar APIs de alto nivel a los desarrolladores de aplicaciones. En este sentido, los desarrolladores de aplicación han de concentrarse sólo en el proceso de innovación, mientras que la tecnología IMS se encapsula en el *framework* y queda oculta a los programadores. Estas dos capas, juntas, crean el dominio *software* del cliente IMS en el terminal o dispositivo.

■ Visión general del *framework* del cliente IMS

En general, la estrategia que ha seguido Ericsson, como líder de las herramientas para el desarrollo de aplicaciones IMS, es la de soportar la estandarización de APIs de alto nivel en la comunidad Java, de tal forma que todos los terminales con soporte Java lo tengan también en IMS, independientemente del tipo de acceso. Esa es la razón por la que Ericsson tomó el liderazgo en la estandarización de la *Java Specification Request* (JSR) 281 y 325 como parte del *Java Community Process* (JCP).

En la figura 3.22 se ilustra la visión del desarrollador de servicios de la arquitectura IMS, siendo el SDS el elemento que ofrece las herramientas para completar el ciclo de desarrollo y testeo **extremo a extremo** sin necesidad de conexión a una infraestructura real. Y se ha remarcado el carácter de extremo a extremo por soportar soluciones tanto de la parte cliente como del lado del servidor: La especificación JSR-116 presenta la SIP *Servlet* API para el desarrollo de la parte servidora del servicio; de la misma manera la pre-JSR-281 ICP API presenta la funcionalidad IMS para el desarrollo de aplicaciones cliente.

Ericsson soporta la versión *embebida* o para descarga del *IMS Client Framework*

- Versión para descarga: *IMS Client Platform* (ICP) es la implementación de Ericsson para descarga en entornos como Symbian, Windows, y en un futuro otros sistemas operativos. La ICP junto con el SDS presenta a los desarrolladores una capa API de alto nivel de las especificaciones pre-JSR 281 y pre-JSR 325. La *IMS JME Client Utility* (IJCU) es la implementación de Ericsson para descarga enfocada a aquellos clientes que a día de hoy soportan el estándar Java *Micro Edition* pero no soportan capacidades IMS o SIP. Las utilidades IJCU a través de la herramienta SDS ponen a disposición de los

[14] En la bibliografía lo han denominado IMS Client Framework. Sin embargo, se ha creído más inteligible la denominación de "plataforma" por su misma definición de sustrato sobre el que se "levantan" aplicaciones.

[15] En este capítulo, donde se habla de un entorno *software* de implementación, hablar de un **cliente** y **aplicación cliente** tiene el mismo valor, puesto que se está frente a un paradigma donde dos partes constituyen un modelo *software* cliente-servidor. Por tanto cuando se habla de clientes IMS se estará haciendo referencia a aplicaciones *cliente* IMS.

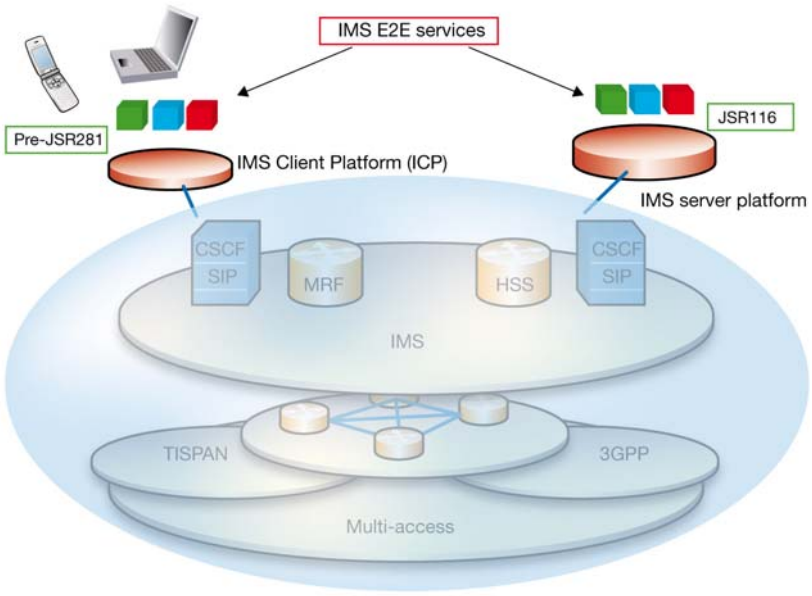

Figura 3.22.: Visión del programador, con la complejidad de la arquitectura de red "oculta" bajo el esquema de desarrollo de servicios.

desarrolladores una API de alto nivel de la versión definitiva de la JSR 281 *release* 1.0.

- Versión *embebida*: Ericsson proporciona el *framework* para clientes IMS de la *Ericsson Mobile Platforms* (EMP). La API de IMS se integra dentro de la *Open Platform Architecture* (OPA) presente en los dispositivos EMP. La EMP proporciona *chipsets* y *firmware* para fabricantes de terminales – aproximadamente un 40 por ciento de los teléfonos 3G con tecnología WCDMA se construyen sobre plataformas EMP, con ejemplos como Sony Ericsson, Sharp o LG.

Ericsson ha definido el *framework* del cliente IMS para soportar:

- El desarrollo rápido y sencillo de aplicaciones cliente
- La descarga de nuevos clientes IMS
- La coexistencia de múltiples clientes
- La integración versátil con las capacidades del dispositivos
- La integración con los *enablers Communication Services* (CoSe) de la red IMS: Presencia y grupos, PTT, mensajería, etc.

Figura 3.23.: Esquema de la implementación de la ICP en la arquitectura de capas de IMS.

3.4.3 La plataforma de clientes IMS, la *IMS Client Platform* (ICP)

■ Visión general de la *IMS Client Platform* (ICP)

El SDS también dispone de un conjunto de herramientas para el desarrollo y testeo de clientes IMS: la *IMS Client Platform* (ICP). La ICP es una implementación del *framework* del cliente IMS, y permite a los desarrolladores de aplicaciones pueden crear proyectos de clientes IMS de tal forma que sólo necesiten concentrar sus esfuerzos en el propio negocio de la aplicación cliente. El programador no necesitará conocer los detalles de implementación de los protocolos, la interacción con el servicio, etc. Los programadores podrán ejecutar y testear las aplicaciones cliente sobre la ICP.

El diseño de la *IMS Client Platform* (ICP) se ha realizado desde una perspectiva extremo a extremo. Los servicios IMS requieren un cliente presentado a través de una GUI; así como es necesario que incluya la lógica de servicio, y funcionalidad para el descubrimiento del *User Equipment* (UE) en la red y enrutado a través de ésta — en un sentido, trasladar la lógica de servicio a la red —. Los servicios IMS se distribuyen a través de la infraestructura de red IMS y requieren un acercamiento extremo a extremo consistente en lo referente al lado del dispositivo. La ICP refleja la arquitectura de la parte de infraestructura IMS mediante la creación de una arquitectura horizontal donde donde se incluyen el *core* y los *enablers* de servicios. Las aplicaciones *end-to-end* se ejecutan sobre estas dos partes. La ICP se ha creado en esta perspectiva con una filosofía de capas, como se ve en la figura 3.23.

La ICP convierte el despliegue de nuevos servicios en una tarea más sencilla, debido a que el núcleo de funciones ya se encuentra accesible para el cliente, como por

ejemplo la interactividad de la aplicación cliente con la infraestructura de red IMS. La ICP está estructurada de tal forma que numerosas aplicaciones cliente pueden convivir en el mismo dispositivo y hacer uso de dicho núcleo de funciones: La ICP proporciona el entorno de ejecución necesario para que múltiples clientes simultáneos reutilicen funcionalidades comunes de IMS.

El diseño de la ICP está orientado a conseguir el acceso y la colocación del cliente en la red. En la práctica, esto se consigue a través de la API de la ICP, que permite el acceso a los recursos multimedia y de comunicaciones. Las aplicaciones cliente realizan peticiones de recursos de manera independiente unas de otras, incluso en el mismo dispositivo, donde no tienen conocimiento de las peticiones de otros clientes anejos.

La ICP asigna flujos de medios, puertos de comunicaciones; negocia con otros servicios y establece sesiones. Cada cliente "posee" los recursos en uso, siempre y cuando la sesión permanezca activa. Cuando la aplicación cliente o alguna de sus sesiones se termina, la ICP libera automáticamente todos los recursos comprometidos durante la sesión. Esta funcionalidad "de acceso" la proporcionan los *managers* de la capa de control de la ICP, que se encargan de distribuir mensajes a las diferentes funciones de IMS y clientes.

La implementación de la ICP asegura un comportamiento acorde a las especificaciones de IMS: inclusión de funcionalidad SIP, negociación de medios y pilas de protocolos, abstracción de los detalles de la red IMS – como por ejemplo la apertura y gestión de conexiones – mediante el acceso a la API de alto nivel pre-JSR 281[16] del *core*. Asimismo, la utilización de la ICP obedece las directrices del ni vel de aplicación, ya que encapsula todos los mecanismos e interacciones necesarias para la implementación de los *enablers* de servicios, y de una manera similar permite expone a los desarrolladores la API pre-JSR 325 de servicios. Este conjunto de APIs predefinidas, la del núcleo o *core* (pre-JSR 281) y la de servicios (pre-JSR 325), está disponible para los desarrolladores de aplicaciones cliente. Con el uso de la ICP, estos clientes pueden crear y controlar conexiones multimedia conforme a las especificaciones del IETF y 3GPP del estándar de IMS.

En la figura 3.24 se puede observar las dos partes en las que se divide la *IMS Client Platform* (ICP):

- Una API para el *core* basada en los estándares del IETF y 3GPP
- Una API específica para los servicios IMS, basada en los estándares OMA: OMA PoC, OMA Presence and GLM (PAG), OMA XDM, y OMA IM

[16] La ICP expone a los desarrolladores dos APIs: la ICP C++ API y la ICP Java (JME/CDC) API. La Java API es una variante temprana de la especificación JSR 281, por ello el nombre de pre-JSR 281.

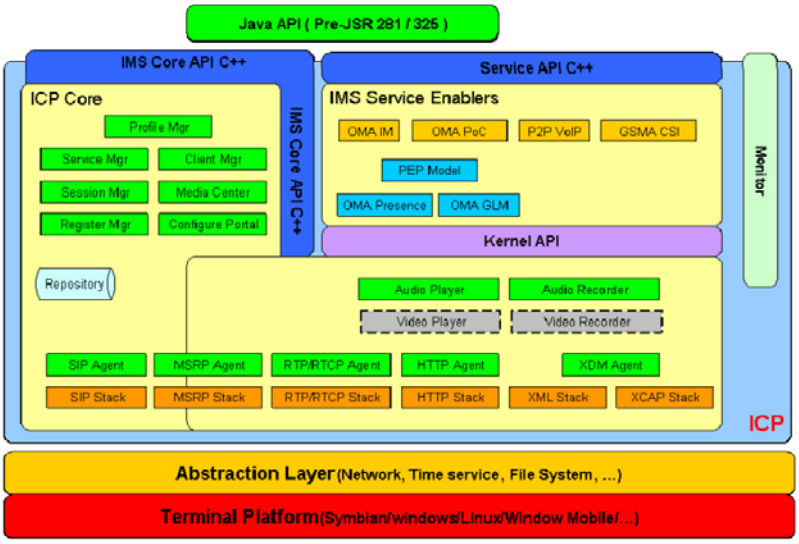

Figura 3.24.: Esquema general de la arquitectura de la ICP versión 4.1. Fuente: Ericsson.

La ICP deja expuestos tanto el *core* como los *enablers* de servicios a través de una API de alto nivel en C++ y Java. La ICP estaba en gran parte basada en el estándar JSR 281 y pretende, en futuras versiones, ser totalmente compatible con el mismo. El desarrollador de clientes solo tiene que implementar la nueva lógica de servicio, así como la nueva interfaz de usuario basándoe en las APIs disponibles. Todos los clientes que hagan uso de la API de la ICP, se ejecutan obligatoriamente sobre el *core* entorno de ejecución de la ICP.

La ICP 4.1 está actualmente disponible para dispositivos móviles y de sobremesa. No existen diferencias de funcionalidad entre la ICP instalada en uno u otro equipo, de hecho, el mismo código base se ofrece tanto para los terminales móviles Symbian (SEMC y Nokia N95) como para los equipos de sobremesa con Windows (XP o Vista). Esta solución versátil asegura la interoperabilidad entre diferentes dispositivos, sin necesidad de efectuar modificación alguna en la implementación funcional. El componente de la arquitectura que permite que la ICP se ejecute en diferentes sistemas operativos es la capa de abstracción o *abstraction layer* (ver figura 3.25) específica para cada dispositivo. Esta capa de abstracción encapsula todos los componentes que son específicos de la plataforma del dispositivo y asegura que la ICP en sí es la misma e independiente de sobre qué dispositivo — *smartphone*, PC o PDA — se está ejecutando o accediendo.

En la figura 3.26 se observa una versión simplificada de la arquitectura horizontal completa de la ICP 3.26 .

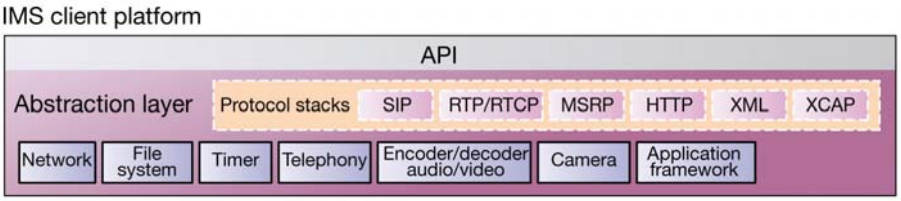

Figura 3.25.: Componentes detallados de la capa de abstracción o *abstraction layer*.

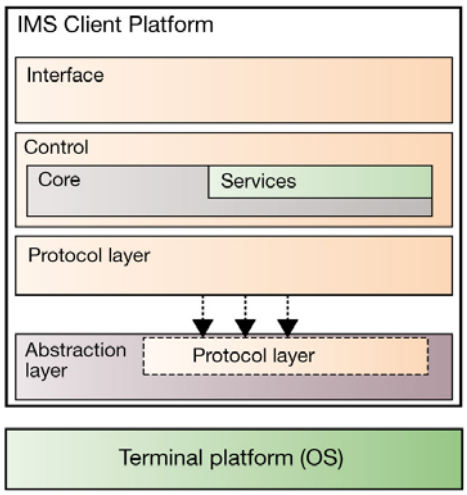

Figura 3.26.: Esquema simplificado de la arquitectura horizontal de la IMS *Client Platform*

■ Características de la *IMS Client Platform* (ICP)

El objetivo de la ICP es proporcionar las características necesarias para simplificar el desarrollo y ejecución de aplicaciones IMS sobre todos los dispositivos (de plataformas abiertas) haciendo uso de la tecnología IMS disponible en la actualidad y continuar simplificándola de cara a futuras aplicaciones. El diseño de la ICP se centra en:

- Desarrollo sencillo de aplicaciones
- Interoperabilidad de las aplicaciones
- Disponer de *enablers* comunes para los servicios estandarizados
- Soporte *multicliente*
- Soporte *multidispositivo*
- Soporte *multidominio*

Respecto a la funcionalidad, esta nueva versión de la ICP (la 4.1) ofrece funcionalidad en dos segmentos: *core* o núcleo de red y *enablers* de servicios:

- Funcionalidad de *core* IMS para la creación de servicios IMS propietarios, incluyendo:
  - Gestión de la sesión (*session management*)
  - *PacketMedia* basado en *Message Session Relay Protocol* (MSRP)
  - *StreamMedia* basado en *Real-time Transport Protocol* (RTP)/*RTP Control Protocol* (RTCP)
  - Reproductores/grabadores de audio
  - Reproductores/grabadores de video
- *Enablers* de servicios, entre los que se incluyen:
  1. Servicio de presencia del estándar de *Open Mobile Alliance* (OMA)
  2. Servicio de grupos y gestión de listas, también de OMA
  3. Agenda/guía telefónica con servicio de presencia añadido (*Presence Enhanced Phonebook* (PeP)). Sobre este servicio de presencia se colocarían los servicios de gestión de grupos y listas
  4. Mensajería instantánea (OMA IM)
  5. *GSM Association* (GSMA) *Combinational Services*
  6. VoIP (*peer-to-peer* sobre PCs, y dispositivos SEMC P1 y Nokia N95)
  7. *Push-To-Talk over Cellular* (PoC) de OMA

Los *enablers* de la ICP extienden el desarrollo sencillo de aplicaciones y la interoperabilidad. Además, se encapsulan todas las primitivas referentes a los servicios estandarizados y se expone la funcionalidad de los servicios a través de la abstracción de las APIs de alto nivel. En este sentido, las aplicaciones cliente no incluyen ninguna implementación de las interacciones del servicio,por lo que pueden interoperar fácilmente en el contexto del servicio.

Al contrario que la implementación del *core* de la ICP, que tenía que incluirse en cualquier configuración posible de la ICP, los *enablers* de servicio son opcionales y permiten mucha flexibilidad en la configuración.

Ejemplo de implementación de un *enabler*: Agenda/guía telefónica con servicio de presencia añadido La aplicación por defecto de la aplicación *Policy Enforcement Point* (PEP) corre paralela a la guía de contactos del dispositivo. El implementar por defecto esta aplicación puede servir como punto de partida para el lanzamiento de servicios basados en IMS desarrollados por el operador. Por ejemplo, al disponer de la presencia de tus contactos se ahorraría en señalización de llamadas no descolgadas, o se dejaría al usuario la opción de seleccionar el tipo de llamada (*videollamada*, llamada de voz, mensajería instantánea...) logrando una segmentación y capilaridad mucho mayores con las que obtener más *Average Revenue Per User* (ARPU)[17].

■ Otras características

Adicionalmente, la ICP 4.1 soporta más capacidades IMS (básicas y avanzadas):

- Autenticación IMS
- Registro
- Autorización
- Señalización y compresión (SigComp 2.0)
- Calidad de servicio, *Quality of Service* (QoS). Control del acceso radio

■ Componentes de la ICP

En la figura 3.24 se muestra la filosofía modular de la arquitectura de la ICP.

Componentes internos de la ICP (ICP *Package*) El ICP *Package* incluye todos los componentes necesarios para soportar aplicaciones cliente IMS desarrolladas sobre la API de la ICP (ICP API) e instaladas en los dispositivos que soportan la ICP. Este "paquete" de componentes internos incluye:

- Entorno de ejecución de la ICP (ICP *Run-time*)
- ICP Monitor
- ICP Client Library

El entorno de ejecución El **entorno de ejecución** de la ICP está actualmente disponible para Symbian 9.1/UIQ3.0 y Symbian 9.2/S60, ambos para entorno real y emulado; y para equipos con Windows XP o Vista. La ICP se ejecuta en el dispositivo como un servicio en segundo plano (en *background*) sirviendo como una plataforma

[17]Ingresos medios por usuario.

común para todos los clientes IMS instalados y desarrollados con la ICP API. La ICP se encarga de gestionar todas las interacciones con la infraestructura del núcleo de red IMS y los servidores de aplicación (en el caso de que intervengan). Este entorno de ejecución se ha testado tanto a pequeña escala y en simulación con el *Service Development Studio* (SDS) como a gran escala y en los nodos e infraestructura reales desplegados por Ericsson. En ambos casos se ha demostrado la compatibilidad con los estándares del 3GPP, IETF y OMA.

Todos los clientes ejecutados en el mismo dispositivo comparten una instancia del núcleo del entorno ejecución de la ICP[18]. La ICP contiene las pilas de protocolo básicas, así como los *enablers* de *core* y servicios, responsables de tratar todos las peticiones y eventos que se produzcan en la interacción con los clientes IMS y toda la señalización y tráfico de datos entre la red IMS y el dispositivo.

El ICP *monitor* El **ICP *monitor*** es la interfaz gráfica, la *Graphical User Interface* (GUI) que interactúa entre el usuario del dispositivo y la ICP. Ésta ofrece las siguientes funcionalidades al usuario final:

- Información acerca el estado del servicio de la ICP en segundo plano
- Arrancar/parar la ICP
- Configuración de los parámetros relacionados con la ICP
- Chequeo de información general de la instalación actual. Por ejemplo, la versión.

La librería de clientes ICP o **ICP *client library*** es un conjunto de DLLs que se carga en el espacio de procesos que la aplicación cliente tiene en la memoria. Estas DLLs encapsulan los mecanismos de comunicación entre la aplicación cliente y la plataforma ICP. Los objetos que se implementan en la ICP quedan de este modo accesibles por las aplicaciones clientes a través de los objetos correspondientes implementados en la librería.

La librería de clientes utiliza un mecanismo cliente-servidor en la comunicación entre las aplicaciones ejecutadas como procesos independientes y el proceso de la ICP. En este sentido, todas las aplicaciones clientes que se ejecutan en el mismo dispositivo comparten el mismo proceso en *background* correspondiente a la ICP.

Componentes externos Los componentes externos de la ICP son:

- **Symbian 9.1 / UIQ 3.0 SDK**:

  El **Symbian 9.1** es, como se ha comentado con anterioridad, el sistema operativo para terminales móviles, y la interfaz gráfica basada en este sistema operativo por defecto en el SDS es la **UIQ 3.0**. La combinación del sistema operativo y esta interfaz conforman el *Software Development Kit* (SDK) que proporciona a los desarrolladores un conjunto de APIs genéricas con

[18]En un dispositivo, sólo se puede lanzar una ejecución de la ICP simultánea.

las capacidades del Symbian OS/UIQ. Este SDK incluye el emulador de un terminal con la interfaz UIQ para el testeo de los desarrollos. Este SDK es abierto, sin control de licencia y se encuentra integrado con el SDS como parte del paquete SEMC P1 CDC. Se instala durante el proceso de instalación del SDS y permite a los desarrolladores crear aplicaciones cliente IMS para el entorno Symbian/UIQ, ejecutables tanto en el entorno real como emulado.

- **Symbian 9.2 / Series 60 3rd Edition FP1 SDK**:

  Siguiendo con el entorno del sistema operativo del terminal, se estudia el conjunto Symbian 9.2/S60, siendo Nokia Series 60 (S60) la plataforma que proporciona la interfaz gráfica. La combinación de Symbian 9.2 con la interfaz S60[19]. El SDK pone a disposición del programador todas las herramientas necesarias para construir aplicaciones Symbian C++, pero es necesario el uso del Nokia Carbide.c++ IDE para hacer más sencillo y accesible el desarrollo con el SDK. El SDK contiene un emulador de un dispositivo con la interfaz S60 para el testeo de las aplicaciones desarrolladas sin necesidad de certificado. Por tanto, el paquete Carbide IDE, junto con el *plugin* para implementar la ICP, permiten el desarrollo sencillo de aplicaciones cliente IMS para el entorno Symbian/S60, ejecutables tanto en el entorno real como emulado.

- **JDK 1.6**:

  El ***Java Development Kit*** **(JDK) 1.6** es el SDK para el lenguaje de programación Java proporcionado por Sun Microsystems para, en este caso, desarrollar aplicaciones cliente IMS en entorno de Java en Windows (es decir, con la máquina virtual (JVM) sobre Windows[20].

- **Nokia Carbide C++ v1.3:**

  Como se ha anticipado, el Nokia Carbide.C++ v1.3 (Express Edition) es un IDE para el sistema operativo Symbian OS. Carbide, con el SDK pertinente es un novedoso conjunto de herramientas de desarrollo de Nokia para la nueva generación de terminales, y extrapolando, de las Comunicaciones Móviles. Carbide aúna las herramientas de desarrollo de Nokia en un *framework* común con nuevas características añadidas. En concreto, Carbide.c++ es una potente familia de herramientas construida sobre Eclipse. Existen cuatro ediciones: OEM, Professional, Developer y Express. La edición Express se usa con el conjunto SDS/ICP, ya que es una versión óptima para la introducción al entorno. El desarrollador sólo tiene que instalar el *plugin* de la ICP proporcionado con el SDS sobre Carbide.c++ v1.3[21]. A partir de ahí, será capaz de desarrollar aplicaciones IMS.

- **Otros componentes externos**:
  - **Microsoft Visual C++ 2008 (Express Edition)**

[19] Con la instauración de la fundación Symbian (Symbian Foundation), la UIQ dejó de existir, siendo la S60 la opción elegida por la fundación Symbian para la interfaz gráfica de su sistema operativo. UIQ contribuirá a la fundación. [28]

[20] Para ejecutar las aplicaciones IMS en Symbian la versión del JDK que soporta es la 1.3.

[21] Cualquier edición: OEM, Professional, developer y Express.

- **Herramientas de compilación: ARM ®RealView ®Compilation Tools (RVCT) y GNU C Compiler Embedded (GCCE)**
- **PC Desktop suite para terminales móviles**: El **PC Desktop suite para terminales móviles** se usa para instalar los clientes desarrollados en los terminales móviles. Esta *suite* viene incluída en el paquete comercial del terminal. Adicionalmente, pueden descargarse desde el sitio web del fabricante.

■ Visión general de la ICP API

Como se ha mencionado en apartados anteriores, la *Application Programming Interface* (API) de la ICP (la ICP API), oculta la complejidad de IMS exponiendo únicamente primitivas de alto nivel a las aplicaciones.

Los desarrolladores tienen por tanto a su disposición un conjunto de herramientas para llevar a cabo todas las tareas de bajo nivel "en nombre" de las aplicaciones. Estas tareas técnicas de bajo nivel se ejecutan bien por medio de llamadas de alto nivel o automáticamente, sin necesidad de ninguna aplicación involucrada — por ejemplo, al registrar una aplicación IMS instalada en el terminal —. La ICP también gestiona los recursos del terminal y ofrece a los desarrolladores de aplicaciones objetos de comunicación y conexiones de medios para intercambiar distintos tipos de contenido entre aplicaciones y terminales. Asimismo, se permite a los programadores requerir una determinada *Quality of Service* (QoS) de una manera fácil e intuitiva.

La ICP API está dividida en una API del núcleo IMS (*core* API) y API de servicios estandarizados (*service* API). Este diseño constituye un método claro para crear nuevos servicios, motivando el desarrollo de nuevos servicios al mismo tiempo que se facilitan medios de comunicación comunes y ampliamente extendidos, como la mensajería.

En esta sección se tratarán las dos APIs que expone la ICP al desarrollador: la **C++ API** y la **Java API** (JME/CDC) analizando sus principales funcionalidades y paquetes.

La ICP C++ API En la versión 4.1 de la ICP, se soporta la C++ API para permitir el desarrollo de aplicaciones nativas, especialmente para dipositivos con el entorno Symbian v9.2/S60 3rd edition FP1. En el *framework* S60, la API C++ de la ICP es la única opción que tienen los desarrolladores, debido a la ausencia de soporte Java CDC[22]. Las aplicaciones nativas normalmente consumen menos potencia y proporcionan un mayor rendimiento comparadas con los clientes en Java. Estos beneficios hacen de estos desarrollos nativos una opción comercialmente más adecuada para el desarrollo de los clientes.

[22]Recordar que *Connected Device Configuration* (CDC) y *Connected Limited Device Configuration* (CLDC) son las especificaciones para la máquina virtual básica que un dispositivo J2ME debe soportar. CDC es para dispositivos algo mayores, con más prestaciones (*smartphones*, PDAs,...), y CLDC es la implementación más básica y para dispositivos limitados en recursos (teléfonos móviles, "buscas", etc.).

- La **API C++** del ***core* IMS** contiene las funcionalidades básicas para desarrollar servicios IMS a medida. Los desarrolladores IMS en C++ tienen a su disposición las siguientes interfaces:
  - IMS Profile
  - IMS Service
  - IMS Session
  - PacketMedia
  - StreamMedia
  - ...
- La **API C++** de los IMS ***service enablers*** provista por la ICP encapsula la implementación de los siguientes mecanismos del nivel de aplicación:
  - OMA Presence
  - OMA Group List Management (GLM)
  - OMA Instant Messaging (IM)
  - GSMA Combinational Service (CSI)
  - P2P Voice over IP (VoIP)
  - OMA Push-to-Talk over Cellular (PoC)
  - Presence Enhanced Phonebook (PEP)

La ICP Java API En la implementación de la ICP, la API en Java (Java API o API Java) es la capa que envuelve la API nativa en C++. La sintaxis de la Java API refleja la sintaxis de las interfaces de la API en C++ y por tanto programar aplicaciones basadas en la ICP tanto en C++ como en Java sigue la misma filosofía.

Por supuesto, existen diferencias entre estos dos conjuntos de interfaces:

- Todas las interfaces C++ están localizadas en un mismo espacio de nombres *namespace*, mientras que en Java se proporcionan multitud de paquetes.
- Las interfaces C++ se basan en valores de retorno (HResult) para invocar métodos. La API en Java, por su parte, hace uso de excepciones.
- Diferencias de lenguaje: C++ usa punteros y referencias mientras que Java utiliza instancias
- La **API Java** para el ***core IMS*** incluye los siguientes paquetes:
  - com.ericsson.icp
  - com.ericsson.icp.media
  - com.ericsson.icp.multimedia
  - com.ericsson.icp.util
- La API Java para los IMS ***service enablers*** se encapsula en multitud de paquetes de clases que siguen el patrón **com.ericsson.icp.services.xxx**,

cubriendo así cada servicio estandarizado para IMS:

- com.ericsson.icp.services.PGM
- com.ericsson.icp.services.imsm
- com.ericsson.icp.services.combination
- com.ericsson.icp.services.telephony
- com.ericsson.icp.services.telephony.Voip
- com.ericsson.icp.services.telephony.PoC
- com.ericsson.icp.services.pep.model

■ Interoperabilidad

La ICP puede interoperar con redes IMS que cumplan con los estándares, y se ha comprobado con el *core* simulado del IMS y el del entorno real *IMS Common System* 4.1 de Ericsson.

Algunas características de la ICP requieren un nodo o AS específico para interconectarse, como el servidor de presencia, PTT-AS, etc.

La versión 4.1 de la ICP sigue minuciosamente los estándares IMS referentes a la *release* 6 del 3GPP, incluso en algún momento sigue la 3GPP R7. Esta fidelidad con la norma asegura la interoperabilidad a nivel de protocolos. La ICP incluye una implementación de estos protocolos necesarios para los servicios IMS.

En segundo lugar, la ICP toma medidas para la autenticación y autorización del propietario del dispositivo junto con el registro de los servicios IMS disponibles en el terminal.

En el nivel de aplicación, la ICP asegura el establecimiento de la sesión de comunicación incluyendo la negociación de la calidad de servicio en la red mientras establece el canal de comunicación en el plano de datos (contexto PDP en el caso de terminales móviles) y negociación de las conexiones de medios.

Los *enablers* de servicio en la ICP son módulos que implementan un determinado número de **servicios estandarizados** asegurando interoperabilidad.

■ Soporte *multidispositivo* (*Multi-Device Support*)

Una de las principales fortalezas del concepto de ICP para dispositivos de plataformas abiertas es el soporte *multidispositivo*. A través de una capa de abstracción dedicada, la ICP es capaz de ignorar toda dependencia con el sistema operativo (SO). Hoy en día se soportan los siguientes SOs:

- Windows XP/Vista
- Symbian v9.1/UIQ 3.08 (solo Java)
- Symbian v9.2/S60 3rd Edition FP19 (solo C++)

De acuerdo con la planificación comercial o *roadmap* que se vio en la sección 3.3, el soporte *multiplataforma* se extenderá en futuras versiones del SDS con los siguientes SOs:

- Linux
- vxWorks
- Windows Mobile y Mobile Linux (ej.: Android)

■ Soporte *multicliente* (*Multi-Client Support*)

IMS proporciona un marco ideal para servicios multimedia enriquecidos. Como se describió anteriormente, el SDS con la ICP proporciona un conjunto de herramientas para el desarrollo sencillo de aplicaciones cliente. Este hecho crea grandes expectativas de cara al crecimiento de las aplicaciones IMS, enriqueciendo la experiencia de usuario y acelerando la implantación de IMS en los mercados. Estas esperanzas suponen unos requerimientos sobre la ICP para soportar la coexistencia de diferentes aplicaciones cliente en el mismo dispositivo. Esta convivencia de clientes fue una de las principales preocupaciones en la concepción de de la ICP, donde se tuvo en cuenta la necesidad del usuario de ejecutar diferentes servicios de comunicaciones simultáneos. Este caso de servicios de comunicaciones en paralelo es especialmente aplicable en los dispositivos que despliegan una interfaz de usuario "pesada", como dispositivos Windows o Linux.

Hay que aclarar que el soporte *multicliente* es un concepto desligado de la ejecución de aplicaciones *multitarea*. Este área ha de ser cubierta por el sistema operativo subyacente. El soporte multicliente de la ICP concierne los siguientes aspectos:

- Registro de todos los clientes instalados
- Lanzamiento de las aplicaciones cliente apropiadas y enrutamiento de los mensajes SIP desde y hacia estas aplicaciones
- Gestión y coordinación de los recursos de IMS y locales (recursos del dispositivo)

■ Soporte *multidominio* (*Multi-Domain Support*)

En el capítulo 2 se habló sobre el perfil IMS, creado para cubrir la necesidad de que un terminal determinado intentara acceder a diferentes servidores IMS situados en dominios distintos. Cada perfil IMS mantiene su propio estado en un contexto independiente, permitiendo que una o más aplicaciones cliente acceder a un dominio IMS específico. Los perfiles IMS están aislados los unos de los otros, sin interferencia o dependencia alguna, siendo posible la activación de múltiples perfiles IMS simultáneamente. De la implementación de la ICP, un perfil IMS incluiría los siguientes contextos:

- Estado de ejecución
- Identidad de usuario y credencial

- Localización del servidor
- Acceso a la red
- Preferencia de lanzamiento
- Otras configuraciones y parámetros

Asimismo, todos los *enablers* de servicio pueden adjuntarse en cualquier perfil IMS activo por medio de requerimientos del cliente. Esto da una gran flexibilidad a los desarrolladores de clientes y a los usuarios finales, permitiendo una gran personalización en las interacciones con los servidores IMS.

3.4.4 La solución para terminales limitados en recursos: IMS JME Client Utility (IJCU)

Para aquellos terminales que por capacidades y recursos no podrían soportar la *IMS Client Platform* (ICP), Ericsson diseñó la *IP Multimedia Subsystem* (IMS) o utilidad JME para clientes IMS. A continuación se describe su origen y principales características.

■ Visión general de la solución

La JSR 281 –API que mejora la ICP API– se lanzó finalmente en junio de 2008 en su versión 1.0. Sin embargo, aún habrá de pasar un tiempo hasta que la mayoría de los terminales estén adaptados para IMS y puedan soportar esta API. No obstante, en el medio plazo es necesario proporcionar implementaciones de esta *release* de cara a cubrir las necesidades de algunos servicios en terminales aún no completamente preparados para IMS. Esta implementación "temprana" es parte del SDS como una solución conceptual mediante la introducción de una utilidad para el lado cliente, la *IMS JME Client Utility* (IJCU). Esta componente es parte de la herramienta del SDS para el desarrollo de aplicaciones JME y posterior despliegue en un entorno real.

En la figura 3.27 se observa la ubicación de la IJCU en el ecosistema de desarrollo. El SDS incorpora esta utilidad para incluirla en los clientes Java.

El objetivo general final es conseguir que todos los teléfonos y terminales accedan a la arquitectura de las redes IMS y en definitiva estar preparados para IMS. Hoy en día esto no es posible debido a limitaciones técnicas de capacidad (los dispositivos actuales no "hablan" ni "entienden" SIP/IMS). Inicialmente, el SDS proporciona una implementación pre-comercial de la API JSR-281 para probar el uso de la API. Esto permite a los usuarios tener una primera imagen de lo que está por venir para los dispositivos móviles en el futuro. Existe un límite de 100 usuarios IJCU para pruebas y demos.

La implementación de la IJCU reside en teléfonos con capacidad MIDP 2.0/CLDC 1.1 y con requerimientos de memoria pequeños. La IJCU ocupa alrededor de 300-350KB de la memoria de los terminales de destino, y es capaz de comunicarse con la red mediante SIP.

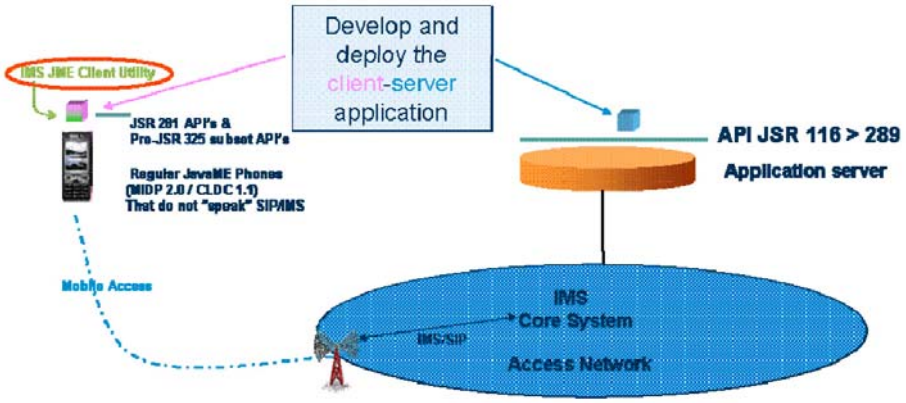

Figura 3.27.: SDS posibilita IMS en teléfonos basados en JME MIDP/CLDC. Fuente: Ericsson.

Los casos soportados de uso, con sus correspondientes métodos SIP, son los siguientes:

- Registro: SIP `REGISTER`
- Presencia: `PUBLISH`, `SUBSCRIBE`, recepción de `NOTIFY`
- Grupos
- Aplicaciones IP: juegos, localización
- Chat: sesión *uno a uno* basada en SIP `INVITE`
- Transferencia de ficheros: envío y recepción de archivos mediante *Message Session Relay Protocol* (MSRP) y JSR 75
- Combinaciones de estos servicios, siguiendo la filosofía de IMS de integración de servicios

■ Entorno de diseño IJCU (IJCU Design Environment)

La versión 1.7.9 de EclipseME se encuentra integrada en el SDS para simplificar y dar soporte al desarrollo de **J2ME *Midlets***. EclipseME proporciona un conjunto de características de cara a facilitar la creación de *midlets*, entre las que se incluyen:

- Crear un nuevo proyecto JME *Midlet*
- Crear un nuevo JME *Midlet*
- Ejecutar/depurar un JME *Midlet*
- Editor *Java Application Descriptor* (JAD)
- Verificación previa de los ficheros de clases
- Soporte para el lanzamiento de *midlets* en ejecución dentro de emuladores JME.
- Ofuscamiento mediante el *software* Proguard

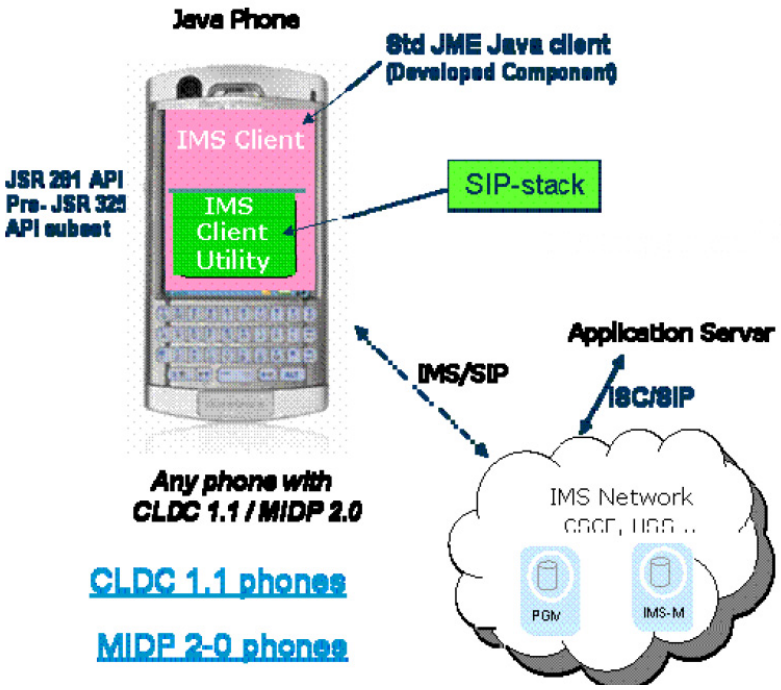

Figura 3.28.: Características de la API JSR 281 soportadas por la IJCU. Fuente: Ericsson.

■ Terminales IJCU

La solución IJCU soporta terminales basados en MIDP 2.0/CLDC 1.1 que permiten la descarga de *midlets* Java. Ericsson ha testado algunos teléfonos y otros los testeará en el futuro. Otros *smartphones* pueden también soportar IJCU, siempre y cuando soporten MIDP 2.0/CLDC 1.1 y permitan la descarga de *midlets*. Los terminales basados en Symbian UIQ también se testearon (M600, P990, P1). La IJCU tendría que funcionar con los terminales Symbian con S60, pero esta configuración no se ha testado aún.

3.4.5 Emulador del núcleo de red IMS

El SDS proporciona un entorno simulado (SDS *Simulated Environment*) donde se simulan los nodos del *core* de la red IMS. Esto permite al SDS actuar como una red IMS virtual desde la perspectiva del desarrollador. IMS, como es conocido, está basado en un protocolo de señalización estandarizado como SIP. Las entidades en el dominio simulado de IMS actúan como el núcleo de red y enrutan la señalización SIP hacia el entorno de ejecución (SDS *Execution Environment*) o bien hacia los servidores de aplicación destino (en su caso) y "disparan" la lógica de servicio requerida.

El entorno de simulación del núcleo de red IMS es, básicamente, un conjunto de funciones lógicas que facilitan al desarrollador de servicios la tarea de testeo funcional de los servicios durante la fase de desarrollo. Determina si se cumplen los objetivos

de negocio tales como *usabilidad* o facilidad de manejo y la idoneidad del producto.

Una gran cantidad de características del servicio pueden probarse sin necesidad de un entorno real o un acceso real a la red IMS. El núcleo de red simulado del SDS 4.1 soporta un subconjunto de funcionalidades de las especificaciones del 3GPP — (TS 24.229/23.228) donde se describen con completitud el estándar del IMS — y del entorno real de Ericsson, el IMS *Common System* (ICS). Esta funcionalidad soportada incluye el registro, autenticación, normalización en la numeración, traducción de ENUM, *triggers* de servicio, enrutado, etc. Estas capacidades se encapsulan en un único componente que lógicamente representa distintos nodos IMS. Este componente está optimizado para un fácil despliegue, configuración y utilización.

El entorno del núcleo de red simulado del SDS comprende:

- *Home Subscriber Server* (HSS)
- *Call Session Control Function* (CSCF), incluyendo sus tres funciones:
  - S-CSCF
  - I-CSCF
  - P-CSCF
- *Breakout Gateway Control Function* (BGCF)
- *Domain Name Server* (DNS), que en el SDS asigna un módulo de nombres de dominio virtuales.

La base de datos de usuario: el *Home Subscriber Server* (HSS) del SDS

El HSS es la principal base de datos donde se almacenan todos los clientes y los datos de los servicios del entorno de simulación. La información almacenada en el HSS incluye entidades de usuario, parámetros de acceso e información de "disparo" de servicios (*service-triggering*). Las identidades de usuario describen los perfiles de usuario, tal y como se vio en el capítulo 2. Los parámetros de acceso se usan para establecer sesiones e incluyen la identidad privada de usuario y su contraseña para autenticar al usuario. La información de *service-triggering* posibilita la ejecución de los servicios SIP, e incluye información que define unas condiciones que determinan si las peticiones las cumplen y proporciona información de encaminamiento cuando los criterios se cumplen.

Las funciones que implementa el HSS en el SDS son las siguientes:

- Soportar el procedimiento de autenticación mediante el almacenamiento de información de autenticación y proporcionando estos datos al S-CSCF
- Proporcionar acceso a los datos del perfil de servicio para su uso dentro del entorno simulado

La información que almacena el HSS comprende tanto datos de usuario permanentes como temporales. Los datos temporales del usuario contienen el estado del registro correspondiente a una identidad pública de usuario. La información perma-

nente se crea, modifica y elimina a través de la interfaz gráfica de usuario (GUI) *provisioning* del SDS. La vista de *provisioning* soporta:

- Dos identidades públicas de usuario: una con forma de SIP *Uniform Resource Identifier* (URI) y otra con formato de TEL URI. En el registro de una de las identidades públicas de usuario la otra queda registrada implícitamente. En el SDS, la SIP URI es obligatoria, mientras que la TEL URI es opcional.
- Una identidad privada de usuario toma forma de un *Network Access Identifier* (NAI)[23] y se asigna permanentemente a una subscripción de usuario. Una identidad privada de usuario comparte un perfil de usuario entre las dos identidades públicas de usuario (representadas por la SIP URI y la TEL URI).
- *Password* de usuario asociada a la identidad de usuario.
- *initial Filter Criteria* (iFC) almacenados para cada aplicación o servicio que las peticiones de los usuarios puedan invocar.
- Perfiles.

***initial Filter Criteria* (iFC)** Dentro del SDS 4.1, los iFC se incorporan de acuerdo a la especificación del 3GPP TS 29.228 R6, que ya se estudió en el capítulo 2. Estos criterios de filtrado se componen de ninguno o un *Trigger Point* y un objeto para el servidor de aplicaciones (AS). El *Trigger Point* está compuesto de uno o varios *Service Point Triggers* (SPTs). Para esta versión del SDS; un SPT puede ser:

- Tipo de sesión *session case*: originada, terminada, o terminada sin registro.
- Método SIP: cualquier método SIP.
- Contenido de la Request-URI.
- Cabeceras SIP:
  - Nombre de la cabecera SIP: nombre de la cabecera SIP que ha de estar presente en una petición.
  - Contenido de la cabecera SIP: una expresión que defina el valor de la cabecera con el nombre especificado con anterioridad.

El *Call Session Control Function* (CSCF)

La función de IMS *Call Session Control Function* (CSCF) lleva a cabo el control de la sesión entre extremos registrados de la red. Se comporta como un *proxy* SIP sin estado, y su comportamiento particular depende de los mensajes SIP que se traten, de su contexto y de las capacidades que se requieran del CSCF para soportar los servicios. Asimismo, el CSCF se comporta como un SIP *registrar*; acepta peticiones de registro y lleva a cabo la autenticación pertinente. El CSCF del SDS es un *proxy* que se comporta como *proxy* sin estado durante la llamada. El SDS presenta una GUI de configuración en la que se pueden configurar los diferentes modos de operación del SDS (ver figura 3.29).

[23] De la forma usuario@reino. Ejemplo: chema@proyectominerva.org

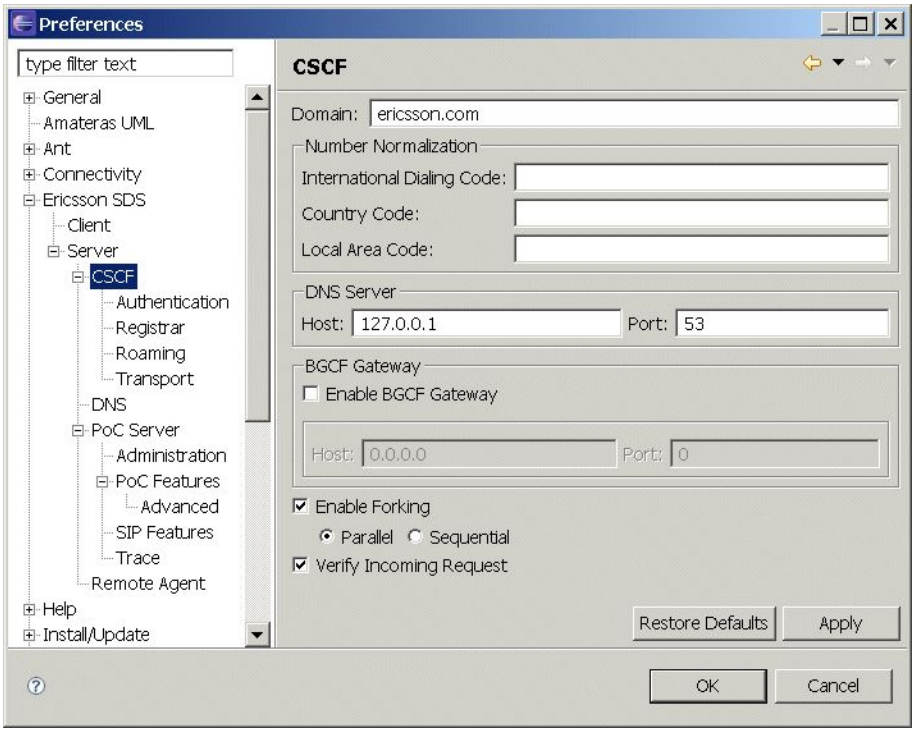

Figura 3.29.: Principales parámetros configurables ("preferencias") del CSCF simulado del SDS.

Cuando el CSCF acepta una solicitud de registro del usuario, los servicios no se "disparan" hasta que no se proporciona un perfil de usuario. El CSCF, por tanto, recupera la información relevante del usuario (el perfil de usuario) del HSS, para llevar a cabo la autenticación entre el usuario final y la red, así como para proporcionar servicios al usuario. El CSCF accede en todo momento a la información del usuario, así como a los datos relacionados con los servicios. Internamente, el CSCF hace uso de habilitadores/deshabilitadores internos para analizar y enrutar las peticiones.

El CSCF incluye un módulo para resolver direcciones DNS para enrutar las peticiones. En particular, este módulo permite al CSCF traducir las SIP URIs asociadas con un número E.164 (número de teléfono internacional). El CSCF aplica el mecanismo DNS/ENUM de acuerdo con la RFC 2761.

En el SDS, el CSCF es un único nodo del entorno de simulación, pero lleva a cabo las funcionalidades descritas en el capítulo 2 y en las especificaciones del 3GPP: funcionalidades *Serving-Call Session Control Function* (S-CSCF), *Proxy-Call Session Control Function* (P-CSCF), *Interrogating-Call Session Control Function* (I-CSCF). Asimismo, incluye la funcionalidad de pasarela hacia redes *Public Switched Telephone Network* (PSTN) de otras redes, la funcionalidad *Breakout Gateway Control Function* (BGCF).

Funcionalidad *Serving*-CSCF El S-CSCF simulado está basado en un protocolo SIP estructurado en capas y soporta un *proxy* SIP bien definido y un *registrar* con autenticación. El S-CSCF se compone de las siguienets funciones:

- Nivel de **transporte**: Soporta el envío y recepción de mensajes SIP mediante TCP y UDP y hace uso de mecanismos multi-puerto
- Capa de **transacción**: Gestiona el tratamiento de las transacciones del servidor y el cliente para un INVITE o non-INVITE de acuerdo a la RFC 3261. La transacción del servidor filtra cualquier petición de retransmisión por parte de la red. Éste recibe las peticiones y las reenvía a los *Application Servers* o directamente al otro extremo de la llamada, dependiendo de los *triggers*
- **Autenticación**: aplicable a todos los métodos. La identidad de usuario privada es la que se usa para autenticar al abonado. Se emplea el esquema *digest* y el algoritmo es MD5.
- ***Proxy***: Esta función incluye:
  - Validación de las peticiones: comprobaciones en el esquema de Request-URI y valor de las cabecera Max-Forwards
  - Mecanismos para la localización de los servidores SIP, según la RFC 3263
  - Record-Routing: encamina las direcciones SIP y entiende una TEL URI en el campo Request-URI de una petición
  - Soporta *forking*[24] SIP según la RFC 3261
- **Registrar**: Soporta las siguientes características:
  - Validación de la petición `REGISTER`: comprobaciones del dominio de Request-URI y el valor de la cabecera
  - Unión o *binding* entre una dirección de contacto con la identidad pública de usuario
  - Gestión de los temporizadores de registro (*desregistrar* al usuario cuando los temporizadores expiran)
  - Adición de una cabecera de Service-Route en la respuesta `200 OK` del `REGISTER`
  - Adición de una cabecera P-Associated-URI en la respuesta `200 OK` del `REGISTER`
  - Adición de la identidad pública de usuario implícitamente registrada (TEL URI) tal y como se establece en el perfil de usuario
- ***Service Triggering***: Gestiona el tratamiento de redirección de la petición inicial hacia el AS si los iFC se cumplen. Esta funcionalidad soporta:
  - Determinación del tipo de sesión, con base a la cabecera Route
  - Tratamiento de las peticiones iniciadas por el usuario "servido": utilización del perfil de servicio del *llamante*

[24] Procedimiento por el cual se generan nuevos procesos "hijos" a partir de un proceso de orden superior o "padre".

- Tratamiento de las peticiones terminadas en el usuario "servido": utilización del perfil de servicio del *llamado*
- Adición de la TEL URI
- Envío de peticiones a la dirección del AS incluída en los iFC
- Incluir una entrada propia en la cabecera Route con el identificador original del diálogo SIP
- Ejecución de los siguientes *filter criteria* de menor prioridad (si las condiciones se cumplen)
- Tratamiento del registro de terceras partes
- Tratamiento de la respuesta

- **Capacidades del *llamante***: Mecanismos por los cuales un agente de usuario SIP (SIP UA) puede trasladar sus capacidades y características a otros agenets de usuario y al *registrar*. Esta información es transportada como parámetros de la cabecera Contact en el `REGISTER`. El *User Agent Client* (UAC) puede pedir las capacidades del usuario *llamado* mediante el envío de `OPTION`. Se soportan las siguientes capacidades:
  - Registro de las capacidades del *llamado* indicadas por parámetros en la cabecera Contact de la petición `REGISTER`
  - La respuesta incluye todos los parámetros de características relacionados con la URI registrada
  - Los parámetros básicos (*base tags*) soportados son: *audio, automata, methods, schemes, application, video, language, type, isfocus, actor, text, extension*. Otros parámetros no mencionados pueden ser añadidos con un signo más ("+"). La definición de los parámetros queda recogida en la RFC 3840
  - No existe validación de sintaxis en los *base tags* o su contenido. El S-CSCF ignora lo que no reconoce.
- ***Preferencias* del *llamante***: Estas *preferencias* son los mecanismos que permiten al usuario *llamante* determinar preferencias sobre el tratamiento de las peticiones en los servidores. Esto se lleva a cabo a través de tres cabeceras de petición: Accept-Contact, Reject-Contact, y Request-Disposition[25].

El S-CSCF hace uso de *enablers* internos para analizar sintácticamente los mensajes SIP y llevar a cabo operaciones sobre el flujo de la llamada. Utiliza el *enabler* de normalización numérica para normalizar un número de teléfono cuando llega en su forma abreviada. Asimismo se basa en el *enabler* DNS para buscar direcciones para encaminar las peticiones. El DNS convierte una URI en un registro DNS compuesto por una dirección IP, un puerto y un protocolo de transporte.

[25] Este es un ejemplo claro del porqué de la elección de SIP en detrimento de otros protocolos como el H.323: SIP presenta una clara facilidad de extensión mediante la inclusión de cabeceras y parámetros.

Funcionalidad *Interrogating*-CSCF El núcleo IMS del SDS está integrado con la función I-CSCF simulada, que actúa como un *proxy* SIP en el borde del dominio administrativo (del operador). Esta función simulada ofrece un conjunto de procedimientos descritos en el capítulo 2, basado a su vez en la sección 5.3 de la especificación TS 24.229 v8.2 del 3GPP.

El I-CSCF del SDS se compone de las siguientes funciones:

- **Registro**: Esta función soporta el registro, *re*-registro y *desregistro* tanto en la red doméstica como en la red visitada
- **Encaminamiento de peticiones**: Las peticiones enrutadas hacia el I-CSCF se gestionan de la siguiente manera:
  - Se permite a las peticiones `REGISTER` de usuarios en itineracia ser encaminadas hacia el I-CSCF de la red doméstica. El *Proxy*-CSCF de la red visitada envía el `REGISTER` al I-CSCF de la red doméstica
  - Se permite el encaminamiento de peticiones terminadas en un dominio perteneciente a otro operador (I-CSCF del dominio destino). El S-SCSF de la red origen envía la petición al I-CSCF de la red destino.
- **Encaminamiento hacia *Public Service Identity* (PSI) destino**: El enrutado hacia el AS donde se encuentra alojado la PSI[26] contenido en la Request-URI. Las peticiones hacia la PSI de destino van directamente hacia el AS que aloja este servicio PSI.
- **Peticiones seguras (*trusted requests*)**: todas las peticiones entrantes se consideran seguras. El procedimiento para que el I-CSCF sepa si la petición viene de un dominio seguro está descrito en la TS 33.210 del 3GPP, pero **esta versión del SDS no la soporta**.
- **Configuración de IP y puerto**: El I-CSCF simulado tiene su propia configuración de dirección IP y puerto, configurables a través de las preferencias del SDS.

Funcionalidad *Proxy*-CSCF La funcionalidad del P-CSCF simulado del SDS soporta las siguientes capacidades:

- **Sobre el registro**:
  - Almacenamiento de la identidad predeterminada del usuario con la cabecera P-Asserted-Identity
  - Almacenamiento de las identidades públicas de usuario que se encuentren en el valor de la cabecera P-Associated-URI

[26] Una identidad de servicios públicos, *Public Service Identity* (PSI); como su nombre indica es un servicio público estandarizado que permite a un operador proporcionar servicios accesibles desde cualquier red SIP. Esto significa que cualquier contenido identificado a través de un PSI es potencialmente accesible por todos los usuarios IMS del mundo, de cualquier operador, e independientemente de si su contrato es fijo, móvil, de cable o de un paquete de convergencia. Esto también significa que cualquier servicio es accesible desde una red no-IMS con conectividad SIP, como por ejemplo Internet.

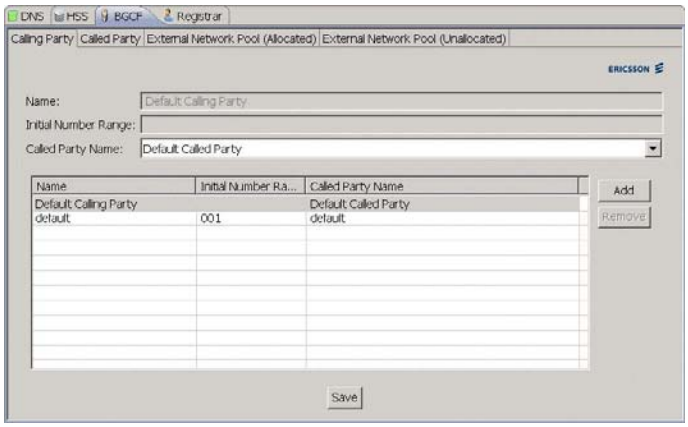

Figura 3.30.: Interfaz gráfica del BGCF simulado

- **Sobre el *desregistro***:
  - El *desregistro* iniciado por el usuario provoca la eliminación de la identidad pública de usuario y de todas las otras identidades públicas asociadas, incluyendo toda la información relacionada almacenada
- **Tratamiento general**:
  - Gestión y tratamiento de la cabecera P-Preferred-Identity (presente o no) con la P-Asserted Identity
  - *Roaming*
  - Soporte para mecanismos de privacidad de acuerdo a la RFC 3325
  - Soporte para la cabecera de registro de caminos: Path, de la RFC 3327

Funcionalidad *Breakout Gateway Control Function* (BGCF) El SDS de Ericsson integró (como se observa en la figura 3.30), **a partir de su versión 4.1** la funcionalidad de pasarela hacia abonados de conmutación de circuitos, basándose en un encaminamiento a partir de números de teléfono.

Por tanto, el *core* de la arquitectura de IMS simulada está integrado con la funcionalidad BGCF simulada, cumpliendo con los principales detalles de las especificaciones TS 23.228 y 24.229 del 3GPP. El BGCF es, básicamente, un servidor SIP que encamina peticiones SIP y TEL URI hacia redes externas. La función BGCF del SDS se invoca cuando el núcleo IMS del SDS recibe una petición SIP dirigida a un número de teléfono y fracasa al traducir el número de teléfono a una SIP URI — cuando falla la consulta DNS pertinente — y es preciso "salir" hacia una red de circuitos PSTN/PLMN a través de una pasarela de salida —bien eligiendo otra red IMS a partir de la cual salir al dominio de conmutación de circuitos (*Circuit Switched* (CS)); bien eligiendo la pasarela hacia el dominio PSTN/CS si el terminal de destino también pertenece a la misma red donde se encuentra el BGCF—. El BGCF del SDS soporta la siguiente funcionalidad:

- **Tipo de sesión (*session case*)**: Esta funcionalidad se invoca internamente

para la parte origen cuando la petición se dirige a un número de teléfono y no existe entrada SDS ENUM en la base de datos para ese número de teléfono, y por tanto no es posible asociarle una dirección IP

- **Petición (*Request*)**: Esta funcionalidad puede invocarse por cualquier petición SIP
- Direccionabilidad: Funcionalidad colocada junto al CSCF (P/I/S) del SDS y no direccionable externamente
- **Encaminamiento**: Gestiona el encaminamiento hacia cualquier nodo configurado (puede ser la función *Media Gateway Control Function* (MGCF) o no). Encaminará llamadas desde un sistema IMS hacia una red externa, especificada por la configuración del SDS para los siguientes casos de tráfico:
  - Encaminar directamente hacia un BGCF externo, que en definitiva gestiona la selección de red externa
  - Seleccionar una red externa (pasarela de salida) para encaminar la petición SIP hacia una PSTN/PLMN
- **Selección de red externa**: Esta función consiste en una SIP URI para pasarelas hacia redes externas (por ejemplo el *Media Gateway Control Function* (MGCF)). Si es necesario, se llevará a cabo una consulta DNS para obtener una dirección IP. La selección de una red externa depende del valor de la cabecera P-Asserted-Identity, del Request URI y de otras cabeceras que se utilizan para realizar consultas en las tablas de configuración del BGCF simulado del SDS. El procedimiento que se sigue para seleccionar la red externa está en consonancia con el nodo destino de Ericsson. Los tres tipos de tablas de configuración de selección externa de red, *External Network Selection* (ENS) son:
  - Tabla de la parte llamante
  - Tabla de la parte llamada
  - Tabla de *pools* de redes externas.

El BGCF del SDS viene con una GUI (figura 3.30) para editar las tres tablas de configuración de redes externas. Esto proporciona acceso a los parámetros de configuración requeridos y relevantes para la selección externa de red.

■ *Domain Name Server* (DNS)

El *Domain Name Server* (DNS) *mapea* nombres textuales y numéricos en direcciones de Internet. El servidor proxy SIP hace uso de procedimientos DNS para resolver URIs SIP o TEL URIs que contengan números E.164 en una dirección IP, puerto y protocolo de transporte del siguiente salto en la señalización SIP. La URI puede ser SIP, SIPS o TEL URI. La lógica de conversión se define en la RFC 3263. El DNS es necesario para resolver dos aspectos del flujo de la llamada:

- El proxy ha de descubrir el servidor SIP en otro dominio, de cara a reencaminar la llamada para el usuario. Debe determinar la dirección IP, el puerto y el

protocolo de transporte de este servidor

- El proxy resuelve la dirección de equipo de las entidades pertenecientes al mismo dominio en dirección IP, puerto y protocolo de transporte en la red.

El SDS hace uso del **DNS BIND server** del 3GPP para dar funcionalidad DNS.

■ El emulador del servidor (*server emulator*)

El SDS, a partir de su versión 4.0 viene haciendo uso del servidor de aplicación (AS) de fuentes abiertas **SailFin**, de Sun Microsystems. El Sailfin AS es un producto de la colaboración entre Sun y Ericsson en la cual Ericsson contribuyó con parte de sus desarrollos en servidores al proyecto *opensource* GlassFish/SailFin. Ericsson cedió su SIP Servlet 1.0 AS, basado en los estándares, al proyecto GlassFish/SailFin bajo la licencia *Common Development and Distribution License* (CDDL). Ericsson tomará los resultados de este proyecto *opensource* que pretende desarrollar un SIP *container* (SSA 1.1) que cumpla con la nueva JSR 289.

El SDS 4.1 integra la *release* SailFin 1.0 *Alpha*. Esta versión *alpha* es conforme a la especificación JSR-116 y parcialmente a la JSR-289. SailFin trabaja sobre una plataforma Windows que proporciona al desarrollador el acceso al JEE/SIP AS en un entorno *hardware* de escritorio. En el futuro, SDS integrará un paquete con la *release* SailFin 1.0 con soporte total de la JSR-289. El SDS con SailFin posibilita a los desarrolladors de servicios programarlos haciendo uso de una API JavaEE/SIP que opera sobre el *core* del protocolo SIP, así como ensamblar y empaquetar los servicios en archivos JEE/SIP listos para su despliegue en entornos reales basados en JEE/SIP *Application Servers* comerciales.

■ APIs de servicios (servidor y cliente)

Una *Application Programming Interface* (API) se emplea por las siguientes razones:

- Proporcionar acceso a los recursos de la red
- Posibilitar el desarrollo de un amplio abanico de servicios
- Permitir que terceras partes, más allá de los proveedores de los equipos, puedan desarrollar servicios de valor añadido
- Proporcionar librerías funcionales de programación a los desarrolladores

APIs del SDS

La API del servidor La API del *servlet* SIP (*SIP Servlet API* (SSA)) especificada en la JSR-116 (SSA 1.0) y en la JSR-289 (SSA 1.1) se usa para la creación de aplicaciones para ejecutarse en servidores de aplicación (ASs) JEE/SIP. La API del servidor del SDS está basada en la JSR-116 y parcialmente en la JSR-289. Esta API es una API Java para desplegar servicios Java basados en SIP sobre el servidor de aplicación de fuentes abiertas SailFin de Sun.

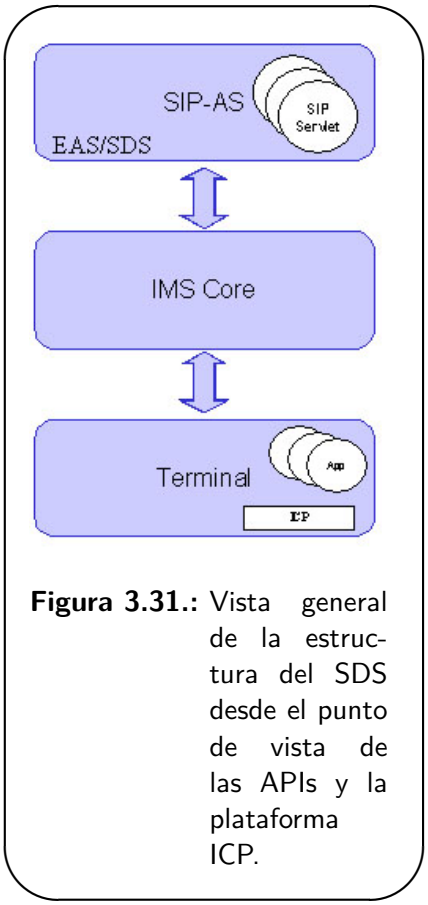

Figura 3.31.: Vista general de la estructura del SDS desde el punto de vista de las APIs y la plataforma ICP.

La ICP API del cliente Como se ha tratado con profusión en la sección 3.4.3, la *IMS Client Platform* (ICP) es el servicio de segundo plano instalado en los dispositivos para todos los clientes IMS. Las aplicaciones clientes acceden al entorno de la plataforma ICP mediante las versiones "tempranas" de dos JSRs, la pre-JSR-281 y la pre-JSR-325 API, tal y como muestra la figura 3.31

Esta ICP API pretende cumplir con la especificación JSR-281 a corto plazo y con la JSR-325 a plazo medio. Como ya es conocido, esta API es un conjunto de librerías de alto nivel para el desarrollo de aplicaciones cliente IMS.

La ICP IJCU del cliente La *IMS JME Client Utility* (IJCU) se instala en terminales JavaME como parte de su paquete de *midlets* (*midlet package*)[27]

3.5 Emuladores de terminales móviles

El SDS 4.1 trae integrado un emulador de terminales con sistema operativo Symbian v9 con interfaz gráfica UIQ 3 o S60 para el desarrollo y despliegue de clientes en

[27]La API de la plataforma IJCU está basada en la JSR-281. Los paquetes y clases de esta API vienen especificados en su correspondiente Javadoc [27]. Están contempladas las actualizaciones de la IJCU hacia nuevas versiones de la JSR-281 y adoptar la JSR-325.

terminales (en concreto denominados *smartphones*) SonyEricsson P1 y Nokia N95. Para los teléfonos más convencionales, es posible añadir al entorno de diseño del SDS/Eclipse varios *plugins* del *Wireless ToolKit* (WTK) de Sun Java. Estos *plugins* permiten a los desarrolladores el acceso a emuladores de teléfonos JavaME.

El emulador de los terminales presenta en la pantalla del PC una interfaz *clickable* correspondiente a las respectivas pantallas/interfaces de los terminales, como se ilustra en la figura 3.32.

En este apartado se ha de aclarar que el sistema operativo Windows puede albergar clientes IMS, siendo el PC con el SDS el propio entorno de ejecución del dispositivo.

Un nuevo cliente desarrollado en el seno del entorno de diseño del SDS pasa a la siguiente fase de testeo con el modo de emulación de dispositivo del SDS, con el emulador Symbian UIQ/S60, o los emuladores JavaME *Wireless Toolkit* o bien el mismo PC para los clientes *Windows*.

Además, la aplicación cliente puede testarse antes de su despliegue en dispositivos reales en su emulador correspondiente, pero contra el *core* IMS simulado del SDS y los *enablers* de servicios de comunicaciones, así como beneficiándose de los *servlets* alojados en el servidor simulado JavaEE/SIP del SDS

Figura 3.32.: Pantallas de los correspondientes emuladores de teléfonos Symbian UIQ, Symbian S60 y teléfonos JavaME.

3.6 Ejemplo de desarrollo de una aplicación IMS con el SDS

A lo largo de este capítulo se ha descrito la herramienta elegida para el desarrollo de aplicaciones y servicios IMS. Mediante una descripción a modo de tutorial del funcionamiento del SDS se han ubicado sus subsistemas más importantes, se han presentado todos los *wizards*, los asistentes de código, y detallado los parámetros

de configuración necesarios para "provisionar" cada elemento tanto de la red como de los dispositivos cliente. Sin embargo, esta guía no queda cerrada ni completa si no se acompaña de un ejemplo que ponga de manifiesto la necesidad del modelo cliente-servidor y su desarrollo independiente, la utilidad de Java y sus APIs, y, por último las ventajas y beneficios del SDS como entorno de desarrollo que añade las capacidades de IMS como valor añadido de los servicios.

Por todo ello, se presenta un ejemplo sencillo pero que pretende clarificar el modelo de comunicación en un servicio, subrayar dónde interviene el lenguaje de programación y describir posibles capacidades y mejoras basadas en IMS.

3.6.1 Modelo y actores del servicio

Este ejemplo consiste en la interacción entre un cliente que bien puede ser un dispositivo móvil — con Symbian para esta versión del SDS — o un cliente Windows; y un servidor situado en la red IMS. La aplicación consiste en un juego muy sencillo: "Adivine un número". El jugador, el cliente, se registra en IMS, y accede al juego. Una vez en la sesión del juego, el servidor le pide que adivine un número. El usuario interactuará con el servidor enviando un número y éste le responderá si ha acertado, o si el número que ha de adivinar es mayor o menor que el número insertado.

Con este ejemplo tan básico se pretende comprender la facilidad de implementar un modelo cliente-servidor en IMS e ilustrar un paso de mensajes SIP básico. Asimismo, el cliente dispone de una *Graphical User Interface* (GUI) sencilla que puede dar idea del potencial de Java en este sentido, creando interfaces adaptadas a dispositivos móviles de una manera intuitiva.

El modelo de comunicación que emplea el servicio puede describirse como el siguiente intercambio de mensajes de señalización SIP:

1. En primer lugar, como en cualquier servicio IMS, es preciso un registro en la red, de tal forma que el CSCF (*via* HSS) ya sepa qué servicios aplicar y en qué servidores de aplicación. Por lo tanto la primera petición será un `REGISTER`. El servidor responderá con un `200 OK`.
2. El juego del ejemplo se lanza cuando el cliente envía un mensaje `INVITE` a una dirección concreta como por ejemplo `adivina@juegosims.com`. El servidor responde con un `200 OK`.
3. El cliente envía un mensaje `ACK` para establecer la sesión del juego. Cuando el servidor recibe el mensaje `ACK` envía un `MESSAGE` para solicitar al usuario que averigüe un número.
4. Si el cliente recibe correctamente el mensaje envía un `200 OK`.
5. En este punto el cliente envía un `MESSAGE` con el número introducido por el usuario. Se abren dos posibilidades:
   - Que el número sea cierto. En este caso, el servidor responde con un `MESSAGE` con felicitaciones en su contenido y un `BYE`.
   - Si el número no es el correcto, el servidor envía un 200 OK y un nuevo

`MESSAGE` solicitando otro intento con un número mayor o menor.

6. El cliente asiente la recepción de este `MESSAGE` con un `200 OK`.
7. Se repetirán los dos puntos anteriores hasta que el usuario acierte. Cuando lo haga, el servidor lo felicitará con un `MESSAGE` y enviará un `BYE`.
8. El cliente responderá a este `BYE` con un `200 OK`.
9. El cliente, a su vez, volverá a mandar un `BYE` para dar la sesión por terminada.
10. El servidor responde con un `200 OK`.

3.6.2 Implementación en Java

Como se ha presentado en la sección anterior, el servicio consta de una aplicación "servidor" y otra aplicación "cliente". El servidor se encargará de aceptar peticiones del cliente y aplicar la lógica del servicio. El cliente, a su vez, mostrará una interfaz gráfica y ejecutará en el "lenguaje" de la comunicación cliente-servidor las peticiones provenientes del usuario.

Como se argumentó en secciones anteriores, el servidor es un *servlet* SIP programado en Java (`Servlet.java`). El cliente será una aplicación Java (`Cliente.java`) tras una interfaz gráfica, dos cuadros de diálogo descritos por las clases `Dialogo.java`, que muestra una petición de número al usuario, y `DialogoMensajes.java`, cuyo cometido es mostrar el mensaje de felicitación cuando se acierta un número. Y por último pero no menos importante, el cliente, por estar destinado a ejecutarse sobre la plataforma del cliente *IMS Client Platform* (ICP) (ver sección 3.4.3) necesita los adaptadores de la ICP para funcionar sobre el dispositivo cliente. Para el presente ejemplo, es posible emplear los siguientes adaptadores incluidos en la herramienta de Ericsson, cada uno descrito por una clase Java:

- `PlatformAdapter.java`: Implementa la interfaz de la ICP *IPlatformListener*, necesaria para crear un *IProfile*.
- `ProfileAdapter.java`: Implementa la interfaz de la ICP *IProfileListener*, necesaria para crear un *IService*.
- `ServiceAdapter.java`: Implementa la interfaz de la ICP *IServiceListener*, necesaria para crear un *ISession*. Los métodos de ISession son los encargados de construir el mensaje SIP `INVITE` para iniciar el juego
- `SessionAdapter.java`: Del mismo modo que los adaptadores anteriores, implementa la interfaz de la ICP *ISessionListener* para capturar los eventos que se incluyen en el cliente:
  - *processError*
  - *processSessionStartFailed*
  - *processSessionMessage*

Todos los clientes montados sobre la ICP requieren uno o más *listeners* para atender a los eventos que se producen en la lógica del servicio. Algunos *listeners* tienen muchas funciones obligatorias. Para hacer uso de esas funciones, aunque

Método SIP	Método Java que lo implementa
INVITE	doInvite()
ACK	doAck()
MESSAGE	doMessage()
BYE	doBye()

Tabla 3.1.: Métodos SIP y su mapeo en Java.

fuera una sola, se deben implementar todas las funciones de ese *listener* para poder usarlas. Para facilitar el desarrollo, el SDS crea **clases de adaptadores**. Estos adaptadores son simplemente una implementación "vacía" o "hueca" de los *listeners*. Los adaptadores proporcionados con el SDS son un método conveniente para implementar todas las funciones de los *listeners* de modo que la aplicación a desarrollar tenga acceso a la función que necesite, y así no sea necesario implementar funciones que nuestra aplicación no usará.

Es posible observar, cómo cada adaptador es necesario para implementar otro, construyendo una estructura de capas sobre cuya superficie se sostiene la aplicación cliente.

No es objetivo de este Proyecto Fin de Carrera analizar línea por línea el código Java que implementa el servicio. Sin embargo, es de interés explicar cómo Java implementa los mensajes SIP que entiende el núcleo IMS. Por ello, se hace una división entre servidor y cliente a la hora de explicar cómo Java se involucra en el desarrollo de aplicaciones y servicios IMS.

■ Métodos SIP en el *servlet*

En la tabla 3.1 se indica cómo el SDS *mapea* los métodos SIP en métodos Java. En el SDS es posible elegir los métodos SIP que se necesitan implementar, y que además, serán en sí mismos la invocación del *servlet*. Es decir, si por ejemplo se desea invocar al *servlet* para lanzarlo con un `INVITE` desde el cliente, es preciso seleccionar la inclusión del método `doInvite()` durante la creación del *servlet*. Al llevar a cabo esta selección, se añade una *mapping rule* por defecto, que habrá que modificar de acuerdo con las reglas de lanzamiento de servicios definida en el HSS.

En el apéndice C.4.4 se muestra el código del *servlet* del juego "Adivine un número" que sirve de ejemplo para ilustrar un servicio IMS.

■ Métodos SIP en el cliente

En el lado cliente, ya se ha introducido que la aplicación se soporta sobre la ICP, que desde la perspectiva Java, la componen "de arriba hacia abajo", los adaptadores de sesión, servicio, perfil y plataforma. Estos adaptadores, hacen de intermediarios a la hora de crear las peticiones y respuestas SIP.

En el ejemplo, se puede ver que para inicializar el adaptador de sesión se crea un

`ISession`. `ISession` es un objeto que representa una sesión genérica y facilita la tarea de implementar el comportamiento típico de una sesión. Es decir, de cara al mapeo de métodos SIP, un `ISession` es el encargado de componer los mensajes `SIP INVITE`.

`IService`, por ejemplo, es una interfaz para manipular sesiones con comportamientos de "búsqueda", incluyendo opciones de envío que descubran las capacidades del otro extremo, subscripciones (método `SUBSCRIBE`) a eventos, publicaciones de un estado (método `PUBLISH`), envío de peticiones `REQUEST`, envío de mensajes, etc. Traducido a Java, un cliente dispondrá de estas capacidades si se ha creado previamente un objeto `IService`.

Del mismo modo, otros adaptadores siguen el mismo patrón, son capaces de construir y analizar mensajes SIP dentro de su lógica a través de este tipo de objetos: `IService`, `ISession`, e incluso algunos adaptadores específicos para servicios definidos como PGM (`IPresence`), VoIP, PoC, IM,... En todos ellos, la "I" indica "interfaz".

Por otro lado `IProfile` es la interfaz principal con la ICP. Proporciona los métodos para las operaciones con la ICP. Una aplicación cliente para la ICP precisa registrarse en la ICP a través de esta interfaz, previa creación de cualquier sesión. No obstante, el objeto `IProfile` no puede existir sin haber creado anteriormente el objeto `IPlatform` sobre el que `IProfile` se soporta.

En el apéndice C.4.4 se muestran los códigos del cliente y de los adaptadores de la ICP usados en el ejemplo.

3.6.3 Creación de valor añadido: capacidades IMS y mejoras

La aplicación de ejemplo es un caso simple de comunicación cliente servidor con las capacidades básicas de registro, una lógica de servicio simple y una provisión sencilla. Sin embargo, IMS ofrece un amplio abanico de capacidades que dotan de valor añadido a muchas de las aplicaciones que se encuentran hoy día por ejemplo en Internet.

De este modo, un simple juego como el "Adivine un número" podría convertirse en un complejo **sistema de lotería**, como el del juego de la Primitiva en España, en el que los jugadores tienen que elegir una combinación de números que se sortean. IMS interviene en el servicio proporcionando valor añadido desde diferentes ángulos:

- Creando un canal de comunicación seguro a partir de la identificación del cliente en la red.
- Ofreciendo un canal *streaming* del sorteo con la interactividad necesaria para participar en directo.
- Desarrollando una ampliación del ejemplo que permita la comunicación PoC con los miembros de una peña en apuestas conjuntas.
- Incluyendo precios más asequibles a ciertos clientes de la operadora, que a su vez se vería beneficiada por un nuevo sistema de fidelización.

Queda patente pues, la gran versatilidad de IMS, al proporcionar un marco con cabida para prácticamente cualquier proveedor relacionado con Internet y/o las telecomunicaciones. A su vez, el SDS queda refrendado por ser un aceptable simulador de la red, con la gran ventaja de la experiencia de Ericsson en el desarrollo *software* de librerías Java y *hardware* de servidores y *racks* de equipos de telecomunicaciones.