Escola Politècnica Superior de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Advanced orchestration services

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: David Rivas Cordero

DIRECTOR: Toni Oller Arcas

DATE: June 30th 2009

TITLE: Advanced orchestration services

AUTHOR: David Rivas Cordero

DIRECTOR: Toni Oller Arcas

DATE: June 30th 2009

Overview

Nowadays, it is known that the success of new telecommunications networks is directly related to the success of offered services that enhance their use. The arrival of the IP Multimedia Subsystem (IMS) promises helping service providers to deploy a complete array of services on real-time, customized business and consumer multimedia services over any access network.

This project describes a new service plain for IMS to give support to the service development. This new service plain is a new platform, the orchestration platform, offered by the network operator to allow the easy development of new and complex services for professional and nonprofessional developers, and to provide a better time-to-market.

On other hand, there is a clearly evolution of Internet applications to the web 2.0 concept based applications, which enhances the role of the user into the applications.

Also, this project presents the adaption of the IMS provisioning and service enablers to the orchestration platform, and conserves the network interaction for the orchestration services with the aim of making a powerful platform and to promote the service web 2.0 concept based applications development.

Finally, to see completely the platform and network functionality, the project contains also an application over the orchestration platform and also a client side application developed on .NET for Windows Operative System and Windows Mobile Operative System.

INDEX

FIGURE INDEX

INTRODUCTION

In the last years the telecommunication business model has changed. Before the services and connectivity were supplied by the telecommunications operator and in the present time there is the service provider to offer services and the network operator to offer connectivity. In the current business model the network operator requires good services from service provider to success in business.

Nowadays, the client side in the networks has acquired more importance. The clients are becoming more powerful due to the increase of computation power and the increase of the bandwidth. This fact makes available the possibility of emerging new business logics where the client is not only a service consumer, but also he can become a service provider. In a future it can become a P2P service network.

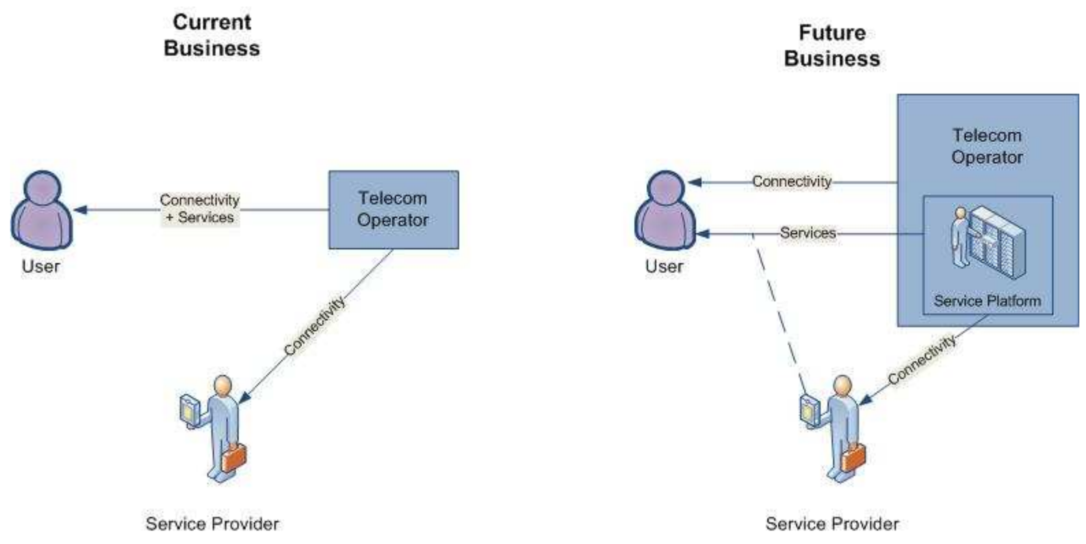


Figure 1 Business logic changes

The Next Generation Network (NGN) considers important the availability of new support business logics and the promotion of the development of new services to increase the usage of the new networks. One example of this is the IP Multimedia Subsystem (IMS) which supports the current business logic.

In the application development field the new Web 2.0 concept model has been successful. Nowadays, the most extended Internet applications are the Web 2.0 [1] concept-based technologies.

The Web 2.0 refers to a second generation of Web development and design, which makes easier to secure communicate information sharing, interoperability, and collaboration on the World Wide Web. Web 2.0 concepts have led to the development and evolution of Web-based communities, hosted services, and applications such as social-networking sites, video-sharing sites, wikis, blogs, mashup and folksonomies.

The concept of Web 2.0 is based on a group of concepts:

- The information is not offered by an only application, instead the users are who share their own information to other users.
- The users collaborate between them to have a richer use experience.
- There is not only static content, also there are multimedia content.
- The user uses the Web browser as like own computer desktop, launching complex logic applications throw any type of terminal.
- In the programmer side is extended the free code APIs and Frameworks use.
- The graphical user interface changes and becomes dynamic. Appear the new technologies like Flash, AJAX, XML and CSS to contribute the designers function.

The best known example applications of the Web 2.0 concept are the following:

The RSS: The Rich Site Summary (RSS) [2] is a family of Web feed formats used to publish frequently updated works in a standardized format. An RSS document includes full or summarized text, plus metadata such as publishing dates and authorship.

The Blogs: Blogs [3] are a type of Website, usually maintained by an individual with regular entries of commentary, descriptions of events, or other material such as graphics or video. Entries are commonly displayed in reverse-chronological order. The relation between blogs and the Web 2.0 concept is the user shared comments of the content.

Social Networks: A social network [4] is a social structure made of nodes (which are generally individuals or organizations) that are tied by one or more specific types of interdependency, such as values, visions, ideas, financial exchange, friendship, sexual relationships, kinship, dislike, conflict or trade.

Objectives

This project is part of a work of a new platform to develop new services for a Spanish telecommunications company. The global project is developed in collaboration with i2CAT Foundation. The i2CAT Foundation is a non-profit organization whose aim is to promote research and innovation in advanced Internet technology, and promotes the deployment of services and wideband applications from both public and private research and innovation communities.

The main objective of the project is the development of a platform to increase the new application publishing and development for the NGN; concretely for the IMS network.

The solution is the Orchestration platform. This platform must improve three essential parts: a directory of services where the third parties can publish his services, an orchestration engine which provides the capability of creating new applications through the direction of published services, and finally the Interface that provides access to the platform for the final user.

The success of this platform depends on a group of characteristics:

- Easy to integrate with IMS
- Provide network interaction for new services development.
- Provide service interaction, including different vendors.
- Provide an easiy, quick and flexible integration of applications.
- Easy to use for service publishing.
- Easy to use for service development.
- Easy to manage.

In the project objectives there is also included the development of a new orchestration platform application as an example of the project function. Having in consideration the success of Web 2.0 applications as described before, the chosen application is a micro-blogging called Twimser.

Twimser is also an integration example between the IMS network resources and the orchestration platform, because it requires of a present service, the Presence Wrapper.

The Presence Wrapper is a service based on IMS Presence Service. Also, the Presence Wrapper is developed as an orchestration platform service to provide presence service for new developed services. The aim of the Presence Wrapper is also to bring the Web 2.0 concept near to IMS.

Document description

In this document we start explaining the technological scenario imposed bases. These bases are related to the Next Generation Network election to host the orchestration platform. The bases are three: first we explain the NGN chosen, after this, there is a wide explanation of the simulation network studio and most used resources, and finally the technological requirement for the development of the network client side.

Once introduced the scenario we have the orchestration platform described where an overview of the requirements, the functionalities and the architecture can be found.

After this we have the explanation of the orchestration platform developed service, Twimser. In this part we show the service developed, the features, the use cases, and finally the architecture.

The next part of the document is dedicated to the core of Twimser, the Presence Wrapper. This wrapper is an important orchestration service for the development of Web 2.0 based services on the IMS through the orchestration platform.

After the developed service, we show the client side developed application for Windows and Windows Mobile Operative System.
Finally the last part of this document is a conclusion from the developed work.

CHAPTER 1. THE CONTEXT

In this part, the technological situation of the project is shown. Firstly we will explain the IMS, the telecommunications network imposed by the telecommunications company. After this we have the software development tool for the IMS platform, the Ericsson Software Development Studio. This tool is also chosen by the telecommunication operator because it is more recommended for the real network integration. In this part, it is also explained that the Presence Enabler is an important part for the orchestration platform services. Finally, there is an explanation of Session Initiation Protocol (SIP), which is the Protocol required by the IMS client development.

1.1. The IP Multimedia Subsystem

The IP Multimedia Subsystem (IMS) [5] is a global, access-independent and standard-based IP connectivity and service control architecture that enables various types of multimedia services to end-users using common Internet-based protocols.

IMS uses the Session Initiation Protocol (SIP) for session setup and teardown, while Diameter is used as an AAA (Authorization, Authentication and Accounting) protocol.

The IP Multimedia Subsystem was initially developed by the Third Generation Partnership Project (3GPP). Its original formulation (3GPP R5) represented an approach to delivering "Internet services" over GPRS. This vision was later updated by 3GPP, 3GPP2 and TISPAN by requiring support of networks other than GPRS, such as Wireless LAN, CDMA2000 and fixed line.

The IMS network architecture has been defined to be access independent. This means that the access technology used to transport user SIP messages to the IMS network does not influence the functionality of the IMS network itself. Consequently, any access can be used.

1.1.1. Architecture

IMS decomposes the network infrastructure into separate functions with standardized interfaces between them. Each interface is specified as a "reference point", which defines both the protocol over the interface and the functions between which operates. The standards do not say which functions should be used, as this depends on the scale of the application, and a single device may contain several functions.

The 3GPP architecture is split into three main planes or layers, each of which is described by a number of equivalent names: Service or Application Plane, Control or Signaling Plane, and User or Transport Plane. Figure 2 shows the design.

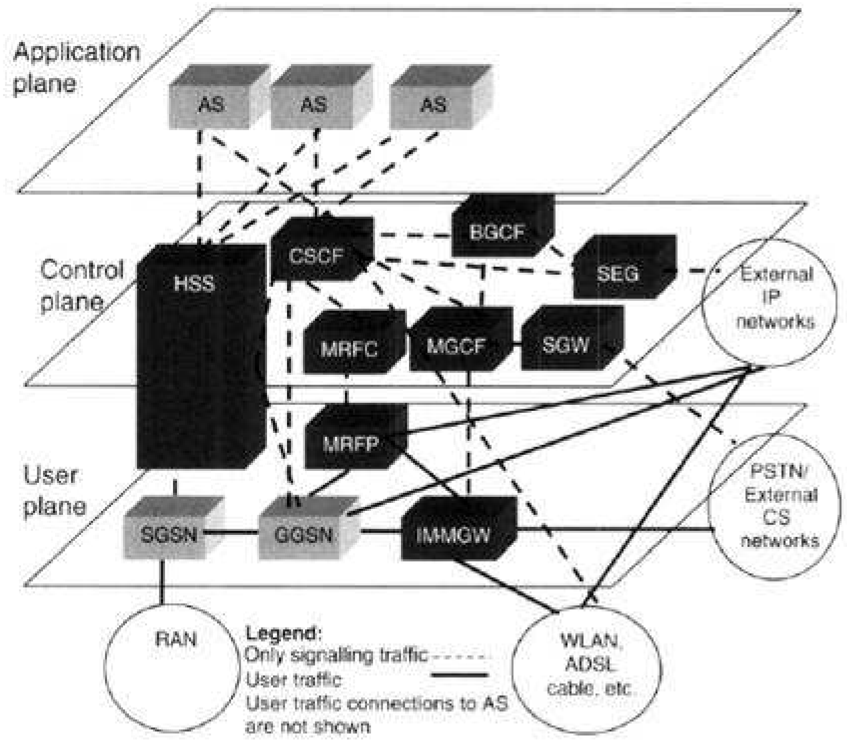


Figure 2 IMS and layering architecture

Application plane: The application plane provides an infrastructure for the provision and management of services, and defines standard interfaces to common functionality including:

- Configuration storage, identity management, user status (such as presence and location), which is held by the Home Subscriber Server (HSS).
- Billing services, provided by a Charging Gateway Function (CGF).
- Control of voice and video calls and messaging, provided by the control plane.

Control plane: The control plane is placed between the application and user planes. It routes the call signaling, tells the user plane which traffic to allow, and generates billing information for the use of the network.

Several roles of SIP servers, called CSCF (Call Session Control Function), are used to process SIP signaling packets in the IMS.

The control plane also controls the user plane traffic through the Resource and Admission Control Subsystem (RACS). This consists on the Policy Decision Function (PDF), which implements local policy on resource usage, for example to prevent overload of particular access links, and Access-RAC Function (ARACF), which controls QoS within the access network.

User plane: The User plane provides an IPv6 QoS-enabled core network with access from the user. This infrastructure is designed to provide a wide range of IP multimedia server-based and P2P services.

Access to the core network is realized through Border Gateways (GGSN/PDG/BAS). They enforce the policy provided by the IMS core, controlling traffic flows between the access and the core networks.

Within the User Plane there can be deployed:

- The Interconnect Border Control Function (I-BCF) controls transport level security and tells the RACS what resources are required for a call.
- The I-BGF, A-BGF Border Gateway Functions provide media relay to hide endpoint addresses with managed pinholes to prevent bandwidth theft, and implement NAPT and NAT/Firewall traversal for media flows.

1.1.2. IMS provisioning / enablers

The IMS application plane enables creation of new, converged applications that include capabilities such as presence, mixed media (e.g. telephony and message exchange in the same session), and seamless working across fixed and mobile boundaries. These capabilities are supplied by the IMS provisioning services and the IMS Enablers.

The IMS provisioning: IMS provides a wide range of session border control, including call access control, reach ability and security. It also provides a framework for the deployment of both basic calling services and enhanced services, including:

- Multimedia messaging
- Video on demand
- Presence-based services
- Push-to-talk

The IMS enablers: [6] are feature-specific servers that provide Internet and communication services with productivity and community-enhancing features, such as instant messaging (IM), IP conferencing, presence/availability, location, group list management, gaming, and Web 2.0 type of value-added capabilities.

1.2. Ericsson Service Development Studio

Ericsson Service Development Studio (SDS) [7] is a fully comprehensive tool for development and end-to-end testing of the client and server side of new IMS applications.

The SDS server side includes an IMS core network emulator and can also be configured as a Java EE/SIP server execution environment for live user or for market IMS service trials. The SDS also includes advanced capabilities, the enablers, such as Presence and Group Management (PGM), Voice over IP (VoIP), Push-to-Talk (PTT), IMS-M and Combinational (CS Voice + PS sessions).

The SDS test part provides a Simulated Environment with diverse tools, where the nodes within the core IMS network are simulated. This allows the SDS to act as a virtual IMS network for a developer.

Finally for the client side, Ericsson defines the IMS client framework so that it will support:

- Easy and quick development
- Downloading of new IMS clients (to Open-OS devices and as Java ME clients)
- Co-existence of several clients (ICP multi-tasking)
- Seamless integration with device capabilities
- Seamless integration with IMS network CoSe enablers (for example, Presence and Groups, PTT, IMS Messaging)

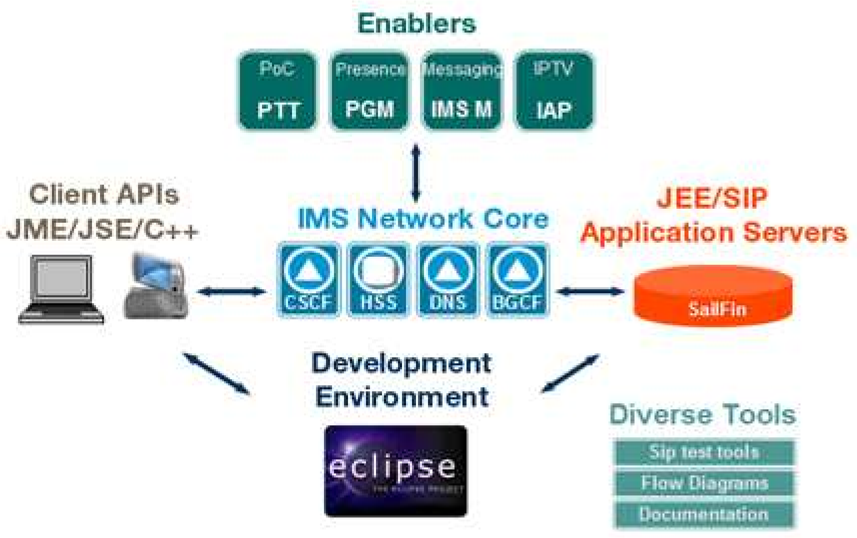


Figure 3 Service Development Studio

1.2.1. IMS Provisioning

The SDS includes IMS Provisioning for the IMS network core configuration and management. The IMS Provisioning is composed of a DNS provisioning, an HSS provisioning, and a Registrar.

DNS Provisioning: Resolves the general call flow sessions between users by matching domain addresses, globalized telephone numbers, IP addresses, and transport protocols. The DNS provisioning also enables the operator to solve the general call flow sessions between two nodes. It must determine the IP address, port, and transport protocol of the next hop element or the server domain that will receive the request.

HSS Provisioning: It allows creating Initial Filter Criteria and their coranswering Service Point Triggers to redirect the service requests. Also HSS Provisioning allows creating, modifying, and deleting user profiles, service profiles, and Public Service Identity profiles.

BGCF Provisioning: It allows configuring the BGCF using tables to determine where requests must be routed to: Calling Party Table, Called Party Table, and External Network Pool Tables.

Registrar: It displays the registration and expiration time of users in the simulated environment.

1.2.2. Presence enabler

The IMS presence enabler is a network service that adds value by enhancing person-to-person communication. It enables new services as well as enhances existing ones, leading to a richer communication experience. It totally adds a new dimension for client communication and can be used in many different applications.

1.2.2.1. Presence architecture

The presence is a dynamic profile of status information of people, applications, or machines that is visible to others. The presence feature enables a user (person or application) to store, manage, and publish presence information that can be viewed by authorized users.

The presence service is composed by the IMS server, the presence server and the XCAP Server as shown in Figure 4.

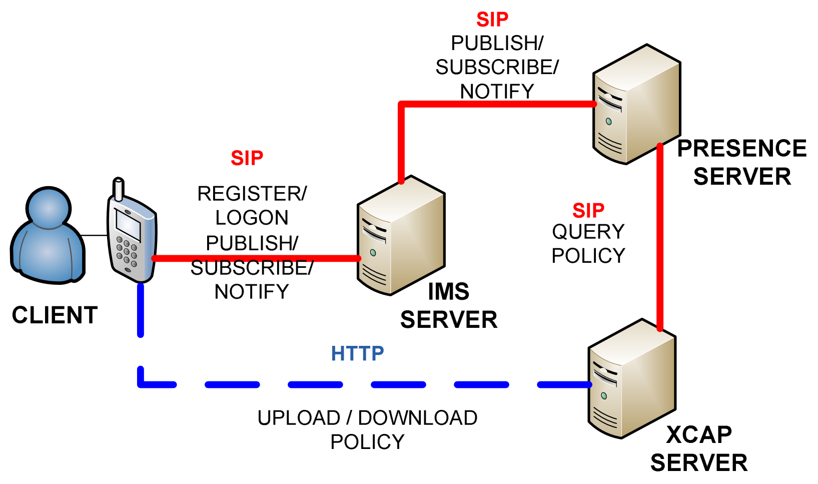


Figure 4 Presence service architecture

The IMS server is the IMS network emulator, concretely the CSCF. The IMS features that are used are the services of Authorization, Authentication and Accounting, and the initial filter criteria for management of SIP requests.

The presence server is the core of presence service. It is the responsible of the management, the subscription and notification of publish states. The information of presence service is stored in XML by the XCAP server.

The XCAP server is the responsible of the XML Configuration Access Protocol (XCAP) that allows the writing, modification, and deleting of data stored in XML format. The XML data elements are mapped to HTTP Uniform Resource Identifiers (URIs) to allow easy access to each of the elements, i.e. it is possible to store XML documents on the server by using HTTP requests.

1.2.2.2. CSCF

The CSCF [8] is responsible for all signaling via SIP between the Transport Plane, Control Plane, and the Application Plane of IMS. The CSCF consists of the Proxy CSCF (P-CSCF), Interrogating CSCF (I-CSCF), and the Serving CSCF (S-CSCF), which each have unique functions within IMS.

The P-CSCF is responsible for interfacing directly with the Transport Plane components and is the first point of signaling within IMS for any end-point. The P-CSCF receives the SIP messages of publishes, subscribes, notifies, and uses a DNS look-up to determine from which I-CSCF sending SIP messages, which could be an I-CSCF in its own network or another I-CSCF across an administrative domain.

The main function of the I-CSCF is to simply proxy between the P-CSCF as entry point and S-CSCF as control point for applications found in the Applications Plane. When the I-CSCF receives these SIP messages, it will perform a Home Subscriber Server (HSS) look-up via Diameter to determine the S-CSCF that is associated with the end-point terminal. Once it receives this information, it will forward the SIP message to the appropriate S-CSCF for further treatment.

The S-CSCF is responsible for interfacing with the Application Servers (AS) in the Application Plane. The S-CSCF is also responsible for routing all SIP messages to the AS allowing the Control Plane session control to interact with the Application Plane application logic. To do this the S-CSCF uses information obtained from the HSS in the form of Initial Filter Criteria (IFC) that acts as a trigger against inbound session establishment requests. When the S-CSCF receives the SIP messages, routes them to the Presence Server.

1.2.2.3. Presence Server

The Presence Server (PS) is an entity that accepts, stores and distributes presence information. The PS performs the following functions:

- Handles publications from one or multiple Presence Source(s) of a certain presentity. This includes refreshing presence information, replacing existing presence information with newly published information, or removing presence information, for a given Presence Source.
- Composes the presence information received from one or multiple Presence Source(s) into a single presence document
- Handles subscriptions from watchers to presence information and generates notifications about the presence information state changes.
- Handles subscriptions from watcher information subscribers to watcher information and generates notifications about the watcher information state changes.
- Authorizes the watcher’s subscription to the presentity’s presence information and applies policies .
- Applies the watcher’s event notification filtering preferences, as appropriate .
- Applies rate control mechanisms to the notifications, as appropriate.

1.2.2.4. XCAP Server & Documents

XCAP server is server able to handle XCAP requests, and also for storing “presence interesting” data. The presence data is stored in XML files and is form by three types of documents:

- Presence-rules: this file contains the presence rules control which users are blocked (i.e. these users will not receive a NOTIFY of the user's presence status), and which users are allowed.
- Group List Management: The Group List Management keeps track of buddy lists on the server.
- Resource List Server: This document can contain references to users buddy lists.

Every type of document is stored in a different folder, and every user has his three own files. The users can edit them using XCAP as the following section explains.

1.2.2.5. XCAP

XCAP is an HTTP-based protocol to access remote configuration data. Data is stored in XML format and XCAP protocol allows querying, modifying or deleting parts of such data. This protocol is described in the RFC 4825 [9].

The creation or modification of a document is performed using the HTTP PUT method.

The following code shows how a user called Bob can create a document:

```
PUT /services/resource-lists/users/bob/fr.xml HTTP/1.1

Content-Type: application/resource-lists+xml


<?xml version="1.0" encoding="UTF-8"?>

<resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists">

<list name="friends">

</list>

</resource-lists>
```

Figure 5 Example of document creation

To retrieve the document created in Figure 5, use the following request:

```
GET /services/resource-lists/users/bob/fr.xml HTTP/1.1
```

Figure 6 Example of document request

The response from the simulation is as follows:

```
HTTP/1.1 200 OK

Content-Type: application/resource-lists+xml


<?xml version="1.0" encoding="UTF-8"?>

<resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists">

        <list name="friends">

                <entry uri="sip:alice@i2cat.dyndns.org">

                        <display-name>Alice</display-name>

                </entry>

        </list>

</resource-lists>
```

Figure 7 Document Retrieve Result

An HTTP DELETE request can be used to delete a document.

Deleting the document created in Example can be done with the request:

```
DELETE /services/resource-lists/users/bob/fr.xml HTTP/1.1
```

Figure 8 Example Document Deletion

Finally, an HTTP PUT request can be used with a node selector. The body of the request contains the element to add.

To add a new <entry> element for Alice to the list named “friends” for the document created in Example use the following code:

```
PUT /services/resource-lists/users/bob/fr.xml/~~/

resource-lists/list[@name="friends"]/entry HTTP/1.1

Content-Type: application/xcap-el+xml


<entry uri="sip:alice@i2cat.dyndns.org">

        <display-name>Alice</display-name>

</entry>
```

Figure 9 Example of information addition

1.2.2.6. *Presence flow diagram*

In this section we show an example of the presence service showing the presence flow messaging.

The scenario of this example is formed by two IMS clients: Alice and Bob, the IMS CSCF server (called Server) and the SIP application server that is the SIP server which contains the presence service. In this example Bob subscribes himself to Alice status, and Alice updates the status.

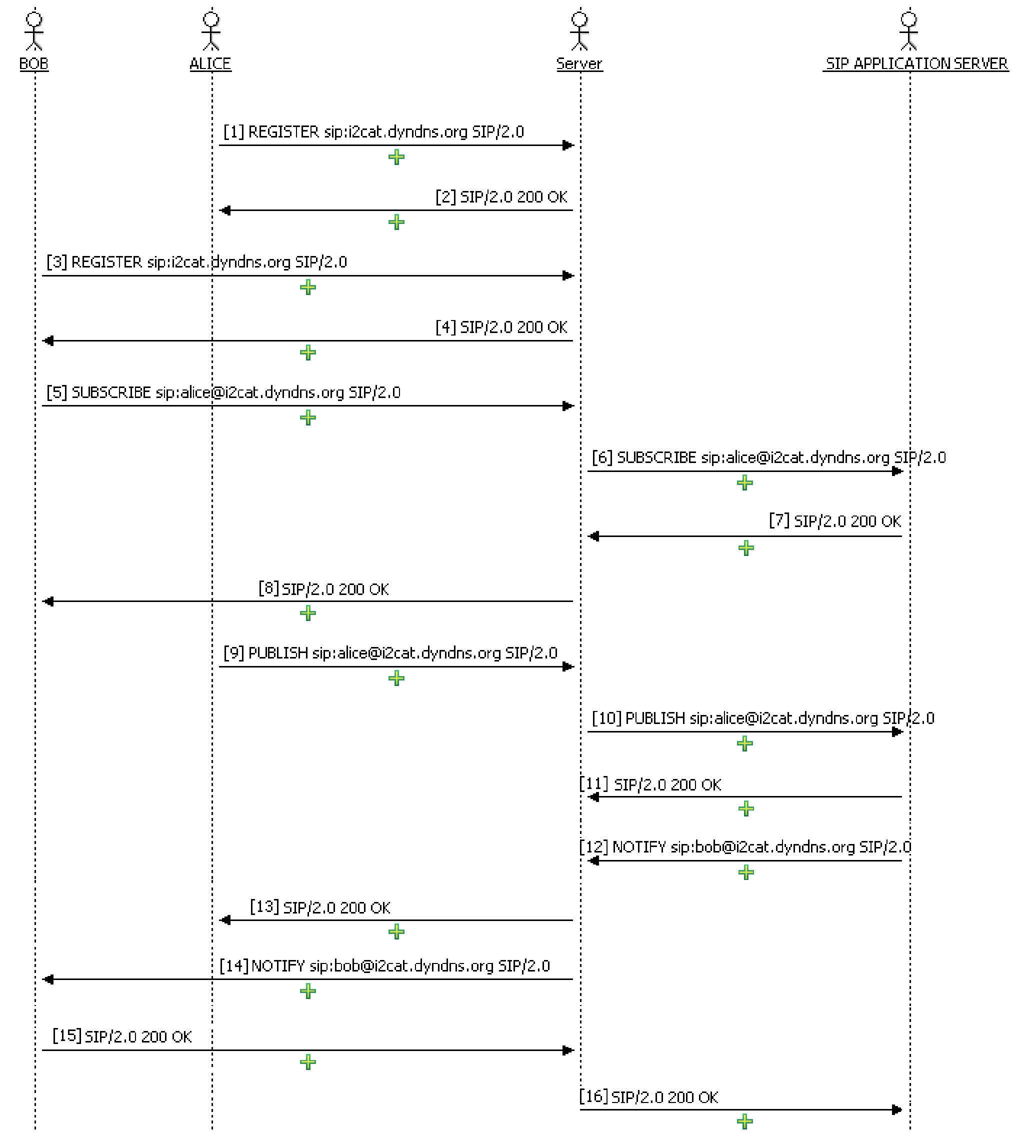

BOB
ALICE
Server
SIP APPLICATION SERVER
[1] REGISTER sip:i2cat.dyndns.org SIP/2.0
[2] SIP/2.0 200 OK
[3] REGISTER sip:i2cat.dyndns.org SIP/2.0
[4] SIP/2.0 200 OK
[5] SUBSCRIBE sip:alice@i2cat.dyndns.org SIP/2.0
[6] SUBSCRIBE sip:alice@i2cat.dyndns.org SIP/2.0
[7] SIP/2.0 200 OK
[8] SIP/2.0 200 OK
[9] PUBLISH sip:alice@i2cat.dyndns.org SIP/2.0
[10] PUBLISH sip:alice@i2cat.dyndns.org SIP/2.0
[11] SIP/2.0 200 OK
[12] NOTIFY sip:bob@i2cat.dyndns.org SIP/2.0
[13] SIP/2.0 200 OK
[14] NOTIFY sip:bob@i2cat.dyndns.org SIP/2.0
[15] SIP/2.0 200 OK
[16] SIP/2.0 200 OK

Figure 10 Presence flow diagram

First of all, both clients have to register their profiles into IMS (messages 1 to 4)

Once the users are registered, Bob announces to IMS the intention of subscribing to Alice status sending a SIP SUBSCRIBE Message (message 5).

The IMS redirects the message to the presence server emplaced on SIP Application Server (message 6).
The presence server requests the presence information of Alice and answers considering this information. In this case Bob is in Allow list of Alice presence information, and the presence server answers a “200 OK” message (messages 7 and 8).

After that, Alice updates her status sending a Publish message with the pertinent XML information (message 9 and 10). The next figure shows this XML.

```
PUBLISH sip:alice@i2cat.dyndns.org SIP/2.0

Max-Forwards: 70

Event: presence

CSeq: 11293 PUBLISH

Expires: 7200

Content-Length: 230

User-Agent: ICP30PGMAID

To: <sip:alice@i2cat.dyndns.org>

From: <sip:alice@i2cat.dyndns.org>;tag=119fad4-2c1c.1c38

Route: <sip:orig@127.0.0.1:5081;lr>

Call-ID: 2dd8-a41-214321ac@127.0.0.1

Content-Type: application/pidf+xml

Via: SIP/2.0/UDP 127.0.0.1:6050;branch=z9hG4bK42.49e1.1c38



<presence xmlns="urn:ietf:params:xml:ns:pidf" entity="sip:alice@i2cat.dyndns.org">

        <tuple id="t1308495763">

                <status>

                        <basic>open</basic>

                </status>

                <contact priority="High">sip:alice@i2cat.dyndns.org</contact>

        </tuple>

</presence>
```

Figure 11 Presence SIP Publish message

If the published message is correct the Presence Server answers a “200 OK” message (messages 11 and 13).

Finally the Presence Server sends a notification of Alice to his subscribers using a SIP NOTIFY message. In this case the Presence Server sends a NOTIFY message to Bob, and him answers a “200 OK” message when receives it.

1.2.2.7. PGM Simulator

The SDS Presence and Group List Management (PGM) is a simulation that allows your client managing the presence of individual users or groups.

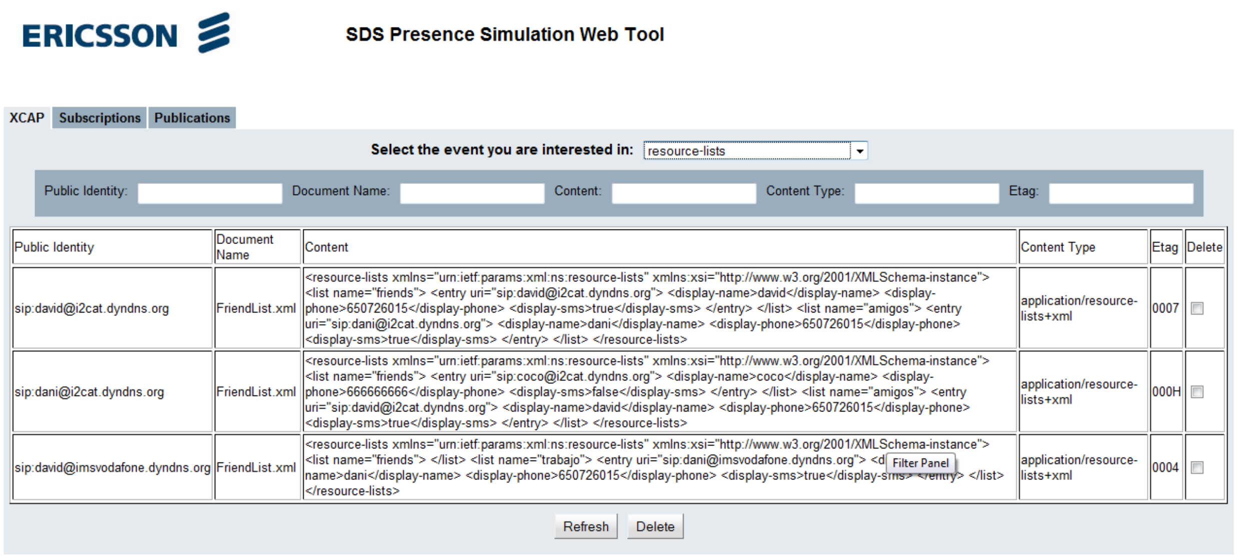
The SDS 4.1 presence simulation is a SIP application installed on the Sailfin SIP container. To use it, SDS comes preconfigured with a Presence iFC and eight SPTs.

The client application for PGM can be any kind of IMS client and the only requirement is that must support the standard presence documents.

Some features to point out are:

- When you start the SIP AS, the PGM simulation is automatically started as a service on the SIP Container.
- SDS is installed by default as a preconfigured Presence iFC and SPT.
- The PGM simulation have no kind of provisioning, so it will accept any subscriptions/publications sent to it.
- The PGM simulation does not authenticate requests.
- Authorization (block/allow) can be modified using the pres-rules.
- To remove a user publication, publish with an expiration of 0. The published content will be sent to all subscribed users with a NOTIFY message.

The PGM simulator has also a Web tool that enables the administrator to know the Presence XCAP files configuration, the user subscriptions and the user notifications. This tool is provided in Sailfin as a service Web application.



Select the event you are interested in: resource-lists

Public Identity: Document Name: Content: Content Type: Etag:

Public Identity	Document Name	Content	Content Type	Etag	Delete
sip:david@i2cat.dyndns.org	FriendList.xml	<resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <list name="friends"> <entry uri="sip:david@i2cat.dyndns.org"> <display-name>david</display-name> <display-phone>650726015</display-phone> <display-sms>true</display-sms> </entry> </list> <list name="amigos"> <entry uri="sip:dani@i2cat.dyndns.org"> <display-name>dani</display-name> <display-phone>650726015</display-phone> <display-sms>true</display-sms> </entry> </list> </resource-lists>	application/resource-lists+xml	0007	☐
sip:dani@i2cat.dyndns.org	FriendList.xml	<resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <list name="friends"> <entry uri="sip:coco@i2cat.dyndns.org"> <display-name>coco</display-name> <display-phone>666666666</display-phone> <display-sms>false</display-sms> </entry> </list> <list name="amigos"> <entry uri="sip:david@i2cat.dyndns.org"> <display-name>david</display-name> <display-phone>650726015</display-phone> <display-sms>true</display-sms> </entry> </list> </resource-lists>	application/resource-lists+xml	000H	☐
sip:david@imsvodafone.dyndns.org	FriendList.xml	<resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <list name="friends"> </list> <list name="trabajo"> <entry uri="sip:dani@imsvodafone.dyndns.org"> <d [illegible] name>dani</display-name> <display-phone>650726015</display-phone> <display-sms>true</display-sms> </entry> </list> </resource-lists>	application/resource-lists+xml	0004	☐

Figure 12 SDS PGM Web tool

1.3. Session Initiation Protocol

The Session Initiation Protocol (SIP) [10] is a signalling protocol, widely used for setting up and tearing down multimedia communication sessions such as voice and video calls over Internet Protocol (IP). Other feasible application examples include video conferencing, streaming multimedia distribution, instant messaging, presence information and online games. The protocol can be used for creating, modifying and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or several media streams. The modification can involve changing addresses or ports, inviting more participants, adding or deleting media streams, etc.

1.3.1. SIP Messages

SIP is a text-based protocol with syntax similar to that of HTTP. There are two different types of SIP messages: requests and responses. The first line of a request has a method, defining the nature of the request, and a Request-URI, indicating where the request should be sent. The first line of a response has a response code.

For SIP requests, RFC 3261 [11] defines the following methods:

- **REGISTER:** Used by a UA to notify its current IP address and the URLs for which it would like to receive calls.
- **INVITE:** Used to establish a media session between user agents.
- **ACK:** Confirms reliable message exchanges.
- **CANCEL:** Terminates a pending request.
- **BYE:** Terminates a session between two users in a conference.

- **OPTIONS:** Requests information about the capabilities of a caller, without setting up a call.

The SIP response types defined in RFC 3261 fall in one of the following categories:

- **Provisional (1xx):** Request received and being processed.
- **Success (2xx):** The action was successfully received, understood, and accepted.
- **Redirection (3xx):** Further action needs to be taken (typically by sender) to complete the request.
- **Client Error (4xx):** The request contains bad syntax or cannot be fulfilled at the server.
- **Server Error (5xx):** The server failed to fulfil an apparently valid request.
- **Global Failure (6xx):** The request cannot be fulfilled at any server.

1.3.1.1. SIP Message example

This is a SIP Register message example. We can see the most usual headers of SIP.

```
REGISTER sip:imsvodafone.dyndns.org SIP/2.0
Via: SIP/2.0/UDP
192.168.1.5:5060;rport;branch=z9hG4bKPjYpBu4f0hGJWwSit02FGTY-
7JllO3SERM
Route: <sip:imsvodafone.dyndns.org:5081;lr>
Max-Forwards: 70
From: <sip:dani@imsvodafone.dyndns.org>;tag=Ys1WScIJQZU-
td70KSAQa9M.56MAdZcY
To: <sip:dani@imsvodafone.dyndns.org>
Call-ID: uV4kkWfgPPPl6rlcWeWOaUmGEzdn8Amn
CSeq: 32750 REGISTER
User-Agent: Sipek on PJSUA v1.0.1/win32-wince
Contact: <sip:dani@192.168.1.5:5060>
Expires: 300
Content-Length:  0
```

Figure 13 SIP Message Header example

CHAPTER 2. THE ORCHESTRATION PLATFORM

The orchestration platform is a group of tools that provides an infrastructure to develop and publish new services to the IMS network. The main features of this platform are:

- Easy development of services by unprofessional developers.
- Easy publishing of new services by third parties.
- Record time to market of new services.
- Offer integration with the network.
- Offer services integration between vendors.

To obtain the best profit of the Orchestration Platform it is necessary to offer the same network interaction between service developers and the network as the IMS specifies. The allowing access of IMS resources through outside networks like Internet can also provide a new front of services that converges Internet and IMS networks.

2.1. Selected Technologies

An important part of needed characteristics come from the paper of the new services that will be integrated in the platform. To provide the integration requirements the service side is based on an actual architecture technology that is becoming important in the programming environment, the Service Oriented Architectures (SOA).

SOA [12] is considered as the philosophy of the encapsulating application logic in services with a uniformly defined interface and making them publicly available via discovery mechanisms. This allows the integration and interoperability among services built from different vendors, that can be built in different technologies and deployed in different networks.

On another hand, to provide easy-to-use service developing we have chosen the Orchestration Director Engine (ODE). ODE is a software tool that permits the direction of Web Services [13], as musicians directed by the orchestra director, and also executes business processes written in Business Process Execution Language (BPEL) that has the role of score. BPEL is a standard executable language for specifying interactions with Web Services. Processes in BPEL export and import information by using Web Service interfaces exclusively.

Finally the platform includes Web interfaces based on Web 2.0 technologies to make easy the access and use of the platform.

2.2. Project Architecture

This platform is composed by the Orchestration Director Engine (ODE), the Web Services, the Universal Description, Discovery and Integration (UDDI) directory, the processes definitions, and the User Interface.

- The ODE is the engine of the service orchestration, and is the core of the unprofessional development system.
- The Web Services are the network services. This services come from third parties, unprofessional developers or Operator provisioning.
- The UDDI folder is an Extensible Markup Language (XML) based registry for businesses worldwide to list themselves on the Internet and on the orchestration platform.
- The Processes definitions are the base of the new developed services. They contain the definition of a new service created by the composition of the Web Services.
- The User interface is a Web application that provides an easy way to create the processes definitions.

The next figure shows the architecture of the Orchestration Platform at high level.

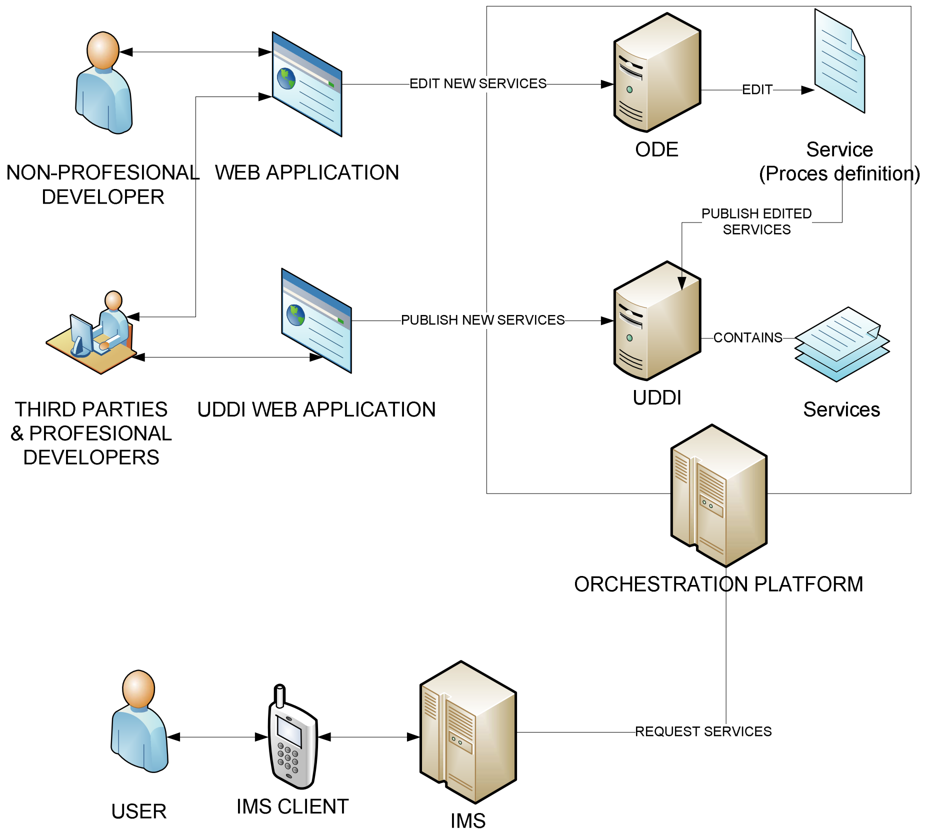


Figure 14 Project Architecture

2.3. Technologies of Architecture

In this section we explain the selected and required technologies to implement the orchestration platform.

2.3.1. Tomcat

Apache Tomcat is a servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP Web server environment for Java code to run.

Tomcat provides to the platform a base to use Web services and Web applications. The most important task of Tomcat is being the base for de ODE deployment.

2.3.2. ODE

As we have mentioned before, ODE is a business processes executor described by the WS-BPEL standard. The ODE software is a Web service and runs over Tomcat.

The ODE paper in the platform is being the executor of new services described by the developers using an easy Web interface which edits the WS-BPEL description. ODE is the attendant of call Web Services; it manages the interfaces for processors, instances and messages.

2.3.3. Processes definitions

The processes definitions are programs written in an orchestration language. The language employed in the project is Business Process Execution Language (BPEL).

BPEL is an Organization for the Advancement of Structured Information Standards *(*OASIS) standard that provides a language for the formal specification of business processes and business interaction protocols. It extends the Web Services interaction model and enables it to support business transactions. BPEL defines an interoperable integration model that should make easier the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

2.3.4. UDDI

UDDI is a public register designed as a structure storage of the information about companies and the offered provided services. A UDDI business registration consists of three components:

- White Pages — address, contact, and known identifiers;
- Yellow Pages — industrial categorizations based on standard taxonomies;
- Green Pages — technical information about services exposed by the business.

Through UDDI it is possible to publish and discover information about a company and his services. The most important characteristics of UDDI are that contains information about the technical interfaces of the companies services, and the possibly of using a group of API XML calls based on Simple Object Access Protocol (SOAP) to interact with UDDI in the design or execution time to discover the services technical data to be invoked and used.

2.3.5. GlassFish Sailfin

Sailfin is a SIP servlet container developed by Glassfish, an open source community comprising Users, Developers, Partners, and Evangelists creating an industry leading Java EE 5 compatible enterprise-quality Application Server.

Sailfin stores SIP applications and it is the first point of access to the orchestration platform once the SIP messages are processed by the IMS platform.

2.3.6. Web Services

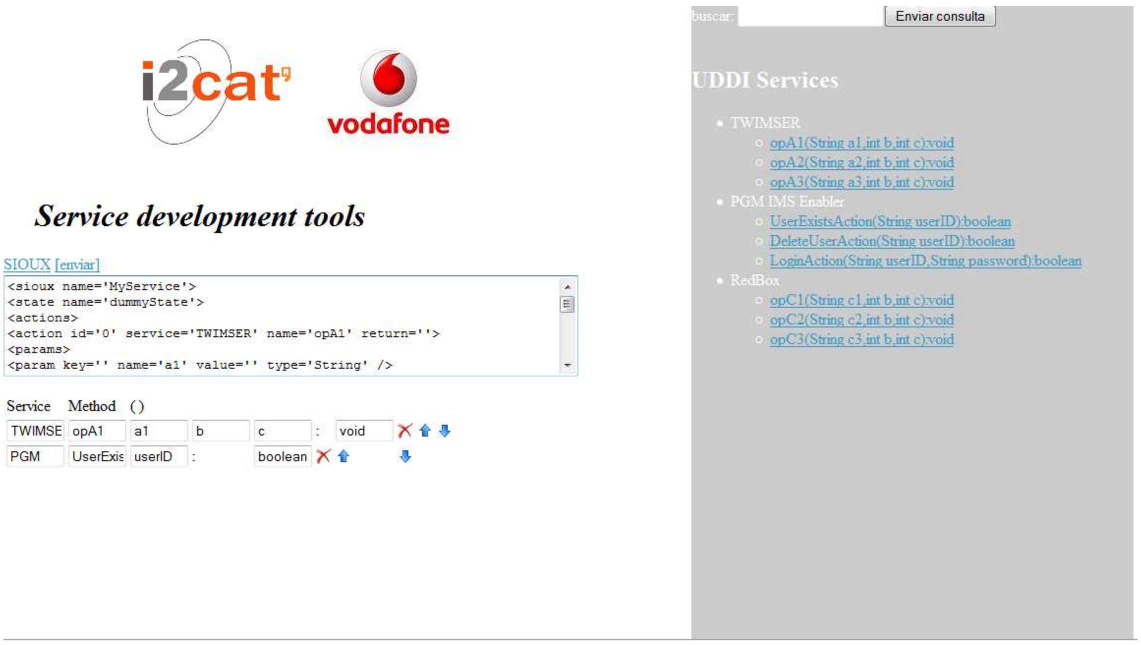
Web Service is software system designed to support interoperable machine-to-machine interaction over a network. Web services are frequently just Internet application programming interfaces that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

The Web Services technology encapsulates the application logic of services with a uniformly defined interface. This characteristic makes possibly the interaction between services provided by different owners, and a standard way to manage them by the orchestration platform.

2.3.7. Web Interface

The orchestration Web interface is a Web application programmed in JSP and JavaScript that permits the user to edit a process definition without writing XML code. This application also saves the process definitions into a file and makes it accessible for new process definitions.

The next figure shows the main page of the Web interface where you can see on the right frame the offered services on UDDI. In the left side you can see the selected function of UDDI services in the bottom and the XML generated in the top.



Service development tools

SIOUX [enviar]

```
<sioux name='MyService'>
<state name='dummyState'>
<actions>
<action id='0' service='TWIMSER' name='opA1' return=''>
<params>
<param key='' name='a1' value='' type='String' />
```

Service	Method	( )				
TWIMSE	opA1	a1	b	c	:	void
PGM	UserExis	userID	:	boolean		

UDDI Services

- TWIMSER
  - opA1(String a1,int b,int c):void
  - opA2(String a2,int b,int c):void
  - opA3(String a3,int b,int c):void
- PGM IMS Enabler
  - UserExistsAction(String userID):boolean
  - DeleteUserAction(String userID):boolean
  - LoginAction(String userID,String password):boolean
- RedBox
  - opC1(String c1,int b,int c):void
  - opC2(String c2,int b,int c):void
  - opC3(String c3,int b,int c):void

Figure 15 Orchestration Platform Web Interface

CHAPTER 3. TWIMSER

Twimser is a micro-blogging application, like Twitter, where users can send text messages from a mobile IMS terminal or from a Web page to his followers. These messages will arrive to the application, which will redistribute messages to the users interested in them.

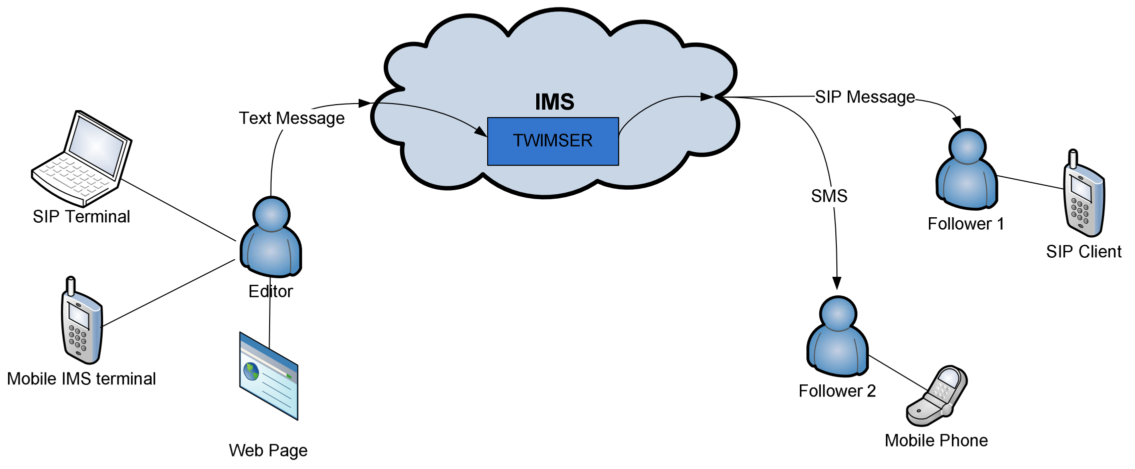


Figure 16 Twimser application

3.1. The architecture

The Twimser application is composed by a Web tool and several services: the Twimser service, the Presence Wrapper service, the Home Subscriber service, the Redbox service, the IMS-Messaging service and the SIP servlet.

The next figure describes the Twimser architecture at high level.

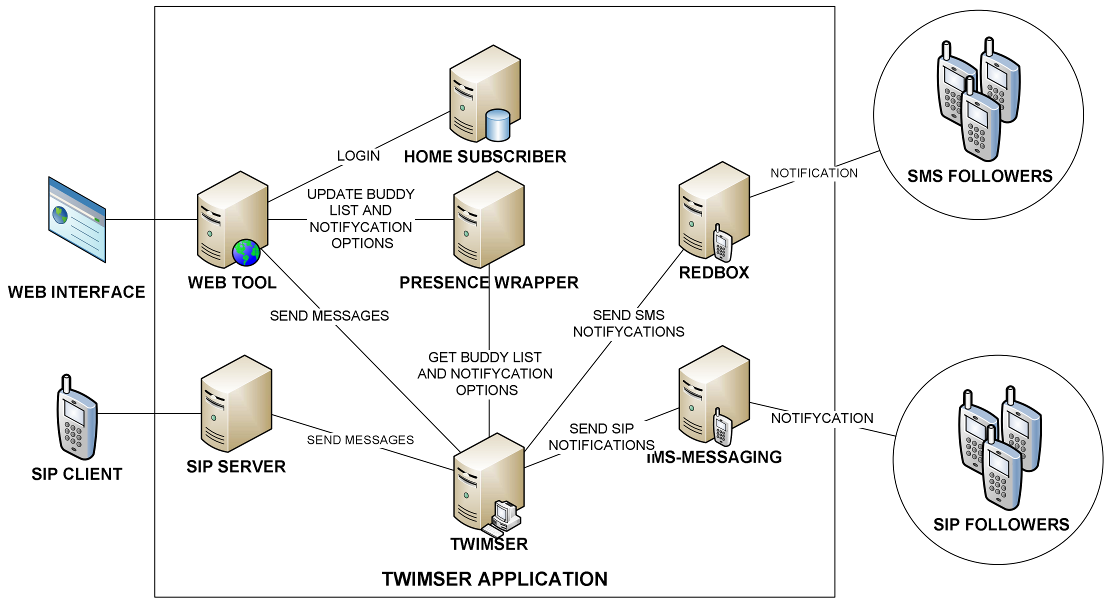


Figure 17 Twimser Architecture

3.1.1. The Presence Wrapper

The Presence Wrapper service is a presence service based on the IMS presence enabler that includes a buddy and group management. This service manages the buddy lists and the organization of buddies in groups. The Presence Wrapper service also stores information of the way of a follower has to be informed: though SMS or SIP Message. This service is described in detail in the next chapter.

3.1.2. Home Subscriber service

Home Subscriber service is a gateway to the IMS Home Subscribe Server (HSS) [14]. The HSS is the database of all subscriber and service data. Parameters include user identity, allocated S-CSCF name, roaming profile, authentication parameters and service information. The HSS also provides the traditional Home Location Register (HLR) and Authentication Centre (AUC) functions. This allows the user to access the packet and circuit domains of the network initially, via IMSI authentication.

The Home Subscriber service is a Web service that enables the interaction of the applications to the HSS database in order to provide an internetwork authentication service. This service is also public in the UDDI, as developer service in the orchestration platform.

To keep security in this service the Home Subscriber Web service is only able to get requests, it has not options to add or modify HSS database information.

3.1.3. RedBox service

The RedBox service is a Web service wrapper of a third party service that offers SMS sending service.

3.1.4. IMS-Messaging simulator

The SDS IMS Messaging (IMS-M) simulator simulates various messaging functionalities such as the popular Chat and Instant Messaging services currently available on the Internet. It also enables the testing of existing short message service messaging solutions by securing their interworking with IMS messaging application development.

The SDS IMS-M Simulator allows IMS subscribers to send and receive messages such as voice, text, video, pictures, and application data, to and from other users. It is an IMS enabler designed according to standards so that it may be used by other IMS applications for their messaging functions and testing.

3.1.5. The Web Tool

The Web Tool is a Web application that makes easier the interaction of the user to Presence Wrapper service. This tool enables the user to manage his buddy list and the way to inform them with a simple interface. The Web Tool also enables the possibly of sending messages directly from the Web. This Web tool is also part of the Presence Wrapper and is explained in the Presence Wrapper Chapter.

3.1.6. SIP Server

The SIP Server is the first point of the Twimser application for SIP clients. Once the client sends a message through the Twimser client application, the iFC in the P-CSCF of IMS redirects this request to the SIP Server.

The SIP server is a SIP container (Sailfin) with a SIP application developed in JSR289 Specification. The SIP application receives the SIP Twimser messages and executes the Twimser engine.

3.1.7. TWIMSER

The Twimser server is the engine of the Twimser application. This engine is a BPEL programming code, wrapper in a Web service and executed by ODE. The function of this engine is to connect to the Presence Wrapper to request the buddy list, and his own way information option (SMS or SIP). Once the Twimser application has this information, it reads the list, and delegates the sending information action to IMS-Messaging service or RedBox service.

3.2. Use cases

In this section there are exposed two possible use cases: In the first case we have a user that edits his buddy list; the second case a user updates his status blog generating a new message.

3.2.1. Edit buddy case

In this part is shown a diagram flow with a description of the case of a user adds a new friend into the buddy list.

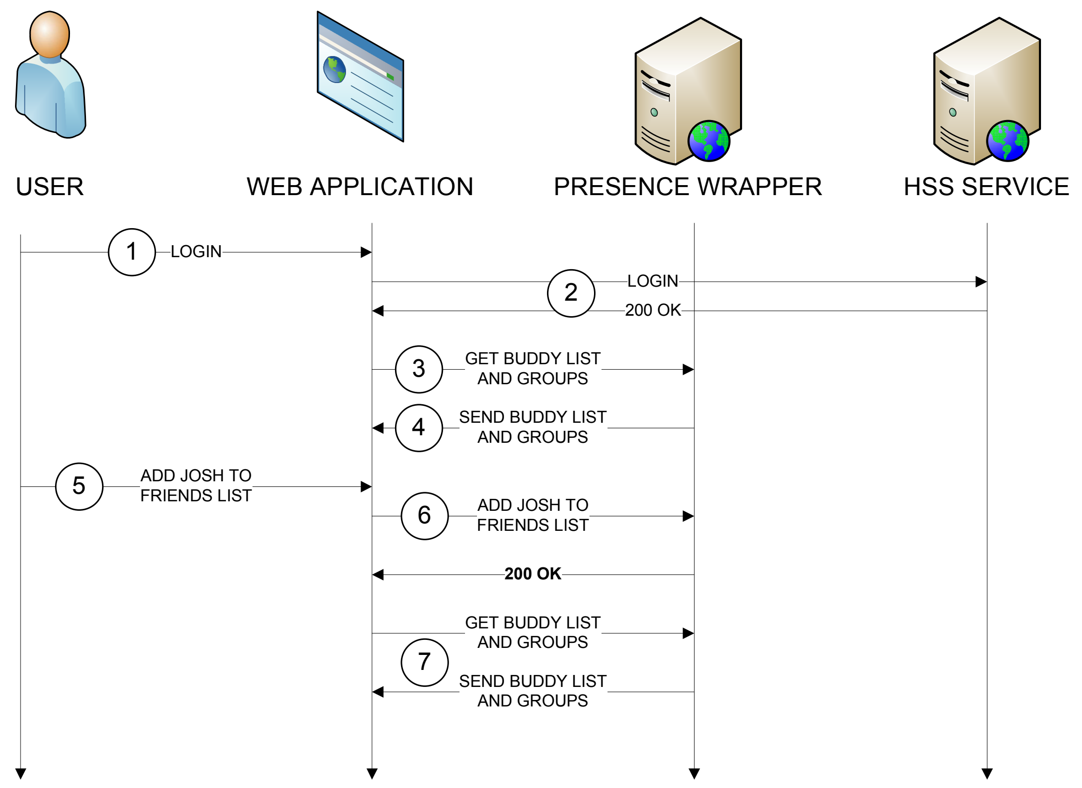

USER
WEB APPLICATION
PRESENCE WRAPPER
HSS SERVICE
1 LOGIN
2 LOGIN
200 OK
3 GET BUDDY LIST AND GROUPS
4 SEND BUDDY LIST AND GROUPS
5 ADD JOSH TO FRIENDS LIST
6 ADD JOSH TO FRIENDS LIST
200 OK
GET BUDDY LIST AND GROUPS
7
SEND BUDDY LIST AND GROUPS

Figure 18 Edit buddy case

1. The user opens the Web application and sends his IMS user and password formulary.
2. The Web application attacks to the HSS service with the user and password.
3. The HSS verify if the login is correct and answers to the Web application.
4. Once the Web application receives a correct authentication, attacks to the Presence Wrapper to know his buddy list and group management.
5. The user can see his buddy list, and now has the option to modify his the notification buddy ways, delete the user or add a new user. Then the user adds a new user and sends the formulary.
6. The Web application sends the information to the Presence Wrapper for storage.
7. When the Presence Wrapper executes the action, returns a “200 ok” message and the Web application is updated.

3.2.2. Sending update status case

In this part is shown a diagram flow of the case of a user updates his status, sending a new notification.

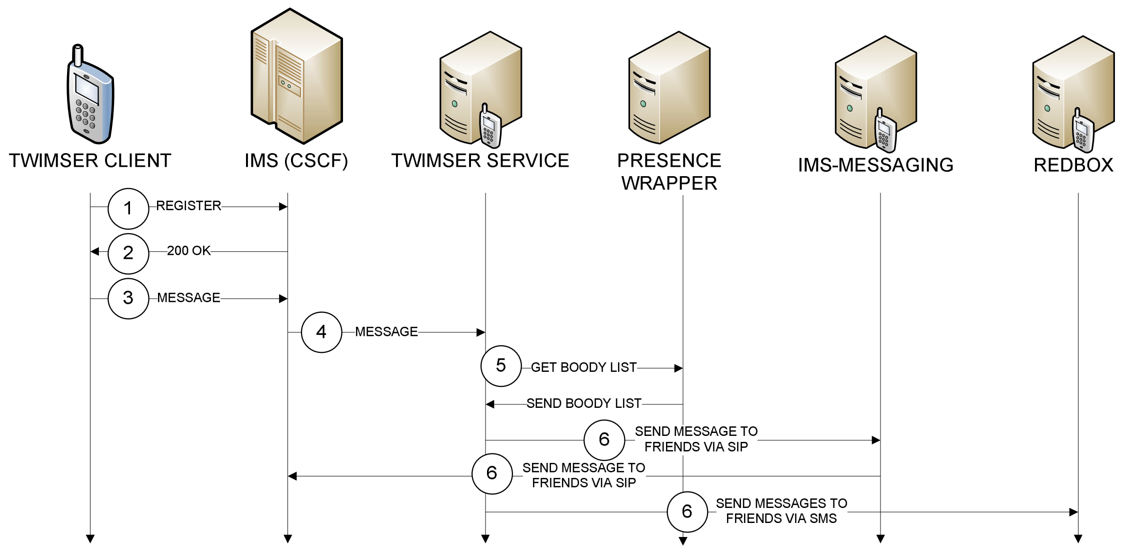


Figure 19 Sending update status case

1. The IMS client sends a SIP register message to the IMS.
2. The IMS authenticates the user and sends a “200 ok” message if the authentication is correct.
3. The IMS client publishes a new status through a message using the Twimser client application.
4. The message is received by the IMS platform and is redirected to the Twimser service.
5. The Twimser service gets the buddy list, buddies address and numbers, and the way to inform the buddies from the Presence Wrapper.
6. Once the Twimser service has the buddy information, makes sent information request to the IMS to use the IMS-Messaging service in case of inform using SIP, or makes the sent information request to Redbox service in case of informing using SMS.

CHAPTER 4. THE PRESENCE WRAPPER

The Presence Wrapper is a presence service based on Ericsson SDS PGM enabler developed for the orchestration platform with the objective of offering presence service to the new developed services. With this wrapper the third parties and nonprofessional developers do not have to implement their own presence service.

This wrapper is an important service to make near the apparition of new services based on Web 2.0 concept. The first objective of this wrapper is being the engine of a micro-blogging application; however it can be used as a social network application or RSS engines. Also the wrapper is presented as a Web service. This fact enables the use of the Presence Wrapper resource through other networks like Internet.

4.1. Architecture

As we have mentioned before, the Presence Wrapper is based on SDS PGM enabler, which is a presence enabler service. The PGM architecture is the same as the typical presence enabler which we have explained on the first chapter. The follow figure shows the PGM enabler architecture.

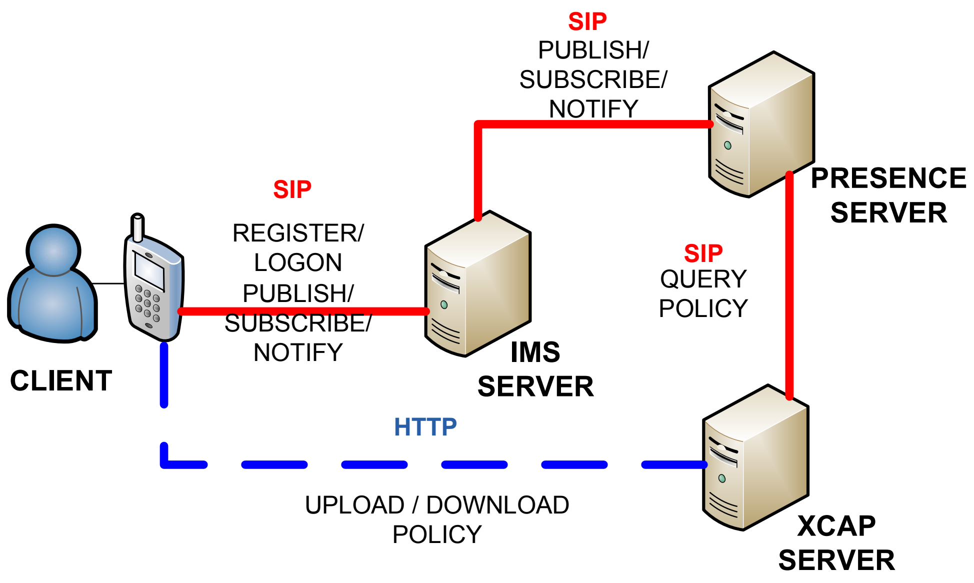


Figure 20 Presence service architecture

The Presence Wrapper respects the Presence enabler and amplifies his access ways through a Web service. This new Web service offers a standard and public interface to access to the presence service and also enables the service to other networks. To complement the wrapper and offer an easy way to manage the presence service, we have also included a Web application. The next figure shows the Presence Wrapper architecture with the new added complements.

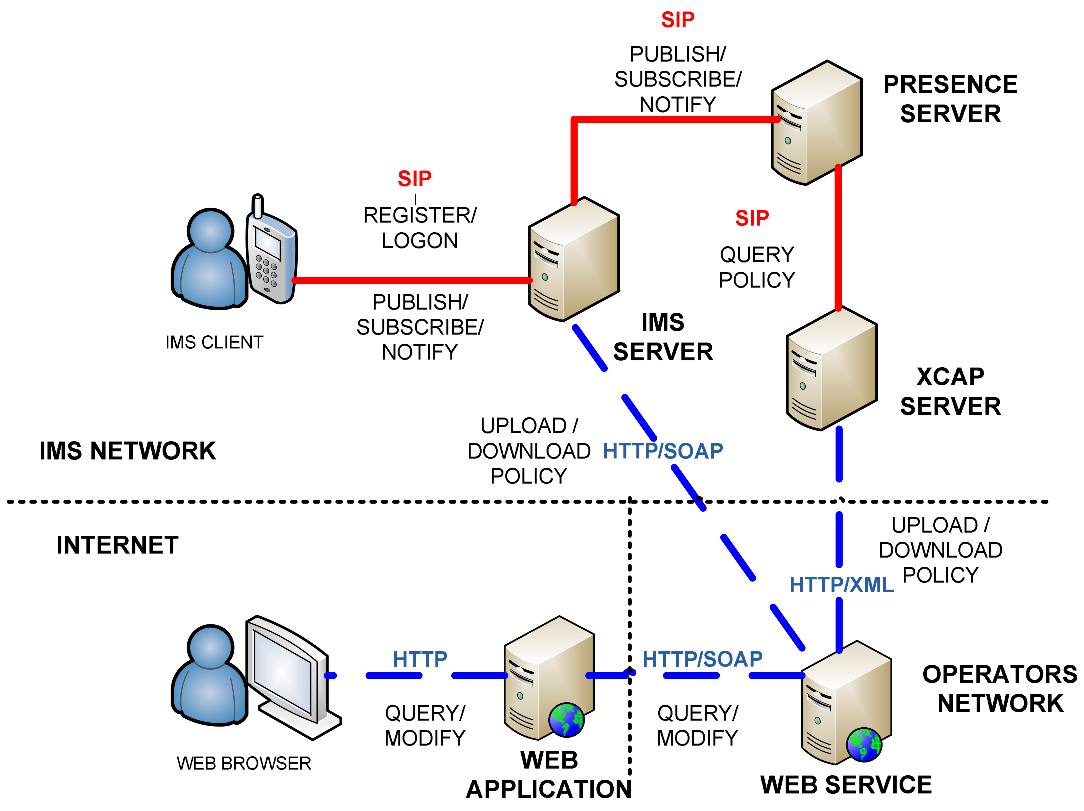


Figure 21 Presence Wrapper architecture

4.1.1. The presence Web service

The presence Web service is based on JAX-WS 2.0 technology. His function is to provide a new layer that works through the management of the XCAP data and communicating with the XCAP server. This new layer of the service offer the advantages of creating new functions more complex, and also performing the useful presence service to the developer.

The presence services functions where limited to put, delete or modify data information into the XML files. These functions were designed for a general use to store data, and this implies a complex and extended programming code to execute a function. Also the presence service implies to process the presence response to obtain the needed information. The most important features of the presence Web services are to solve these complications:

- It has specific functions for presence management.
- It is the responsible of the use the correct HTML method.
- It indicates the correct file path.
- It is able to program the code to write.
- It processes the XCAP responses and offers the response in basic data structure instead of XML format.

By using the Web service the user only has to invoke the type of function and provide the data information. The next list shows the implemented functions of the Web service:

- **start:** this method creates the required files into corresponding folders.
- **exist:** this method checks if a buddy exists on the buddy list
- **addFriend:** this method adds a buddy into buddy list in the default group.
- **addFriendInGroup:** this method adds a buddy into a specific group in the buddy list
- **setFriendName:** this method changes the name of the buddy contact
- **setFriendPhone:** this method changes the phone number of buddy contact
- **setFriendGroup:** this method moves the buddy to an specific group
- **setFriendSMSNotifycation:** this method distinguishes between the SMS way notification or SIP message notification
- **removeFriend:** this method removes a buddy of the buddy list.
- **getFriend:** this method returns the information of a buddy (name, phone and SIP address)
- **addGroup:** this method creates a new group into buddy list
- **getGroupList:** this method returns a list of all the groups that exists in your buddy list
- **removeGroup:** this method deletes a buddy group and his buddies.
- **getFriendUriList:** this method returns the SIP address of all buddies.
- **getFriendsPhone:** this method returns the phone numbers of all buddies.

Another advantage that presence Web services provide is that are able to edit the three types of XCAP files using an only function. For example when a new buddy is added using “addFriend” function, the presence Web service adds him to the resource-list file and also includes him into allowed list in the presence-rules file.

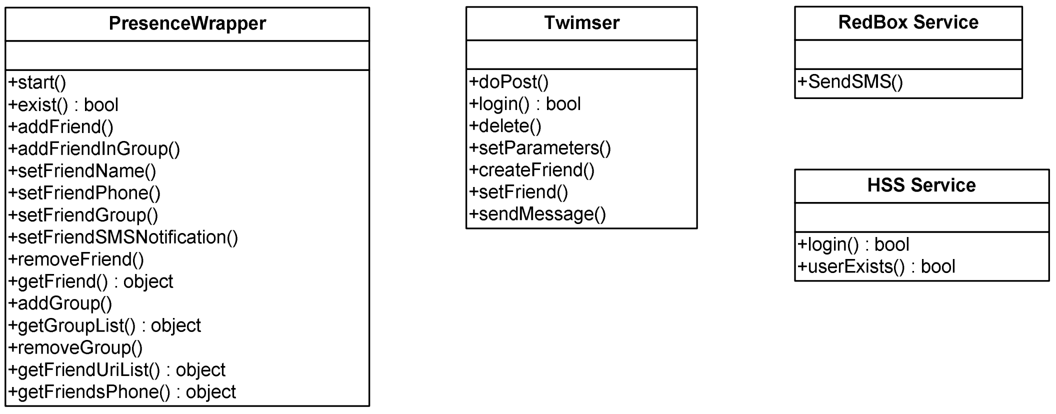


Figure 22 UML Diagram

4.1.2. The Web Application

The Web application is programmed in JSP language using Salifn as servlet container. This Web application is composed by three main Web pages. The first Web page is the login page which function is to authenticate the IMS user.

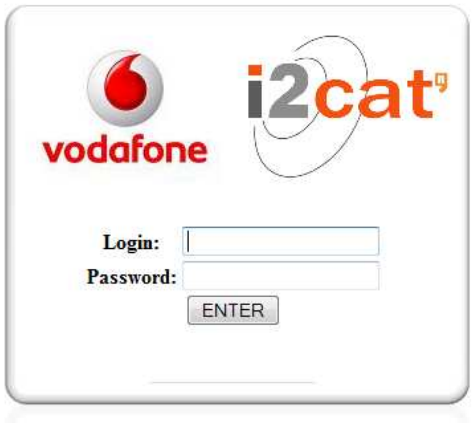
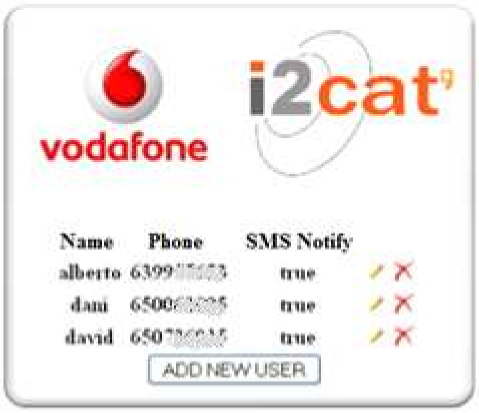


Figure 23 Login Web page

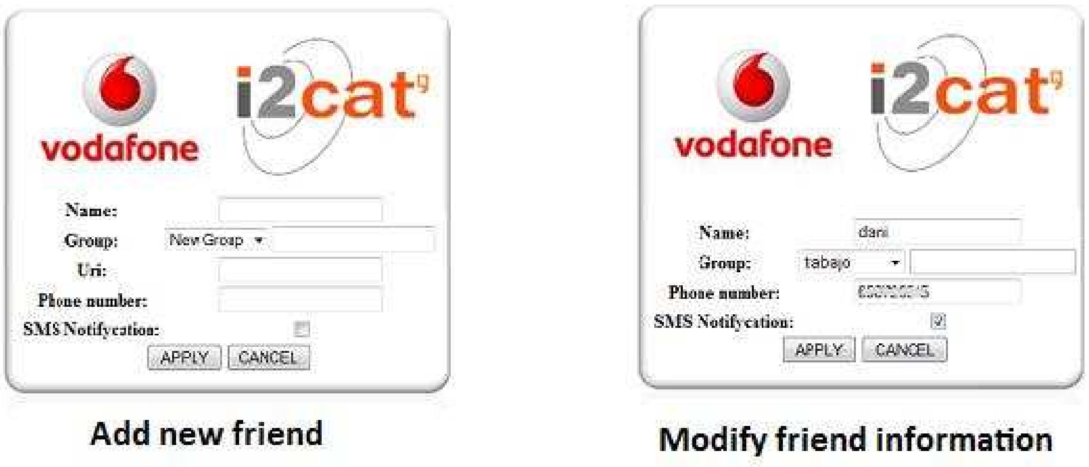
Once the login is done succesfully, we enter to the principal page, where we can see our buddy list and the notify way information. This Web page permits the user modifying the friend information, add a new friend or delete a friend.



Name	Phone	SMS Notify
alberto	6399[redacted]	true
dani	65006[redacted]	true
david	6507[redacted]	true

Figure 24 Principal Web page

The next figure shows the add new user page and friend information modification page.



Name:	
Group:	New Group
Uri:	
Phone number:	
SMS Notifycation:	☐

Name:	dani
Group:	tabajo
Phone number:	6507[redacted]
SMS Notifycation:	☑

Figure 25 Adding new friend page and Modifying friend information page

CHAPTER 5. THE TWIMSER CLIENT

To have a complete real demonstration of the platform we have also developed an IMS client for the Twimser application. This client is a C# language program compatible with a Microsoft Windows Operative System (Windows XP and Windows Vista) and also compatible with Microsoft Windows Mobile 6.0.

5.1. Technology

The client application is developed in Microsoft.NET Framework platform [15]. The core of the client is in the SIP stack, which is essential for the communication with the IMS.

5.1.1. Microsoft .NET Framework

The Microsoft .NET Framework is a software framework that can be installed on computers running Microsoft Windows operating systems. It includes a large library of coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering and is intended to be used by most new applications created for the Windows platform.

5.1.2. SIP Stack

The SIP stack chosen for the client is the PJSIP Wrapper [16], which is a wrapper of PJSIP open source stack [17].

5.1.2.1. PJSIP

PJSIP is a SIP stack programmed in C and supporting many SIP extensions/features, with the following key benefits:

- Extremely portable. Write the application once, and it would run on many many platforms (all Windows flavors, Windows Mobile, Linux, all Unix flavors, MacOS X, RTEMS, Symbian OS, etc.)
- Very small footprint. With less than 150KB for complete SIP features, PJSIP is ideal not only for embedded development where space is costly but also for general applications where smaller size means shorter download time for users.
- High performance. Which means less CPU power requirement and more SIP transactions/calls can be handled per second.

- Many features. Many SIP features/extensions such as multiple usages in dialog, event subscription framework, presence, instant messaging, call transfer, etc. have been implemented in the library.
- Extensive SIP documentation. There can never be enough documentation, so we try to provide fellow developers with hundreds of pages worth of documentation.

5.1.2.2. PJSIP Wrapper

PJSIP is a SIP stack written in C. It is a small footprint, high performance and portable library. And as such it is ideal for Softphone GUI developers. But there is a problem. GUI developers do not like C programming language. They prefer more sophisticated GUI designer tools and programming languages. In this article, a technique for C# (.Net) integration with pjsip (C) is described.

Integration with .Net framework (C#) requires a wrapper module accessible from C# as well from C. The module itself is written in C/C++ and it is recommended to be part of the pjsip project solution (Microsoft Visual Studio). The wrapper output is a dynamic library (.dll) which contains a common API needed by GUI.

5.2. The Application

In this part it is explained the .NET project SIP application. This part is important because the application is oriented to be reusable as standard IMS client. This client fulfils most of the basic SIP operations that IMS client requires and is structured in layers for a better integration with other applications.

5.2.1. Project structure

The Twimser Client Project is programmed in layers separating the network part, the business logic and the user interface. The aim of this structure is make the application compatible with other network layers based on different SIP stacks or future protocols, and the possibly of integrate the Twimser client into other projects or change the interface. The following figure describes the project structure.

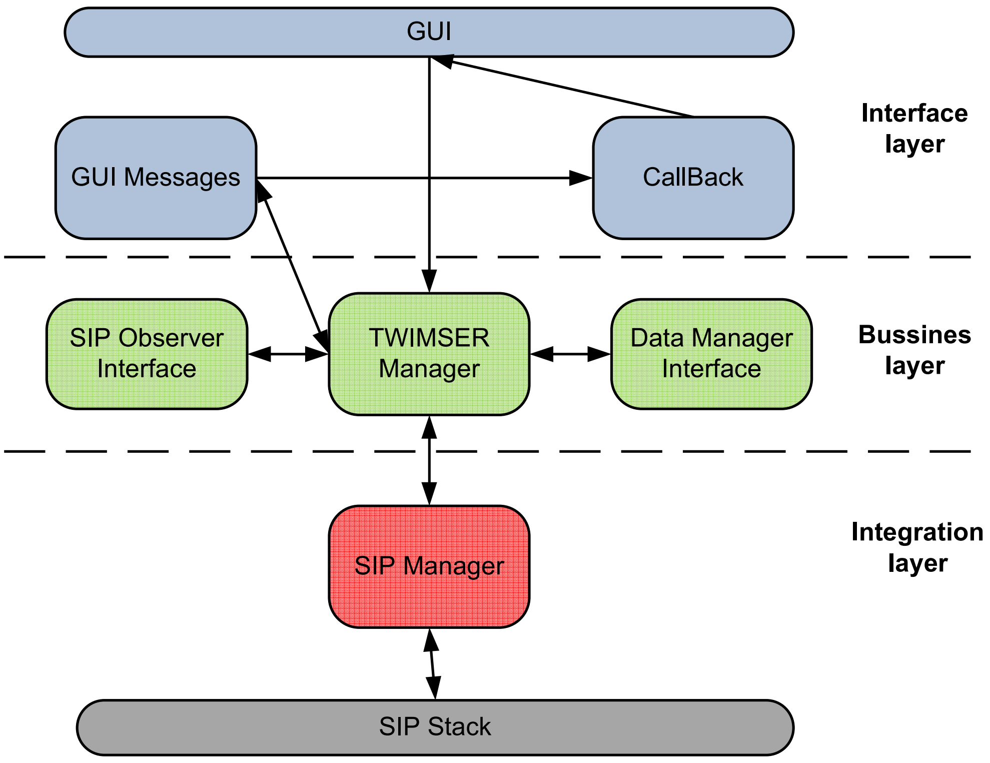


Figure 26 Twimser Client Application structure

The integration layer:

The integration layer is composed by the PJ SIP Manager. The PH SIP Manager is the responsible of the SIP network interaction. The main task of this part is the configuration of the PJ SIP library which is done using the PJSIP .NET Wrapper API [18].

The business logic layer:

The business logic is a level over the integration layer and his function is to provide an standard interface to adapt the Twimser client required functions over the network protocol stack (PJSIP in this case). The core of business logic layer is the TwimserManager which implements two defined interfaces: ISipDataManager and ISipObserver.

The ISIPDataManager interface implements three basic functions to program in the TwimserManager:

- **Register:** this function has to register the user into IMS
- **Unregister:** this function is to unregister the user into IMS.
- **Message:** this function has to send a SIP Message.

Also, The ISipObserver interface implements four functions more:

- **Log:** this function is an observer of client program information.
- **Register:** this function is an observer of register SIP responses.

- **Unregister:** this function is an observer of unregister SIP response.
- **MessageReceived:** this function is an observer of SIP Message received.

The interface layer:

The interface layer is the responsible of transmit the user interaction with the interface to the business logic layer, and also being the responsible of listen and tract the events done when an observer sends information.

5.2.2. Graphical User Interface

The Twimser client application is composed by two frames. The first frame is the configure frame where the IMS client information is required. The IMS required information is: the IMS Domain, the P-CSCF port, the user and password. When all the information is completed we can register our user. In the next figure the first frame is shown.

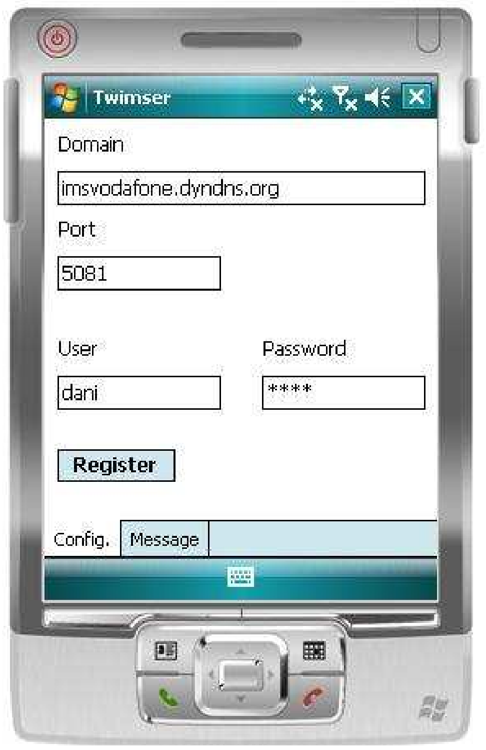
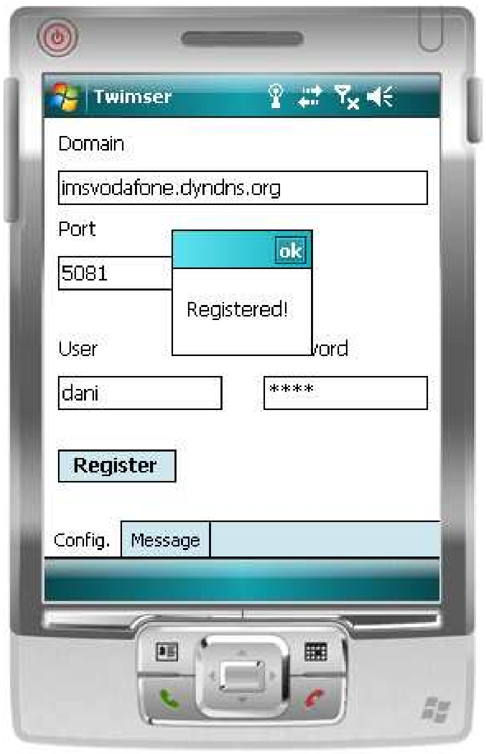

Twimser
Domain: imsvodafone.dyndns.org
Port: 5081
User
dani
Password: ****

Figure 27 Twimser Client Configuration frame

Once the register is done correctly an alert window appears confirming the success of the process.



imsvodafone.dyndns.org

5081

dani

****

Registered!

Figure 28 Register successful notification

The second frame is the interface to actualize the status message of the micro-blogging. This frame is formed by a Textbox where we can write our status and a Send button to update the status.

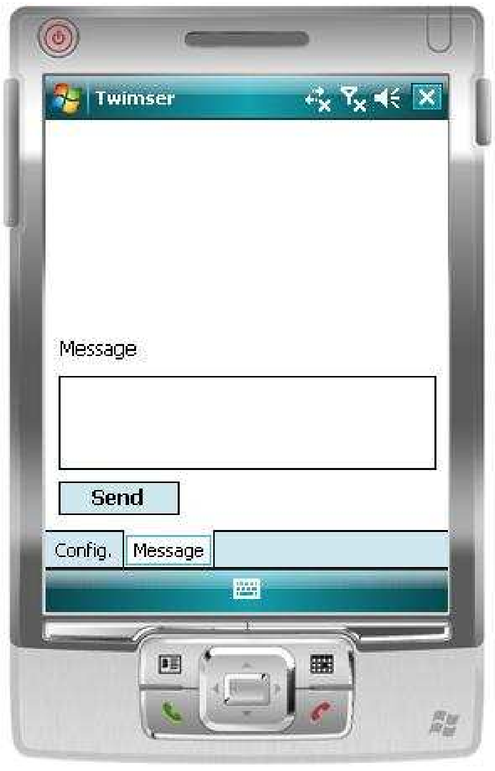
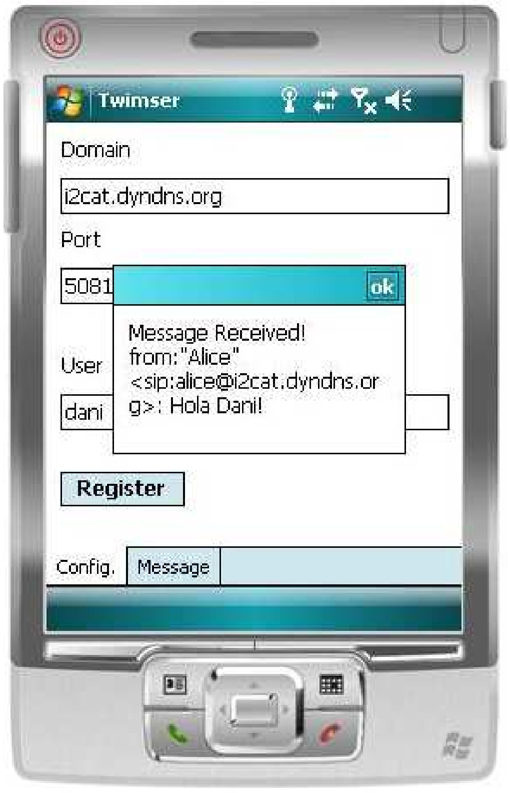


Figure 29 Update Status frame

At last, when a Message is received by the terminal a new popup appears informing of the sender and the information.



Domain

i2cat.dyndns.org

Port

5081

User

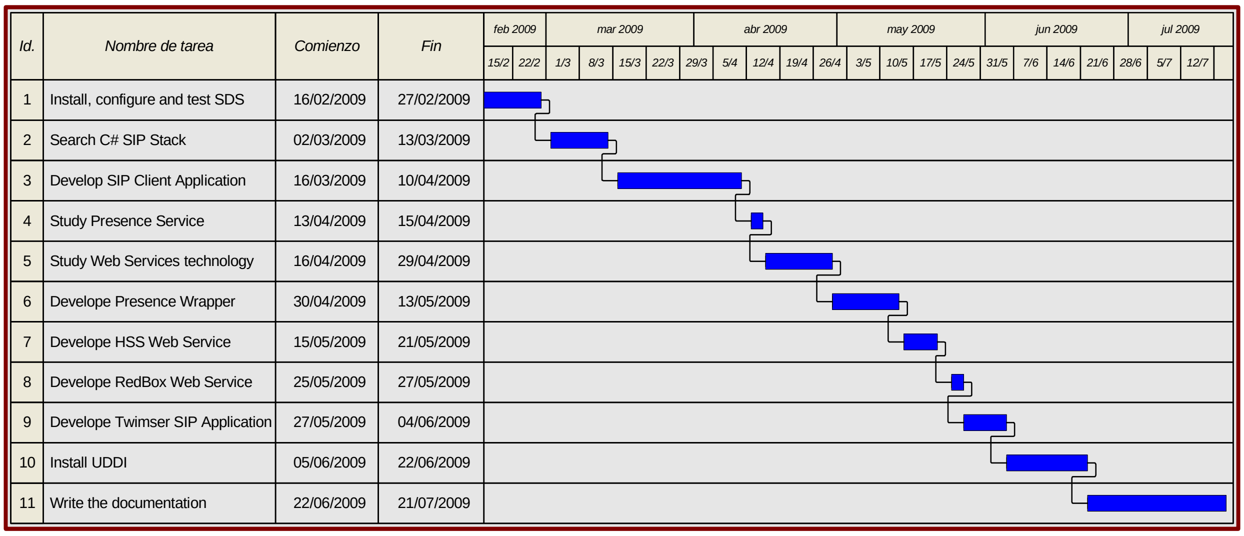
dani

Message Received!
from:"Alice"
<sip:alice@i2cat.dyndns.org>: Hola Dani!

Figure 30 SIP Message Notifycation

CHAPTER 6. WORK PLAN

In this chapter the different tasks that have been developed to realize Twimser project are explained. The next figure shows the Gantt diagram.



Id.	Nombre de tarea	Comienzo	Fin
1	Install, configure and test SDS	16/02/2009	27/02/2009
2	Search C# SIP Stack	02/03/2009	13/03/2009
3	Develop SIP Client Application	16/03/2009	10/04/2009
4	Study Presence Service	13/04/2009	15/04/2009
5	Study Web Services technology	16/04/2009	29/04/2009
6	Develope Presence Wrapper	30/04/2009	13/05/2009
7	Develope HSS Web Service	15/05/2009	21/05/2009
8	Develope RedBox Web Service	25/05/2009	27/05/2009
9	Develope Twimser SIP Application	27/05/2009	04/06/2009
10	Install UDDI	05/06/2009	22/06/2009
11	Write the documentation	22/06/2009	21/07/2009

Figure 31 Gantt diagram

The first task was the Installation, configuration and test of the SDS software. This task consisted on a familiarization with the environment (the IMS provisioning) and the realization of some tutorials that SDS offers.

Once we knew how SDS ran, the second task was searching a SIP Stack for C# language in order to start developing a SIP client in ".NET framework".

The first release of SIP Client Application was an IMS Client with the functionalities of Register, Unregister and Send Instant Messages to execute the SDS tutorial with a real mobile client.

The next task was the Presence Service part. The Presence Service started with the Presence enabler study and demonstration with SDS tools. Once we tested the Presence enabler the next objective was making it available through the orchestration platform. At this point we studied the Web Services technology for the network availability. And the finally part of the Presence Service was the Web Application to complete Presence wrapper developing.

The Presence Wrapper Web application required a user authentication service, and this was the next task of the project.

At this point the RedBox service was introduced to amplify the Twimser application functionality.

Finally, developing the SIP servlet application, the Twimser application was ended. After testing it, the next task was creating an UDDI server and registering all the developed services.

At last the project ended with the development of this master thesis report.

TASKS	HOURS
Installing and configuring SDS	80 h
Searching C# SIP Stack	80 h
Developing SIP Client Application	160 h
Studying Presence Service	20 h
Studying Web Services Technology	80 h
Developing Presence Wrapper	80 h
Developing HSS Web Service	40 h
Developing RedBox Web Service	20 h
Developing Twimser SIP Application	60 h
Installing UDDI	90 h
Writing the documentation	160 h
TOTAL	870 h

Figure 32 Time table

CHAPTER 7. ENVIRONMENTAL IMPACT & CONCLUSIONS

7.1. Environmental impact

The capability of adding new platforms and systems inside the IMS architecture enables the possibility of adapting future application architectures and application business models, and modernizing the network services without the requirement of important changes in the operator network. This fact implies a better usability of existing network resources.

On other hand this project has disadvantages in terms of environmental impact in terms of spent informatics resources, phone devices and all related with the project documentation.

7.2. Conclusion

The success of the IMS platform is directly related to the usability and content applications offered. With the aim to increase them it is necessary to make them easier to the service developers. The orchestration platform is a very powerful tool to provide an easier development, and also a record time to market. Another consideration is that an orchestration engine has no sense if it has not relation with the network provided services; the network provisioning is a very important tool.

In this context, the Presence Wrapper is an important service for the orchestration engine and it can supply the presence service developed for new services. Also, the Presence Wrapper service tries to make the IMS platform successful, increasing the use of it and approaching to the developers the new Web 2.0 concept applications.

7.3. Personal Conclusion

The main objectives of this project have been fulfilled getting as a result a high power potential tool. The orchestration platform has being a very interesting work for my experience due to the architecture complexity, the new market technologies used, and the last implanted new generation network base.

On another hand, the presence wrapper has been an important service for the platform, showing the potential of this platform and possibilities. The presence wrapper has become an essential part of the new Web 2.0 concept applications and also a known essential part of the IMS resources. This second fact is demonstrable showing the lasts Ericsson SDS department news which is promoting a new SDS add-on of presence and messaging service called TWITTY.

Finally, I would like to remark the satisfaction of all the project collaborating people, and the final consumer and the telecommunications operator. And I would also like to show my interest and enthusiasm on keeping working in this project to improve it.

ACRONYMS

AJAX – Asynchronous JavaScript and XML

ACK – Acknowledgement

AUC – Authentication Centre

BPEL – Business Process Execution Language

BGCF – Breakout Gateway Control Function

CSCF – Call Session Control Function

CGF – Charging Gateway Function

CSS – Cascading Style Sheets

DNS – Domain Name Server

GPP – Generation Partnership Project

GPRS – General Packet Radio Service

HTML - HyperText Markup Language

HTTP - HyperText Transfer Protocol

HRL – Home Location Register

HSS – Home Subscriber Server

ICF – Initial Filter Criteria

I-CSCF – Interrogating Call Session Control Function

JAX-WS – Java API for XML-Based Web Services

JSR – Java Specification Request

IMS – IP Multimedia Subsystem

LAN – Local Area Network

NGN – Next Generation Network

OASIS – Organization for the Advancement of Structured Information Standards

ODE – Orchestration Director Engine

P2P – Peer To Peer

P-CSCF – Proxy Call Session Control Function

PS – Presence Service

PGM – Presence and Group Management

PTT – Push-To-Talk

RFC – Request for Comments

RSS - Rich Site Summary

S-CSCF – Serving Call Session Control Function

SDS – Service Development Studio

SIP – Session Initiation Protocol

SIP AS – Session Initiation Protocol Application Server

SMS – Short Message Service

SOA – Service Oriented Architecture

SOAP – Simple Object Access Protocol

SPT – Service Point Trigger

TISPAN – Telecoms & Internet converged Services & Protocols for Advanced Networks

UDDI – Universal Description, Discovery and Integration

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

XCAP – XML Configuration Access Protocol

XML – Extensible Markup Language

REFERENCES

[1] Parashant Sharma. *Core Characteristics of Web 2.0 Services* [online] [Consulted: 2 July 2009]. Available at: <http://www.techpluto.com/web-20-services/>.

[2] *The application/rss+xml Media Type*. Network Working Group [online]. May 22, 2006. [Consulted: 3 July 2009]. Available at: <http://www.rssboard.org/rss-mime-type-application.txt>

[3] Rebeca Blood. *Weblogs: a history and perspective* [online] September 7, 2000 [Consulted: 2 July 2009]. Available at: <http://www.rebeccablood.net/essays/weblog_history.html>

[4] Danah M. Boyd and Nicole B. Elison. *Social Network Sites: Definition, History, and Scholarship* Journal of Computer-Mediated Communication, *13*(1), article 11 [online] 2007. [Consulted: 3 July 2009]. Available at: <http://jcmc.indiana.edu/vol13/issue1/boyd.ellison.html>

[5] *IMS Overview* [online] [Consulted: 1 July 2009]. Available at: <http://www.dataconnection.com/sbc/imsarch.htm>

[6] Mike Mc Hugh. The *Need for IMS Enabler Innovation* TMC net. [online] October 2007. [Consulted: 1 July 2009]. Availabe at: <http://www.tmcnet.com/ims/1007/industry-perspective-1007.htm>

[7] Ericsson.com [online] October 10, 2008. [consulted 10 July 2009]. Available at <http://www.ericsson.com/developer/sub/open/technologies/ims_poc/docs.html>

[8] *Call Session Control Function* Mobilein.com [online] . [Consulted: 4 July]. Available at: <http://www.mobilein.com/CSCF.htm>

[9] *The extensible Markup Languaje.* Network Working Group RFC 4825 [online] May 2007. [Consulted: 3 July 2009]. Available at: < http://tools.ietf.org/html/rfc4825>

[10] Mark A. Miller Understanding SIP [online] May 31, 2005 [Consulted: 15 July 2009]. Available at: <http://www.voipplanet.com/backgrounders/article.php/3508601>

[11] *SIP: Session Initiation Protocol.* Network Working Group RFC 3261 [online] June 2002. [Consulted: 3 July 2009]. Available at: < http://tools.ietf.org/html/rfc3261>

[12] Newcomer, Eric. *Understanding SOA with Web Services*. Lomow, Greg (2005). Addison Wesley. ISBN 0-321-18086-0*.*

[13] *JAX-WS* [online] [Consulted: 12 July 2009] Guide Available at: <http://ws.apache.org/axis2/1_5/jaxws-guide.html>

[14] *Session Control in the IP Multimedia Subsystem - White Paper* Page 2. [online] [Consulted: 2 July 2009] Available at: <http://www.newport-networks.com/whitepapers/IMS-2.html>

[15] *.NET Framework Guide* [online] [Consulted: 15 July 2009]. Available at: <http://msdn.microsoft.com/es-es/library/cc160717.aspx>

[16] Sasa *Coh. PJSIP.Net Wrapper* 2008 [online] [Consulted: 12 July 2009]. Available at: <http://sipekphone.googlepages.com/pjsipwrapper>

[17] Benny Prijono PJSIP.ORG [online] [Consulted: 12 July 2009]. Available at: <http://www.pjsip.org>

[18] *PJSIP .NET Wrapper API* [online]. [Consulted 14 July 2009]. Available at: <http://sipekphone.googlepages.com/pjsip.netwrapperapi>

ANNEXES

1. IMS Ericsson

The main component used to implement the IMS of Ericsson was the Ericsson Service Development Studio 4.1. As follows the installation instructions are provided.

SDS 4.1 installs and runs on a standard PC platform. The procedure described here describes that you will need to:

1. Read and accept the license agreement.
2. Enter a valid license key.
3. Choose which components you wish to install.
4. Select an installation directory.

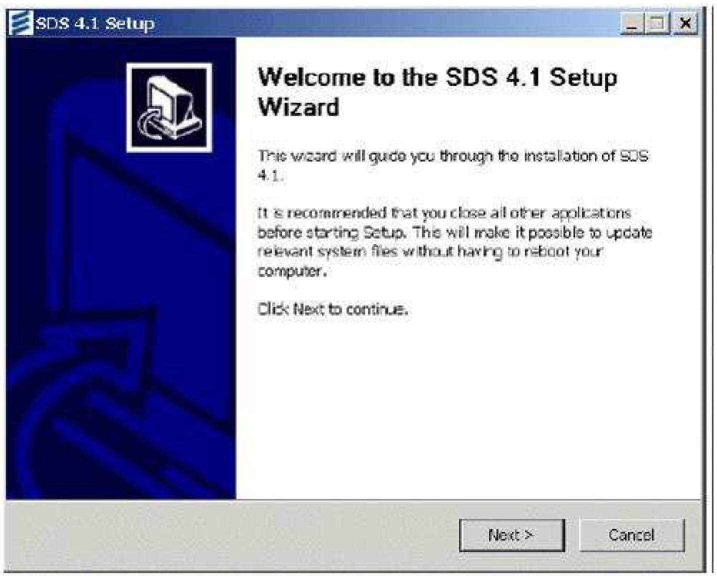
The subsequent part of the installation has to do with the components you selected for installation.

1.1 Before You Start

Installation of SDS 4.1 and all components is performed by a series of wizards.

- Installation of SDS requires a reboot.
- Close all applications on your workstation before installing SDS.
- Close all Windows Explorer windows before installing SDS.
- Ensure the Computer Management (or Services) window is closed.
- The SDS installation wizard detects if SDS has previously been installed, and prompts you to uninstall it before the latest version can install.
- If SDS was installed previously, it is not necessary to uninstall UIQ 3 SDK, the phone emulators, or any component not specifically mentioned.

1.2 SDS 4.1 Installation



SDS 4.1 Setup

Welcome to the SDS 4.1 Setup Wizard

This wizard will guide you through the installation of SDS 4.1.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

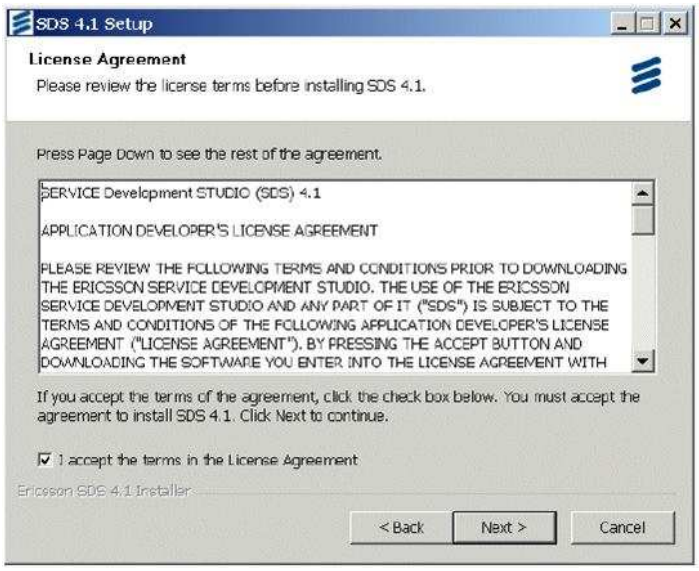
Click Next to continue.

Figure 1 SDS 4.1 Setup Wizard

To install SDS 4.1:

1. In the SDS 4.1 Welcome window, click **Next** to continue.

The **License Agreement** screen appears, prompting you to accept the terms of the license.



SDS 4.1 Setup

License Agreement

Please review the license terms before installing SDS 4.1.

Press Page Down to see the rest of the agreement.

SERVICE Development STUDIO (SDS) 4.1

APPLICATION DEVELOPER'S LICENSE AGREEMENT

PLEASE REVIEW THE FOLLOWING TERMS AND CONDITIONS PRIOR TO DOWNLOADING THE ERICSSON SERVICE DEVELOPMENT STUDIO. THE USE OF THE ERICSSON SERVICE DEVELOPMENT STUDIO AND ANY PART OF IT ("SDS") IS SUBJECT TO THE TERMS AND CONDITIONS OF THE FOLLOWING APPLICATION DEVELOPER'S LICENSE AGREEMENT ("LICENSE AGREEMENT"). BY PRESSING THE ACCEPT BUTTON AND DOWNLOADING THE SOFTWARE YOU ENTER INTO THE LICENSE AGREEMENT WITH

If you accept the terms of the agreement, click the check box below. You must accept the agreement to install SDS 4.1. Click Next to continue.

☑ I accept the terms in the License Agreement

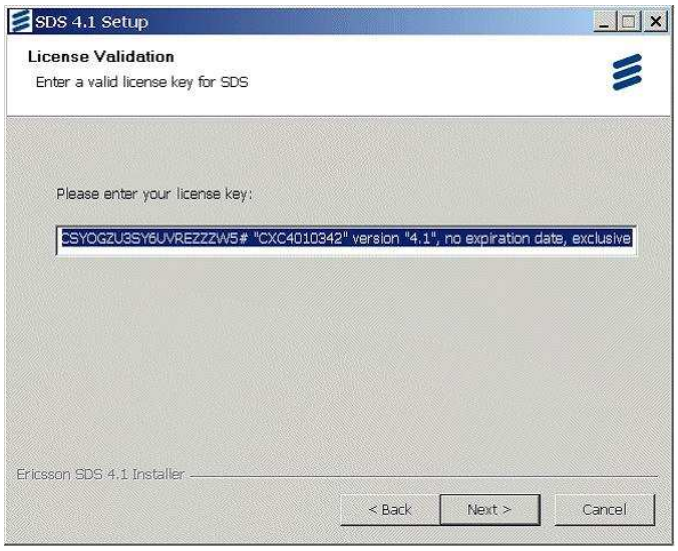
Ericsson SDS 4.1 Installer

Figure 2 License Agreement Window

Note: The **Next** button remains unavailable until you accept the license agreement.

2. Select the **I accept the terms in the License Agreement** check box and click **Next** to continue.

The **License Validation** window appears, prompting you for your license key.



SDS 4.1 Setup

License Validation

Enter a valid license key for SDS

Please enter your license key:

CSYOGZU3SY6UVREZZZW5# "CXC4010342" version "4.1", no expiration date, exclusive

Ericsson SDS 4.1 Installer

< Back | Next > | Cancel

Figure 3 License Validation Window

If you had an earlier version of SDS, the license key will be filled in automatically.

If you downloaded SDS from the EMW Website, a license key will be e-mailed to you at the address you specified when you registered on the Website. If you did not receive one, contact SDS support at **support@ericsson.com**.

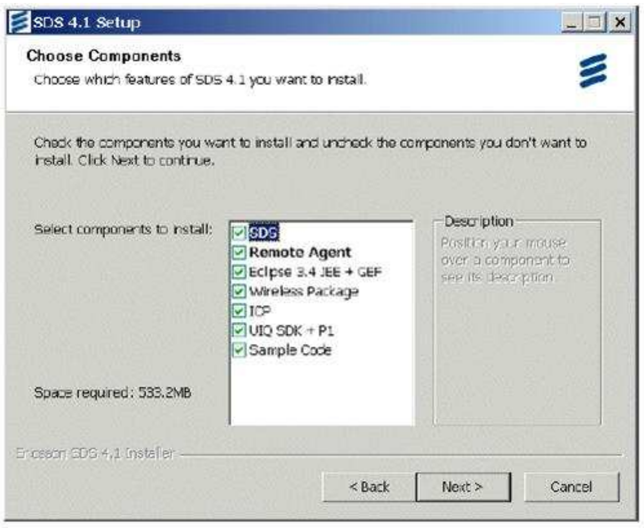
```
*ABRG4WPGB0IENAC9CBZKLAV6XPWYCYXVJ8TRHKNO6LXGTDP40V4VMSNQ80JUVSK28#
"CXC4010342" version "4.1", no expiration date, exclusive
```

Example 1 License Key (Invalid)

Note: The part of the license key starting with # is a comment on the license type of the key and is optional.

3. Click **Next**.

The **Choose Components** window is displayed, prompting you to select the features to install.



SDS 4.1 Setup

Choose Components

Choose which features of SDS 4.1 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] SDS
- [✓] Remote Agent
- [✓] Eclipse 3.4 JEE + GEF
- [✓] Wireless Package
- [✓] ICP
- [✓] UIQ SDK + P1
- [✓] Sample Code

Description

Position your mouse over a component to see its description.

Space required: 533.2MB

Ericsson SDS 4.1 Installer

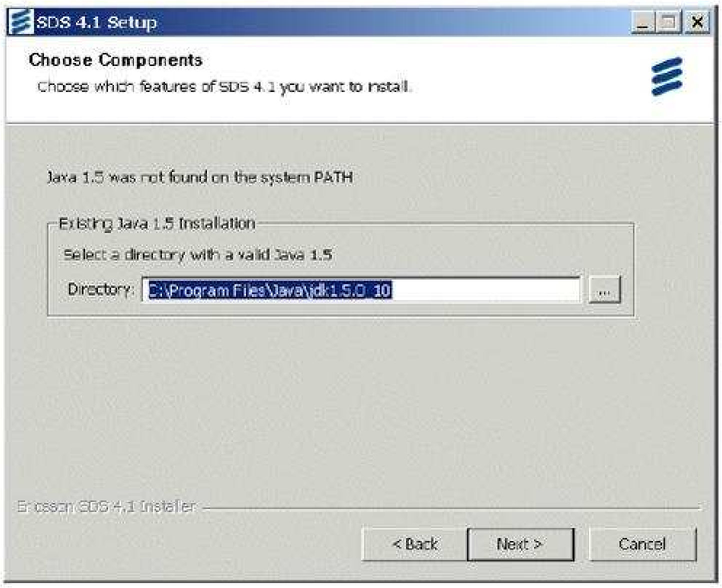
< Back | Next > | Cancel

Figure 4 Choose Components Window

Component	Description
SDS	Installs SDS 4.1 core components. This item is mandatory.
Remote Agent	Installs the SDS Remote Agent. This item is mandatory.
Eclipse 3.4 JEE + GEF	Installs Eclipse 3.4 and Graphical Editing Framework (GEF).
Wireless Package	Installs Wireless Package. This contains EclipseME and Antenna, which require a wireless toolkit to use.
ICP Windows	Installs IMS Client Platform (ICP) for Windows.
UIQ SDK + P1	Installs the Symbian Emulator, UIQ 3 SDK, and extension package for the Sony Ericsson P1 handset.
Sample Code	Installs sample applications that illustrate how to code programs with SDS. Required if you wish to use the Tutorial that comes with SDS.

4. Select SDS and ICP Windows which are the components we want to install. When done, click Next.

   If Java is not found on the system PATH variable, you will be prompted by this window.



SDS 4.1 Setup

Choose Components

Choose which features of SDS 4.1 you want to install.

Java 1.5 was not found on the system PATH

Existing Java 1.5 Installation

Select a directory with a valid Java 1.5

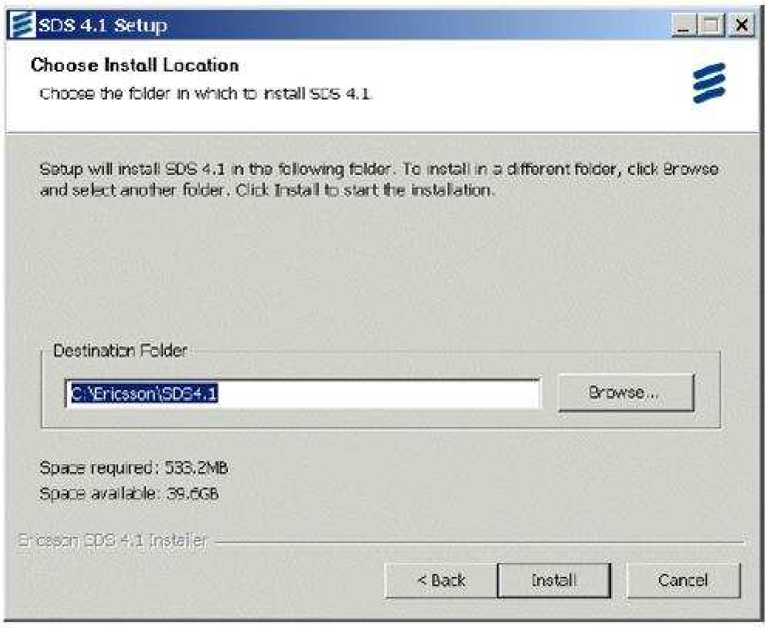
Directory: C:\Program Files\Java\jdk1.5.0_10

Ericsson SDS 4.1 Installer

Figure 5 Select Java Directory

5. Enter the path where Java 1.5 is installed on your machine or browse and select the path. When done, click **Next**.

   The **Choose Install Location** screen appears, prompting you for the SDS install path. The recommended (default) installation path is `C:\Ericsson\SDS4.1`. The path must not contain any spaces.


SDS 4.1 Setup

Choose Install Location

Choose the folder in which to install SDS 4.1.

Setup will install SDS 4.1 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Ericsson\SDS4.1

Space required: 533.2MB

Space available: 39.6GB

Ericsson SDS 4.1 Installer

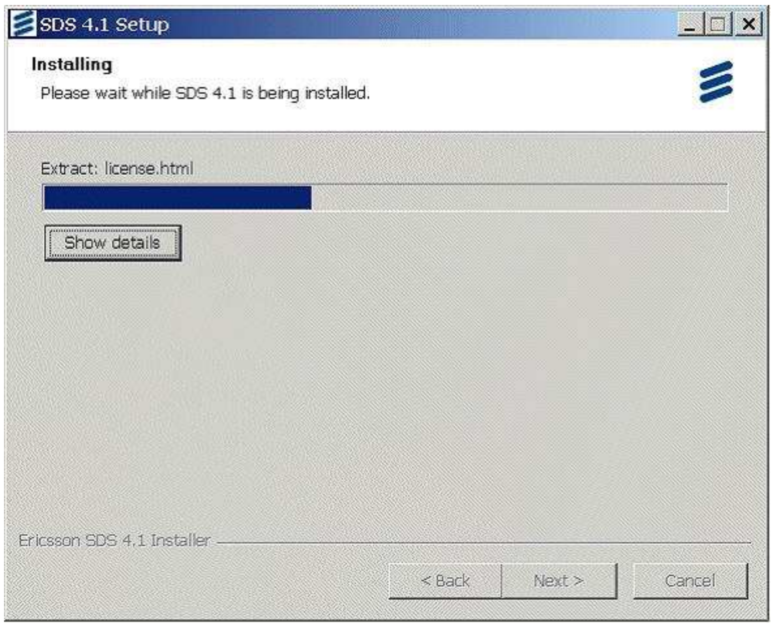
Figure 6 Choose Install Location

6. Click **Install** to accept the default C:\Ericsson\SDS4.1 installation directory.

Installation may take several minutes. A progress bar displays the status of the installation.

If you specify your own installation path for SDS, it must **not** contain spaces. Installation will fail otherwise.

If you want to display the details of the installation, click **Show Details**.


SDS 4.1 Setup

Installing

Please wait while SDS 4.1 is being installed.

Extract: license.html

Ericsson SDS 4.1 Installer

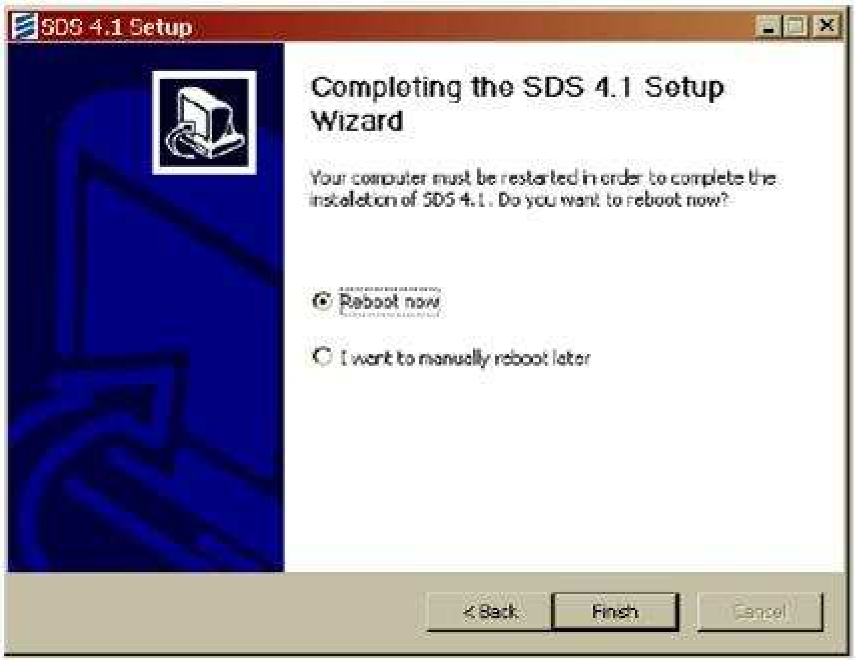
Figure 7 SDS 4.1 Installation Progress

1.3 Completing the SDS Installation

Once SDS is installed, the **Installation Complete** window is displayed. The installation procedure of all components needed to use SDS is complete.

However, to run SDS you must reboot your computer.

1. Click next. The Completing the SDS 4.1 Setup Wizard window is displayed.



SDS 4.1 Setup

Completing the SDS 4.1 Setup Wizard

Your computer must be restarted in order to complete the installation of SDS 4.1. Do you want to reboot now?

Figure 8 Completing the SDS 4.1 Wizard

2. Select the Reboot now radio button and click Finish.

   Any open applications on your desktop are closed and your system reboots.

   An SDS 4.1 item is added to your Start > Programs menu.

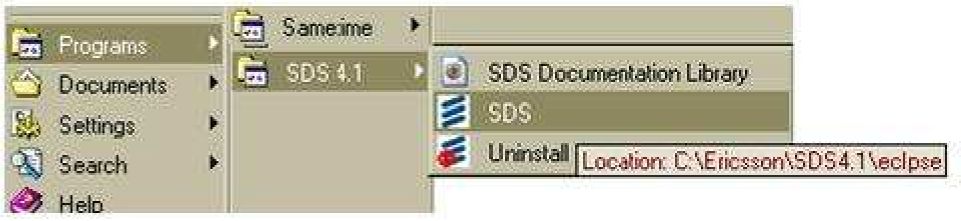


Figure 9 SDS in the Window Programs Menu

The selected SDS components and all required plug-ins are installed on your computer.

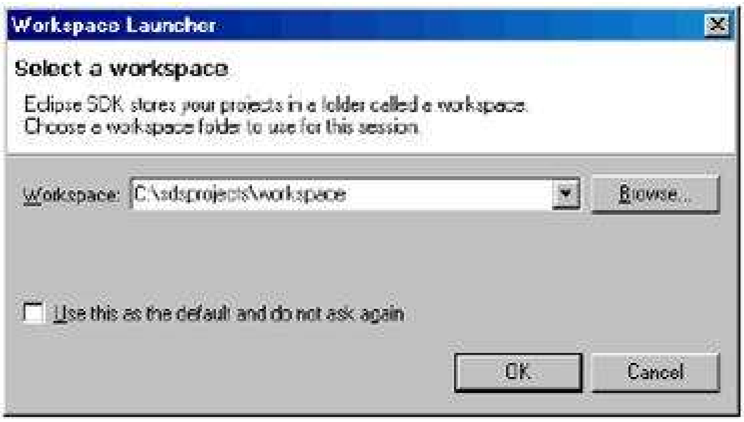
1.3.1 Starting SDS

2. To start SDS, in Windows click **Start > Programs > SDS 4.1 > SDS**.

An Eclipse splash screen is displayed while SDS loads. This could take a few moments.

Figure 10 Eclipse Splash Screen

The **Workspace Launcher** appears, prompting you to choose your workspace for this session.



Workspace Launcher

Select a workspace

Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\sdsprojects\workspace

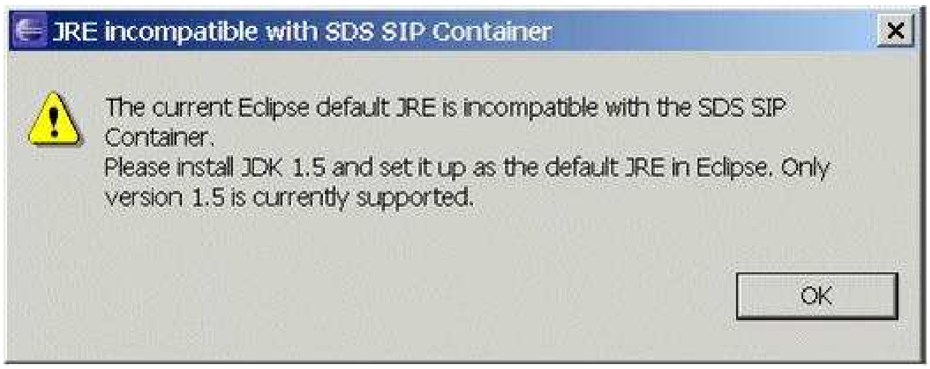
Figure 11 SDS Workspace Launcher

3. Enter the path to the workspace and click **OK**.

Note: Select **Use this as the default and do not ask again** if you do not wish to be prompted with this screen every time you start SDS.

If you are installing SDS 4.1 on a laptop computer or if your DHCP IP address is subject to change, do not select this option, as you may need to change workspaces to accommodate different IPs.

If you do not have correct default JRE settings, you will receive this warning.


JRE incompatible with SDS SIP Container

The current Eclipse default JRE is incompatible with the SDS SIP Container.
Please install JDK 1.5 and set it up as the default JRE in Eclipse. Only version 1.5 is currently supported.

Figure 12 Incompatible JRE Warning Window

4. Click **OK**. Consult the Service Development Studio (SDS) 4.1 Developer's Guide, 198 17-APR 901 753/2 for more information on how to configure the default JRE.

The first time you open Eclipse, you are presented with a **Welcome** screen.

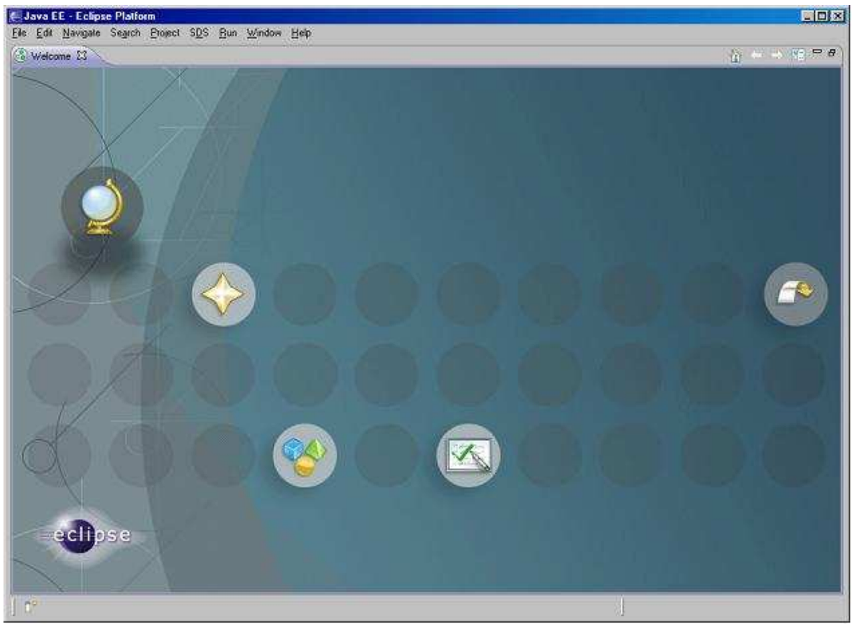
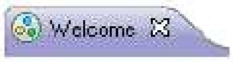
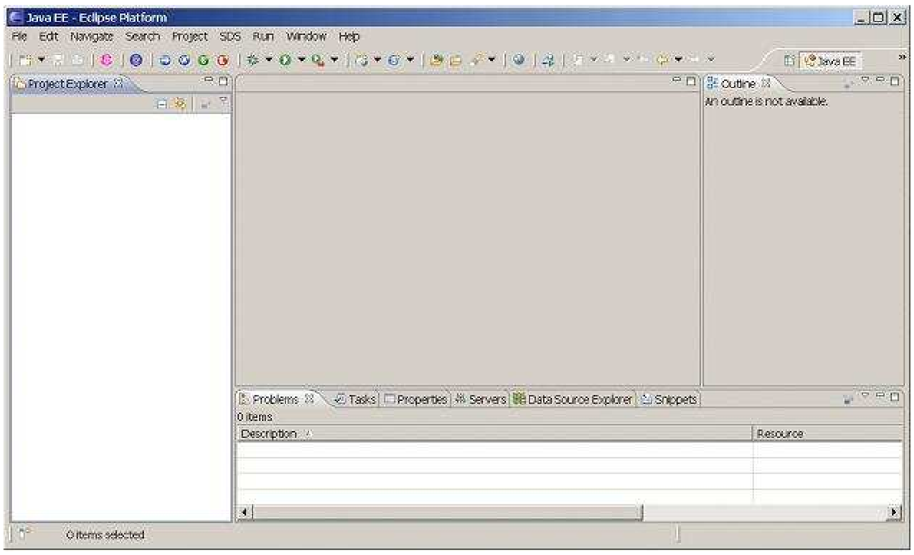

Java EE - Eclipse Platform
File Edit Navigate Search Project SDS Run Window Help
Welcome
eclipse

Figure 13 Eclipse Welcome Screen

5. Click the **X** on the  tab to close this window. It will not appear again.

   The Eclipse workbench appears.



An outline is not available.

0 items

Description	Resource

0 items selected

Figure 14 Eclipse Workbench Window

- If this is the first time you are launching SDS, the workbench is blank.

Note: If you have reinstalled SDS, you may encounter a workbench restore error. This occurs if SDS has modified a view or perspective that you had displayed when you closed SDS. The warning message can be safely dismissed without any consequences.

2. App Server Web Services

As follows the necessary steps to be able to create with AXIS the Web Services, which will be inserted in the App Server, are explained.

Apache Axis is nothing else than an open source, XML based Web service framework. It consists of a Java and a C++ implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, developers can create interoperable, distributed computing applications.

Firstly we will revise the necessary tools to be able to proceed.

2.1 Tools

- Java Virtual Machine: Needed to be able to execute the rest of tools:

  http://www.java.com/es/download/

- Apache Tomcat: Web Services application that can be downloaded from:

  http://tomcat.apache.org/download-60.cgi

- Apache Axis2: Web Services development kit. It allows the Web Services execution. Downloading link:

  http://ftp.udc.es/apache-dist/ws/axis2/1_4/axis2-1.4-war.zip

- Eclipse Project: Development platform needed to realize the source code and to generate the services through Axis2 plugins for Eclipse:

  http://www.eclipse.org/downloads/

  - Apache Axis 2 Service Archive Generator: Plugin for the Web Service generation:

    http://www.apache.org/dyn/mirrors/mirrors.cgi/ws/axis2/tools/1_4/axis2-eclipse-service-archiver-wizard-1.4.zip

  - Apache Axis2 Code Generator: Plugin for the WDSL and Web Service clients generation:

    http://www.apache.org/dyn/mirrors/mirrors.cgi/ws/axis2/tools/1_4/axis2-eclipse-codegen-wizard-1.4.zip

Once all the needed tools have been downloaded we shall proceed to their installation. First we must install the Java virtual machine, if we have not done it yet. As follows we must install Tomcat and Eclipse.

Once we have Tomcat and Eclipse properly working we must decompress the Kit Axis2 *.war and copy it into the “Webapps” folder from the Tomcat root directory; to end the installation we must start the server.

Both Axis2 plugins for Eclipse must be decompressed and copied into the plugins folder of the Eclipse root directory.

2.2 Generating WSDL from the Java code

1. To create the WSDL we will choose the interface that provides the functionalities. Over the project we want to use, we click with the secondary button and choose "New>>Other" and we select "Axis Wizards>>Axis Code Generator".

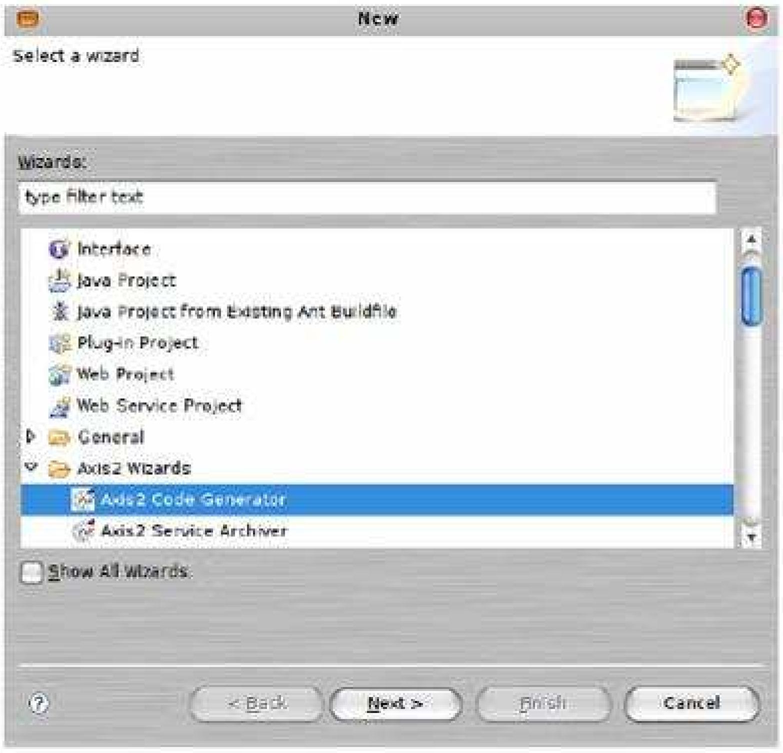
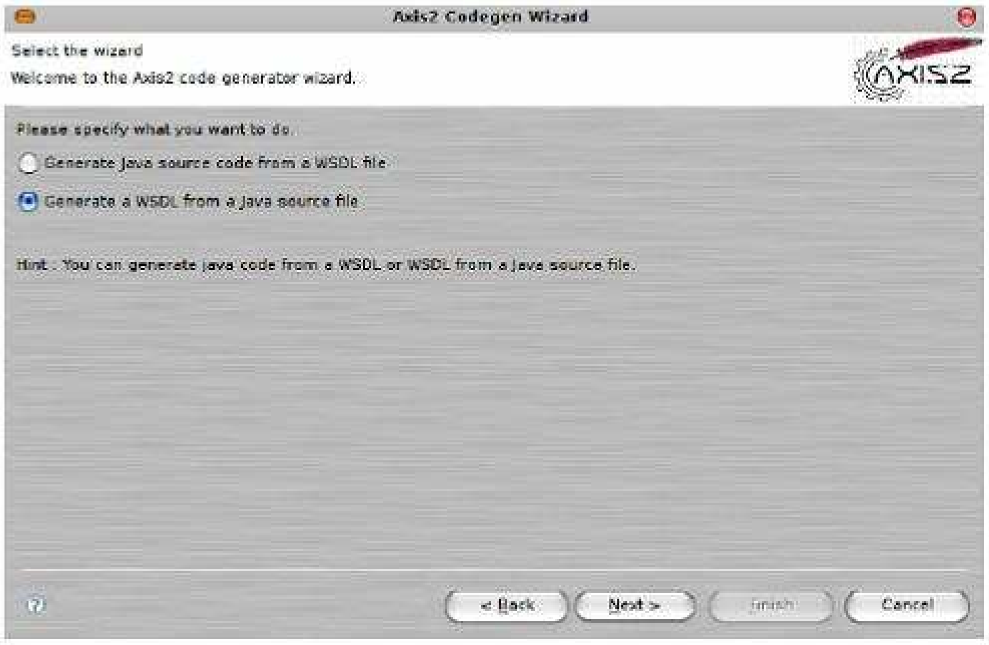


Figure 15 Plugin selection

2. As follows we will select the option of WSDL generation.



Select the wizard
Welcome to the Axis2 code generator wizard.

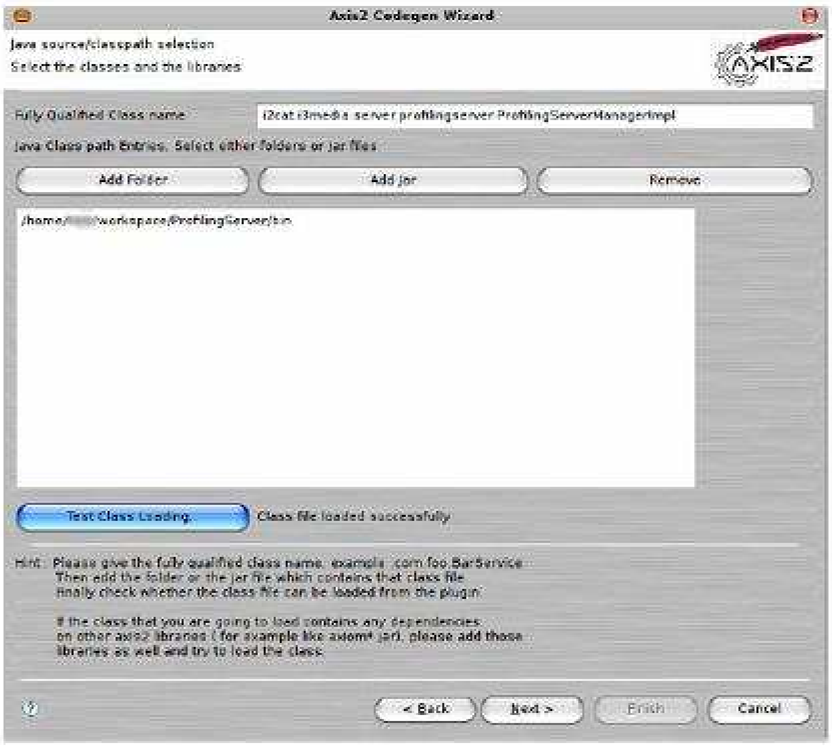
Please specify what you want to do.

( ) Generate Java source code from a WSDL file
(●) Generate a WSDL from a Java source file

Hint : You can generate java code from a WSDL or WSDL from a Java source file.

Figure 16 Choose the type of generation

3. We select the class that will be read indicating its route, “Add folder”.



Axis2 Codegen Wizard

Java source/classpath selection

Select the classes and the libraries

Fully Qualified Class name: i2cat.i3media.server.profilingserver.ProfilingServerManagerImpl

Java Class path Entries. Select either folders or jar files

/home/[redacted]/workspace/ProfilingServer/bin

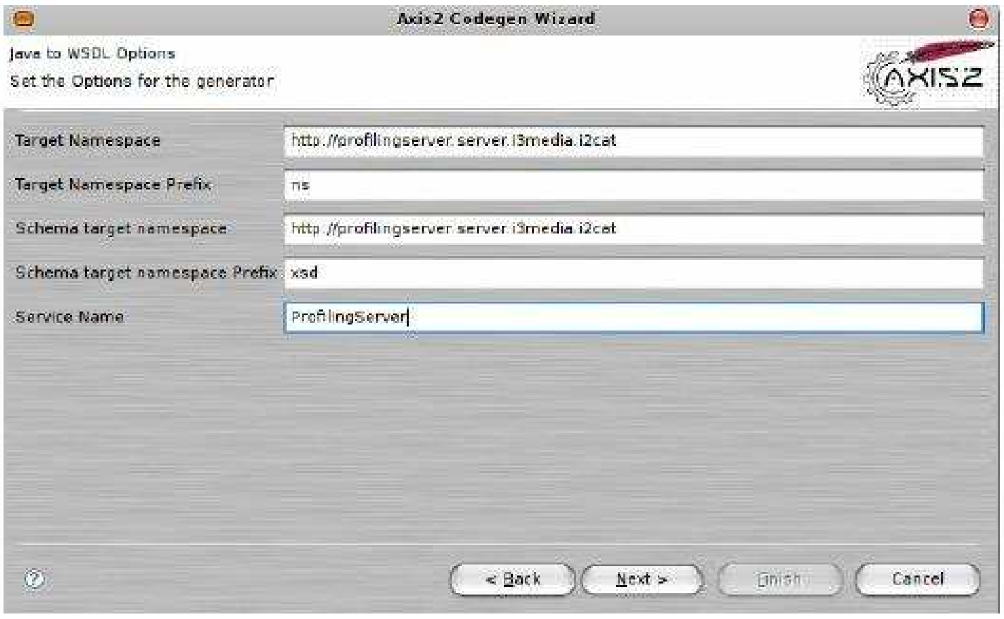
Class file loaded successfully

Hint : Please give the fully qualified class name, example :com.foo.BarService
Then add the folder or the jar file which contains that class file.
Finally check whether the class file can be loaded from the plugin.

If the class that you are going to load contains any dependencies
on other axis2 libraries (for example like axiom*.jar), please add those
libraries as well and try to load the class.

Figure 17 Interface selection

4. Subsequently we will be able to change the name of the Web Service, if we prefer it.



Axis2 Codegen Wizard

Java to WSDL Options

Set the Options for the generator

Target Namespace: http://profilingserver.server.i3media.i2cat

Target Namespace Prefix: ns

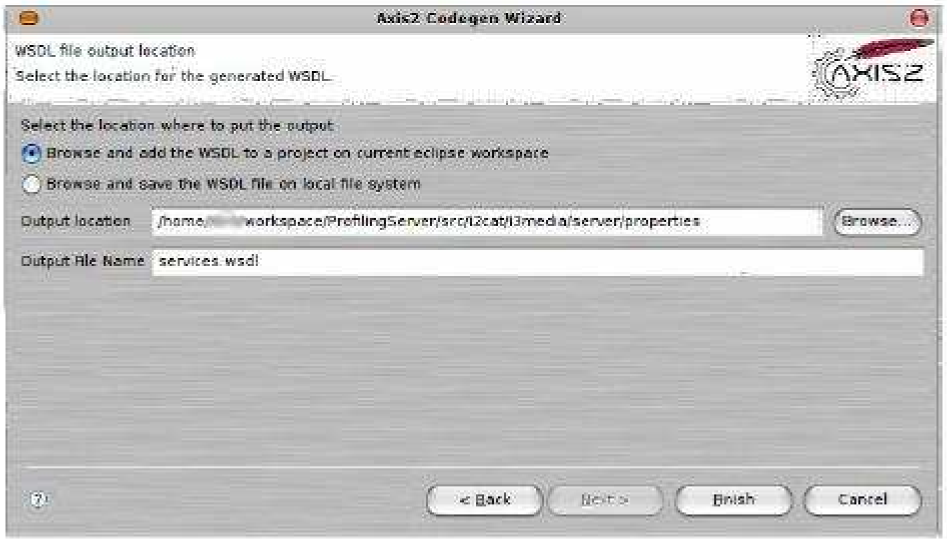
Schema target namespace: http://profilingserver.server.i3media.i2cat

Schema target namespace Prefix: xsd

Service Name: ProfilingServer

Figure 18 Name for the Web Service

5. We select were to save and the name of the WSDL file, afterwards we save.



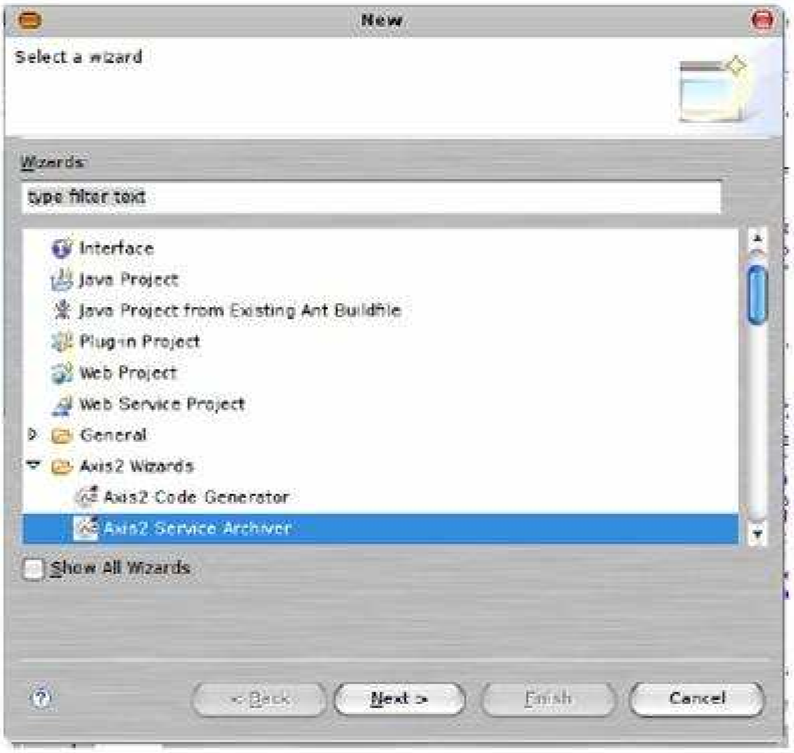
WSDL file output location
Select the location for the generated WSDL.

Select the location where to put the output
[✓ Browse and add the WSDL to a project on current eclipse workspace]
Browse and save the WSDL file on local file system
Output location /home/[redacted]workspace/ProfilingServer/src/i2cat/i3media/server/properties
Output File Name services.wsdl

Figure 19 Last step

2.3 Generating a Web Service

1. To set up the service, over the project we want to use, we click with the secondary button and choose “New>>Other”, after we select “Axis Wizards>>Axis2 Service Archive Generator”.



Select a wizard

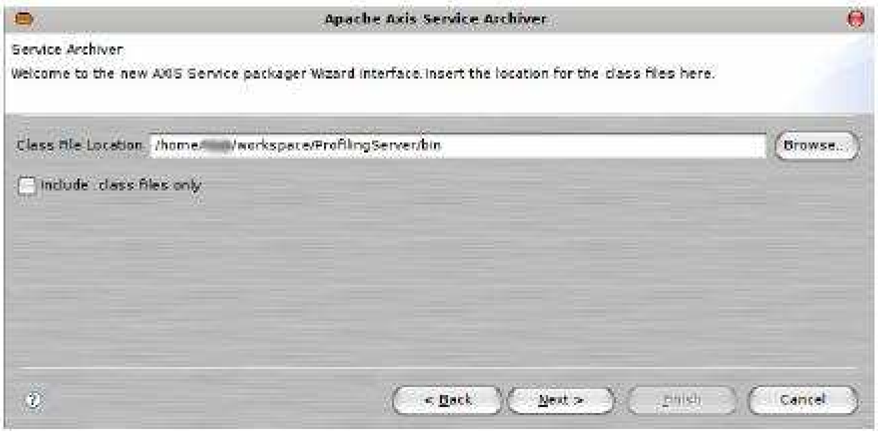
Wizards:
type filter text

Interface
Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
Web Project
Web Service Project
General
Axis2 Wizards
Axis2 Code Generator
[✓ Axis2 Service Archiver]

Show All Wizards

Figure 20 Plugin selection

2. The next step is indicating the path of the service binaries. Do not mark the box “Include .class files only” if the service needs files from other types.



Service Archiver

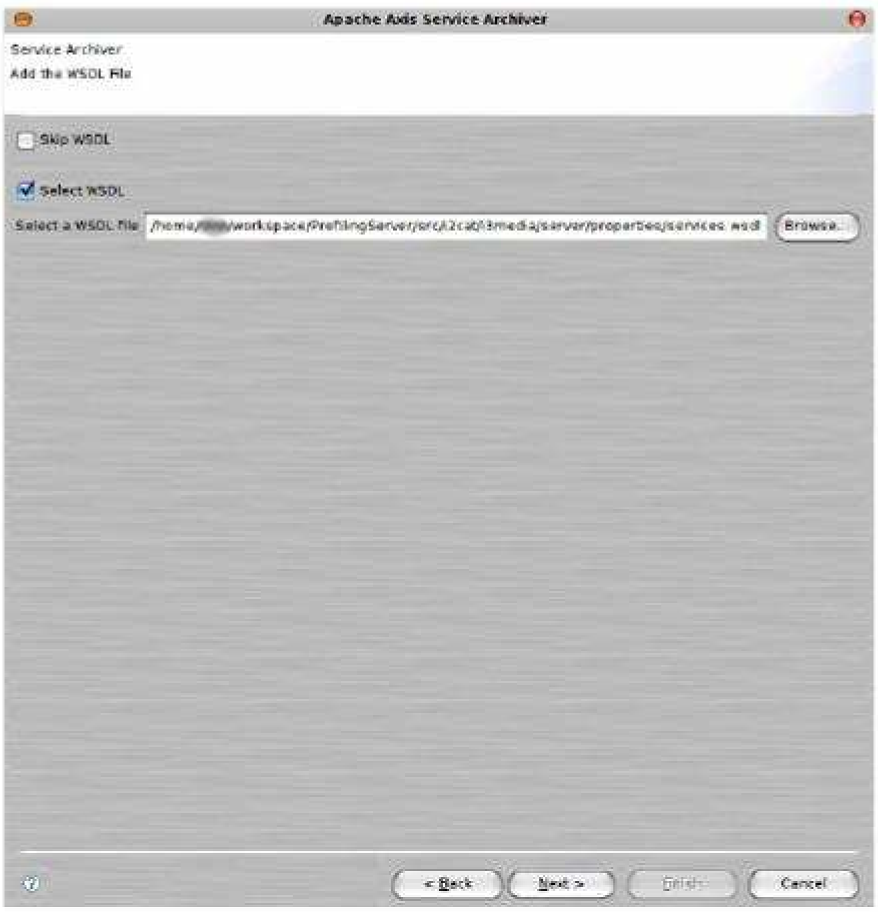
Welcome to the new AXIS Service packager Wizard interface. Insert the location for the class files here.

Class File Location: /home/[redacted]/workspace/ProfilingServer/bin Browse...

☐ Include .class files only

Figure 21 Selection of the path of the binaries

3. Afterwards we select the WSDL created in section 2.2.



Service Archiver

Add the WSDL File

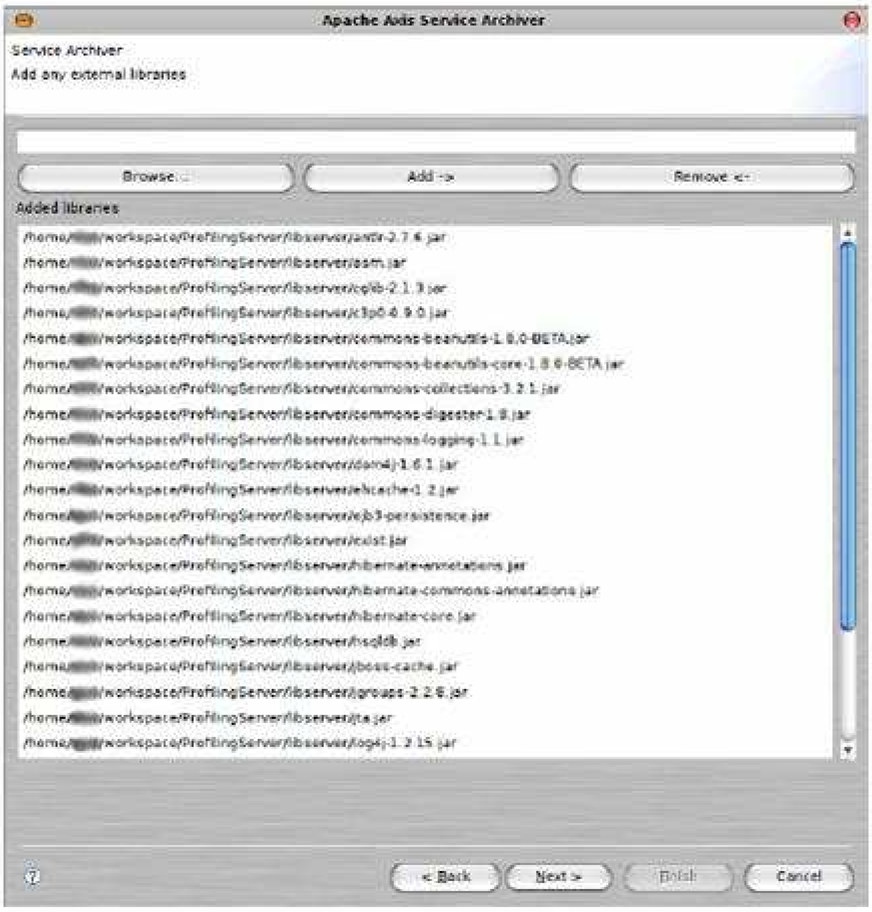
☐ Skip WSDL

☑ Select WSDL

Select a WSDL file: /home/[redacted]/workspace/ProfilingServer/src/i2cat/i3media/server/properties/services.wsdl Browse...

Figure 22 WSDL selection

4. In this step we will select all the libraries that may be needed for the correct Web Service operation.



Apache Axis Service Archiver

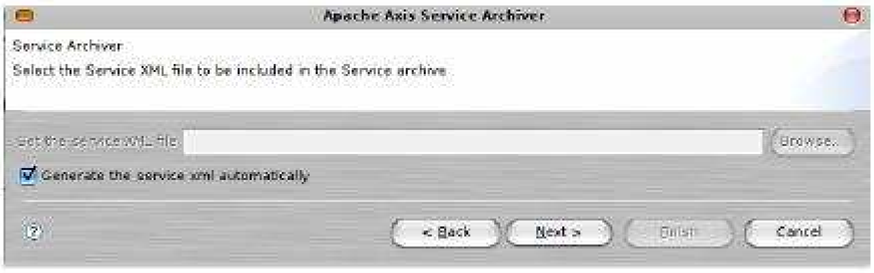
Service Archiver
Add any external libraries

Added libraries

```
/home/[redacted]/workspace/ProfilingServer/libserver/antlr-2.7.6.jar
/home/[redacted]/workspace/ProfilingServer/libserver/asm.jar
/home/[redacted]/workspace/ProfilingServer/libserver/cglib-2.1.3.jar
/home/[redacted]/workspace/ProfilingServer/libserver/c3p0-0.9.0.jar
/home/[redacted]/workspace/ProfilingServer/libserver/commons-beanutils-1.8.0-BETA.jar
/home/[redacted]/workspace/ProfilingServer/libserver/commons-beanutils-core-1.8.0-BETA.jar
/home/[redacted]/workspace/ProfilingServer/libserver/commons-collections-3.2.1.jar
/home/[redacted]/workspace/ProfilingServer/libserver/commons-digester-1.8.jar
/home/[redacted]/workspace/ProfilingServer/libserver/commons-logging-1.1.jar
/home/[redacted]/workspace/ProfilingServer/libserver/dom4j-1.6.1.jar
/home/[redacted]/workspace/ProfilingServer/libserver/ehcache-1.2.jar
/home/[redacted]/workspace/ProfilingServer/libserver/ejb3-persistence.jar
/home/[redacted]/workspace/ProfilingServer/libserver/exist.jar
/home/[redacted]/workspace/ProfilingServer/libserver/hibernate-annotations.jar
/home/[redacted]/workspace/ProfilingServer/libserver/hibernate-commons-annotations.jar
/home/[redacted]/workspace/ProfilingServer/libserver/hibernate-core.jar
/home/[redacted]/workspace/ProfilingServer/libserver/hsqldb.jar
/home/[redacted]/workspace/ProfilingServer/libserver/jboss-cache.jar
/home/[redacted]/workspace/ProfilingServer/libserver/jgroups-2.2.8.jar
/home/[redacted]/workspace/ProfilingServer/libserver/jta.jar
/home/[redacted]/workspace/ProfilingServer/libserver/log4j-1.2.15.jar
```

Figure 23 List of the selected JARs

5. We dispose that it is possible to generate the XML file unless we wish to specify one.



Apache Axis Service Archiver

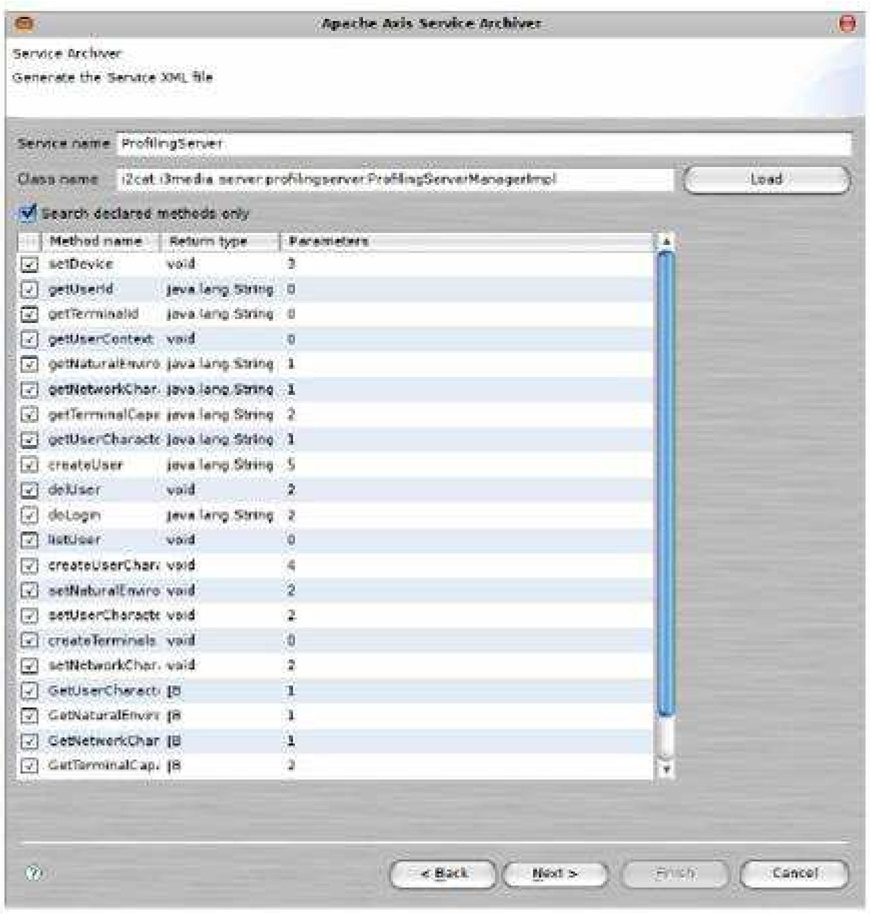
Service Archiver
Select the Service XML file to be included in the Service archive

Set the service XML file

[✓] Generate the service xml automatically

Figure 24 Generation "services.xml"

6. In the next step we will be able to indicate which methods we wish to include in the Web Service. To do so we have to select the interface which has to be the same used to generate the WSDL in section 2.2.



Service Archiver
Generate the Service XML file

Service name: ProfilingServer
Class name: i2cat.i3media.server.profilingserver.ProfilingServerManagerImpl

[✓] Search declared methods only

	Method name	Return type	Parameters
✓	setDevice	void	3
✓	getUserId	java.lang.String	0
✓	getTerminalId	java.lang.String	0
✓	getUserContext	void	0
✓	getNaturalEnviro	java.lang.String	1
✓	getNetworkChar	java.lang.String	1
✓	getTerminalCaps	java.lang.String	2
✓	getUserCharacte	java.lang.String	1
✓	createUser	java.lang.String	5
✓	delUser	void	2
✓	doLogin	java.lang.String	2
✓	listUser	void	0
✓	createUserChar	void	4
✓	setNaturalEnviro	void	2
✓	setUserCharacte	void	2
✓	createTerminals	void	0
✓	setNetworkChar	void	2
✓	GetUserCharact	[B	1
✓	GetNaturalEnvir	[B	1
✓	GetNetworkChar	[B	1
✓	GetTerminalCap	[B	2

Figure 25 Method list of the interface

7. Finally we will indicate where we wish to save the Web Service and the name of the future file. To deploy it onto the Tomcat Axis 2 save it in “TOMCAT_DIR\Webapps\axis2\WEB-INF\services” where TOMCAT is the root directory.

2.4 How to generate source code of the client

1. To create the Web Service Client, over the project we want to use, we click with the secondary button and select “New>>Other” and then “Axis Wizards>>Axis2 Code Generator”.

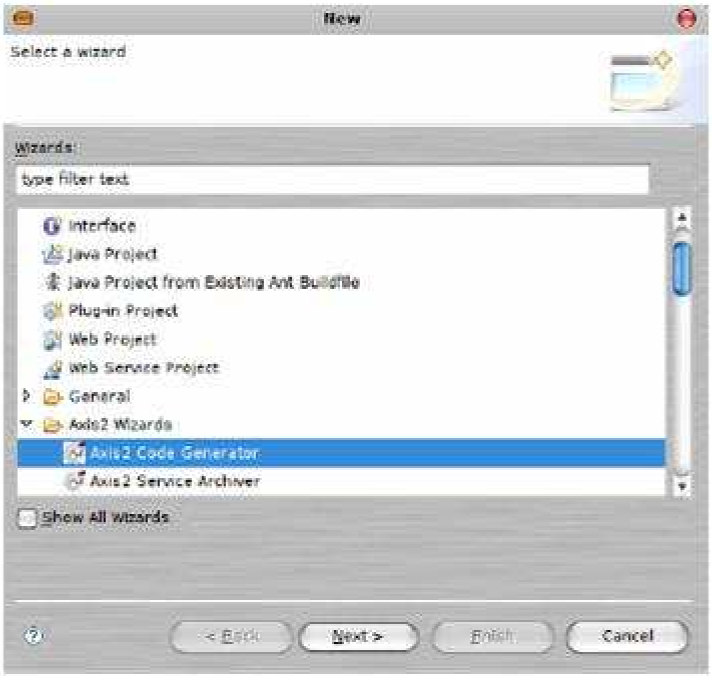
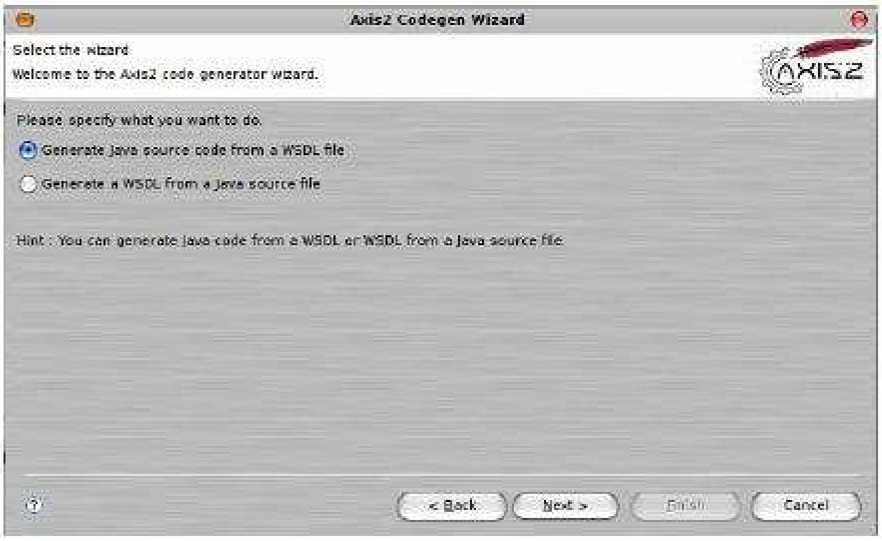


Figure 26 Plugin selection

2. Now we select the option “Generate Java source code from a WSDL file”.


Axis2 Codegen Wizard

Select the wizard

Welcome to the Axis2 code generator wizard.

Please specify what you want to do.

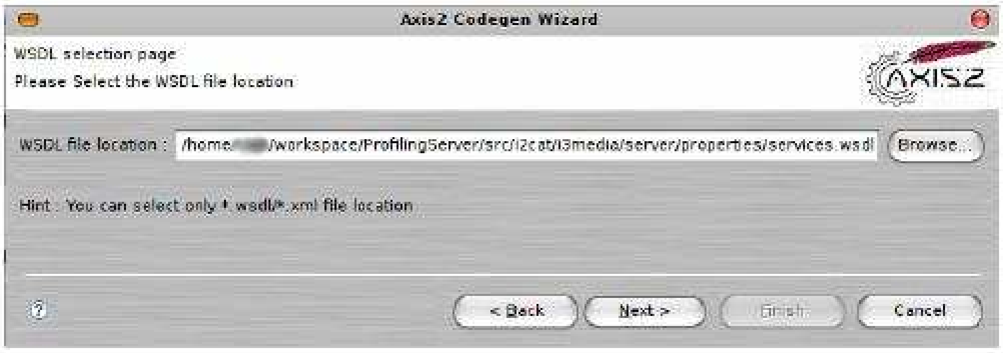
[✓ Generate Java source code from a WSDL file]

Generate a WSDL from a Java source file

Hint : You can generate java code from a WSDL or WSDL from a java source file

Figure 27 Choosing the type of generation

3. Now we select the WSDL that we generated in section 2.2.



Axis2 Codegen Wizard

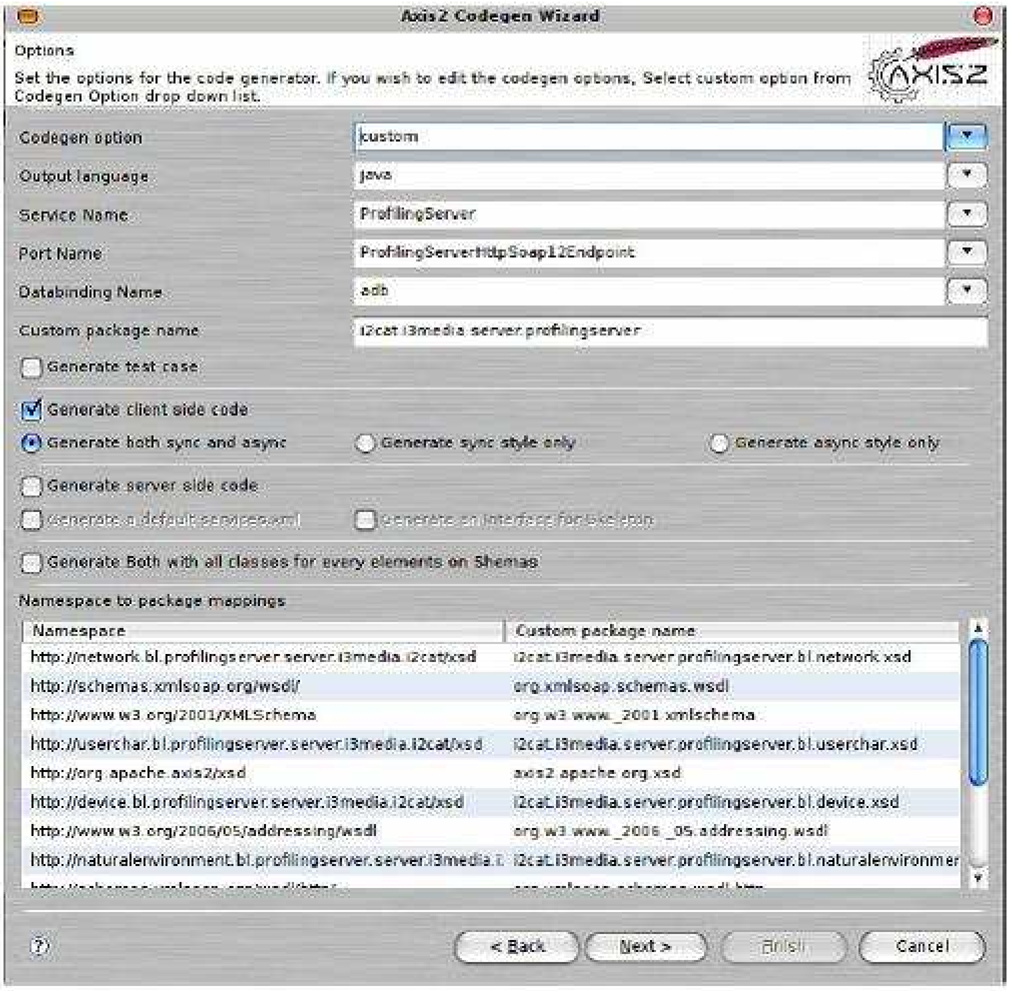
WSDL selection page
Please Select the WSDL file location

WSDL file location : /home/[redacted]/workspace/ProfilingServer/src/i2cat/i3media/server/properties/services.wsdl

Hint : You can select only *.wsdl/*.xml file location

Figure 28 WSDL selection

4. In the following screen in “Codegen Option” we must select “custom” if we wish to change the client preferences. In “Custom package name” we can select the output path of the generated source code.



Axis2 Codegen Wizard

Options
Set the options for the code generator. If you wish to edit the codegen options, Select custom option from Codegen Option drop down list.

Codegen option	custom
Output language	java
Service Name	ProfilingServer
Port Name	ProfilingServerHttpSoap12Endpoint
Databinding Name	adb
Custom package name	i2cat.i3media.server.profilingserver

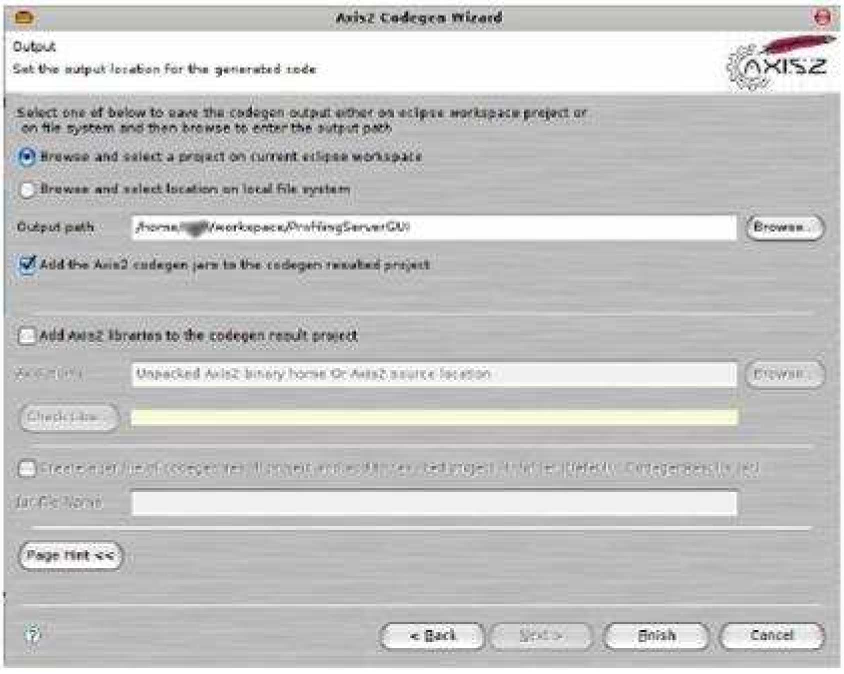
Generate test case
[✓ Generate client side code]
[✓ Generate both sync and async] Generate sync style only Generate async style only
Generate server side code
Generate a default services.xml Generate an interface for skeleton
Generate Both with all classes for every elements on Shemas

Namespace to package mappings

Namespace	Custom package name
http://network.bl.profilingserver.server.i3media.i2cat/xsd	i2cat.i3media.server.profilingserver.bl.network.xsd
http://schemas.xmlsoap.org/wsdl/	org.xmlsoap.schemas.wsdl
http://www.w3.org/2001/XMLSchema	org.w3.www._2001.xmlschema
http://userchar.bl.profilingserver.server.i3media.i2cat/xsd	i2cat.i3media.server.profilingserver.bl.userchar.xsd
http://org.apache.axis2/xsd	axis2.apache.org.xsd
http://device.bl.profilingserver.server.i3media.i2cat/xsd	i2cat.i3media.server.profilingserver.bl.device.xsd
http://www.w3.org/2006/05/addressing/wsdl	org.w3.www._2006._05.addressing.wsdl
http://naturalenvironment.bl.profilingserver.server.i3media.i	i2cat.i3media.server.profilingserver.bl.naturalenvironmer

Figure 29 Client configuration

5. At last we indicate the destination path of the code indicating that it is in the Eclipse workspace directory. We must select “Add the Axis2 codegen.jar...” if we want the code to dispense with the imports, calling the classes by their path, e.g. com.apache.common. ...



Axis2 Codegen Wizard

Output
Set the output location for the generated code

Select one of below to save the codegen output either on eclipse workspace project or
on file system and then browse to enter the output path

[✓ Browse and select a project on current eclipse workspace]

Browse and select location on local file system

Output path: /home/[redacted]/workspace/ProfilingServerGUI

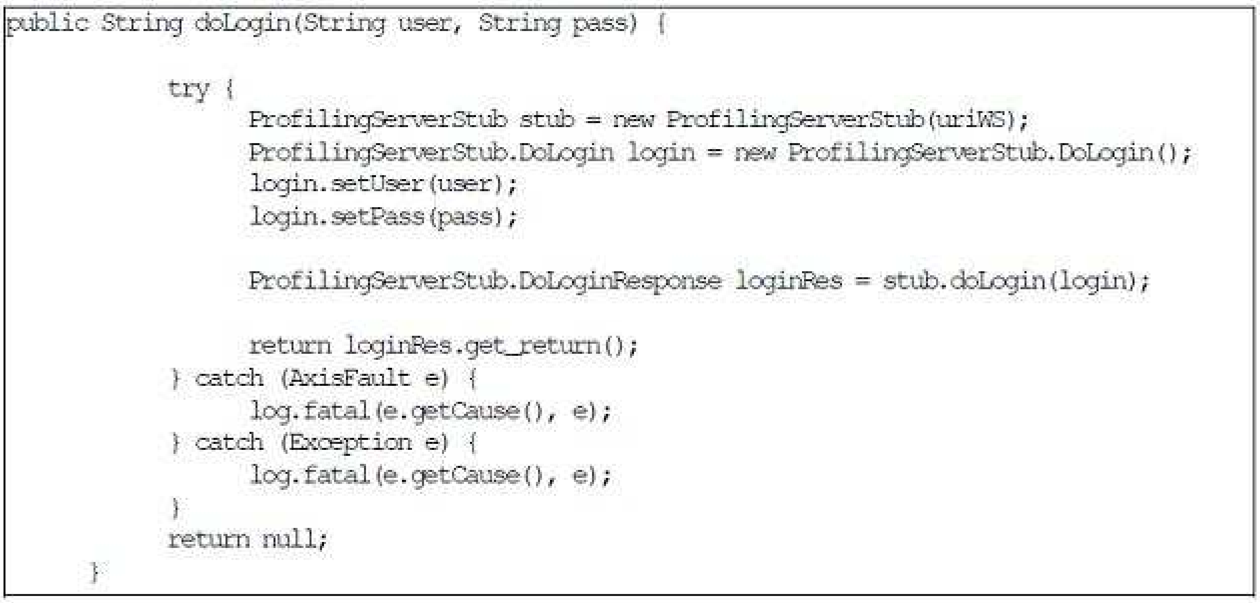
[✓ Add the Axis2 codegen jars to the codegen resulted project]

Add Axis2 libraries to the codegen result project

[illegible]: Unpacked Axis2 binary home Or Axis2 source location

Figure 30 Last step

6. Once we have the Web Service Client, we have to create a class to interact with. In the following example a login petition is shown.



```
public String doLogin(String user, String pass) {

            try {
                  ProfilingServerStub stub = new ProfilingServerStub(uriWS);
                  ProfilingServerStub.DoLogin login = new ProfilingServerStub.DoLogin();
                  login.setUser(user);
                  login.setPass(pass);

                  ProfilingServerStub.DoLoginResponse loginRes = stub.doLogin(login);

                  return loginRes.get_return();
            } catch (AxisFault e) {
                  log.fatal(e.getCause(), e);
            } catch (Exception e) {
                  log.fatal(e.getCause(), e);
            }
            return null;
      }
```

Figure 31 Example of the interaction with the Web Service client

Following this example we are able to call the methods that are in the server as if we had them in local.