Chapter 17

JAVA IMS Mobile Application Development

Javier Díaz, Claudia Queiruga, and Jorge Rosso

Contents

17.1 Motivation

During the year 2008, the Computer Science School of the UNLP (National University of La Plata) signed a cooperation agreement with Ericsson Company in Argentina with the goal of undertaking activities to promote innovation, training, and research in the area of NGN (new generation networking) and IMS (IP multimedia subsystem), primarily addressing issues related to the development of Java applications on this new vision of converged networks.

Students from the "Software Lab" subject, which belongs to the fourth year of the Computer Science and Systems degrees, learn to develop Java client applications, both for desktop and mobile devices. We consider that having a high-quality converged network infrastructure like IMS, which makes it possible to integrate services that are usually provided by heterogeneous networks, is a challenge for the construction of new applications on mobile devices, where communication and sharing play a central role.

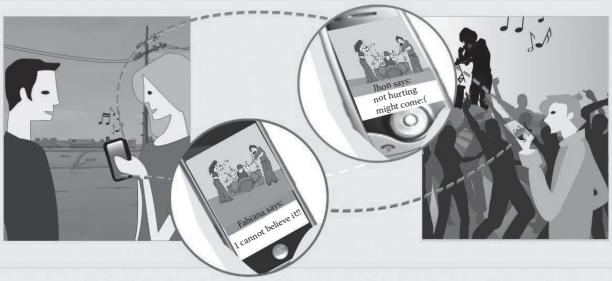
According to Patrik Heldmund, innovation area manager in Ericsson-Argentina, the expectations in the development of Java applications on IMS is to find innovative ideas on how to use the technology and create services that create real value for the consumer. Ultimately, the driving force for network transformation will be related to applications and end user services. The cooperation between Ericsson and UNLP create interesting synergy effects for the students, the university, and Ericsson, and is a successful example of the need to create more industry–university
AQ1 projects in this field. Examples of applications on IMS

To fully understand the services provided by the IMS architecture, we illustrate an example of daily life according to current communication behavior, where mobility and device diversity are already in common use. IMS encourages new experiences for users, promoting the use of a convergent network which guarantees a comfortable bandwidth, quality of service, and device neutrality. When the odds of communication increase, physical distances become shorter, and media simultaneity and device diversity further enrich the experience.

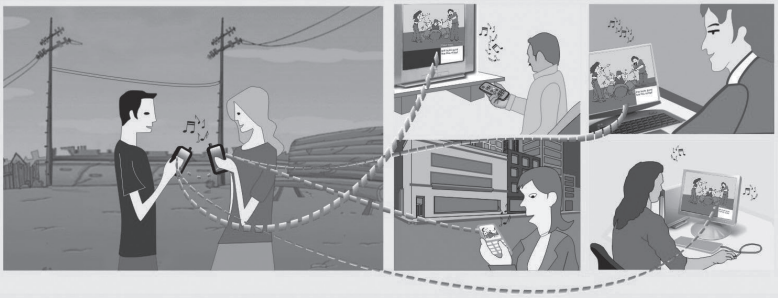
Let us imagine a situation in which a group of friends share the experience of a rock concert by means of their devices connected to an IMS network. Following, we provide a possible IMS use case by means of a simple story.

David goes to the first show of his friend Charlie's band in the most important venue of the city and remembers that his friends Fabiana and Pedro had to leave town unexpectedly. David decides to share with them the moment in which Charlie does his drum solo by sending them a live video which he captures with his cellphone camera.

"I cannot beleive it!! Charlie is playing incredibly well!!" Fabiana writes David, as she shows the video to Pedro. David and Fabiana exchange opinions while they enjoy the fine music.



Pedro surprises his father by sending him an invitation to his TV, and while they share the video they talk about Charlie's new songs. Fabiana also contacts Charlie's brothers, who enjoy their artist brother's music miles away from their birth city, in their PCs at their office.



17.2 What Is IMS?

The major evolution in fixed and mobile telephony networks occurred in the last 20 years. Today, we live in a world in which digital communication has changed the way people communicate. Mobile telephony has a central role in this change, as it is not only used for "talking," but also for capturing and reproducing videos, taking pictures, consulting work schedules, news pages, using dynamic maps, etc.

Historically, voice, data, and video services have been provided on dedicated network infrastructures, even by different telecommunication providers. In turn, it took multiple types of devices exclusively designed to access the services provided by each type of network (cell phone, notebook, fixed telephone, TV, etc.) using different identities and without the possibility of "moving" (roaming) in a transparent way among those devices. These different networks have offered us multiple, powerful, and attractive ways of communication; nevertheless, the services offered behave as islands (video, voice, data, email, etc.) with no synergy between them.

Figure 17.1 shows the services offered by each access technology: cell phone, cable modem (TV and Internet), digital subscriber line (DSL) (fixed telephony and Internet) and Wi-Fi. The limitations given by the base technology hinder agility in broadening the range of services offered as well as their integration by means of different channels.

Classic telephony employs a scheme based on an end device connected to a circuit-switched network with services supplied only by the provider of telephony or specific services. This scheme tends to disappear in the evolution of "networks."

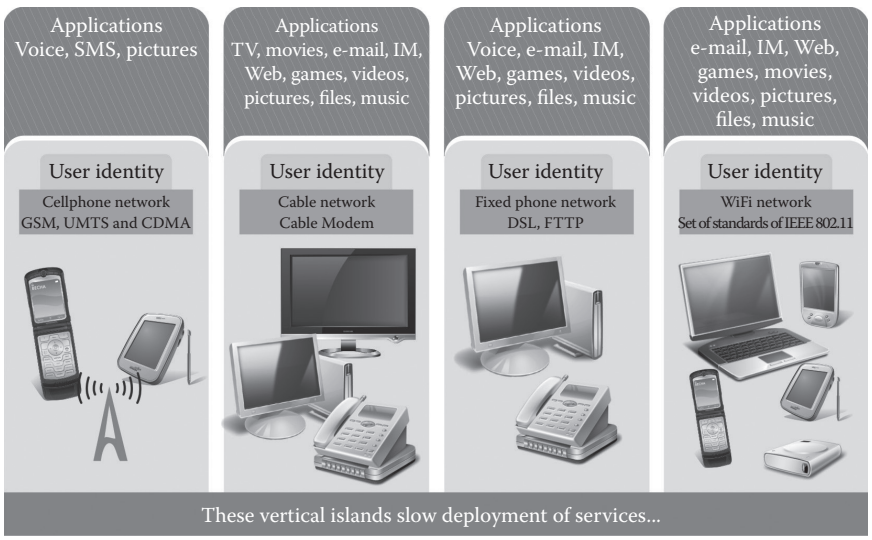


Figure 17.1 Without IMS, network functionality exists in silos, slowing deployment of services.

It is a fact that, in IP networks, any workstation (host) may act as a consumer or a provider of services, giving rise to P2P networks (peer-to-peer). At a time, the most widespread service was NAPSTER [L1]. Other P2P networks emerged later, such as Kazaa [L2] and eMule [L3], which were also used to share files. Other P2P networks promote instant messaging services, from the precursor ICQ [L4] to the popular SKYPE [L5] and MSN [L6], which permit connections between peers without the need of a central Internet service provider server.

IMS is a network architecture, which converges multiple services offered over a common infrastructure for IP data transmission. IMS is a set of specifications that describes the NGN architecture for implementing IP-based telephony and multimedia services. IMS defines a complete architecture and framework that enables the convergence of voice, video, data, and mobile networking technology over an IP-based infrastructure. It fills the gap between the two most successful communication paradigms: cellular and Internet technology. Did you ever imagine that you could surf the Web, play an online game, or join a videoconference no matter where you are using your cell phone from? This is the initial vision for IMS: to provide cellular access to all the services that the Internet provides. Another example of the new services IMS provides within the world of applications is thinking of systems such as TWITTER [L7], which currently enables people to find out what their friends are doing; IMS-TWITTER would also allow us to "see" what is happening, vote online, etc. This new form of communication allows any mobile telephony device to consume and/or produce multimedia services, which combine voice on IP calls, teleconferences, file transference (music, video, general documents, etc.), web navigation, instant messaging, etc., opening the doors to attractive multimedia applications [R1]. Also, communication companies can provide users their service packages in an increasingly attractive way, combining, for example, a flat rate for voice calls over the fixed and cellular lines, broadband Internet, and IP television [R1].

The 3GPP consortium (Third Generation Partnership Project) developed the first standard for delivering "Internet services" over GPRS (General Packet Radio Service) [L8]. This specification is called R5 (Release 5) 3GPP (3GPP R5). IMS is part of that specification. Then, the vision evolved into a standard that contemplated other networks apart from GPRS, such as wireless networks, fixed telephoning, and CDMA2000 [L9]. Thus arises a new specification called R7 (Release 7) 3GPP (3GPP R7) in which the consortiums 3GPP2 [L10] and TISPAN [L11] also participated.

The IMS architecture over network telecommunications offers a complete convergence of standard services (fixed and mobile telephony, cable and satellite TV, Internet). With IMS, telecommunication operators can integrate fixed calls with mobile ones and voice with broadband Internet access, TV and video on demand, opening doors to new multimedia residential and business applications. [R1].

Figure 17.2 shows how IMS provides a common consolidated communications network. This way, the user has access to a multiplicity of different services, integrated through a single identity with the capacity to change media nimbly and improve its experience.

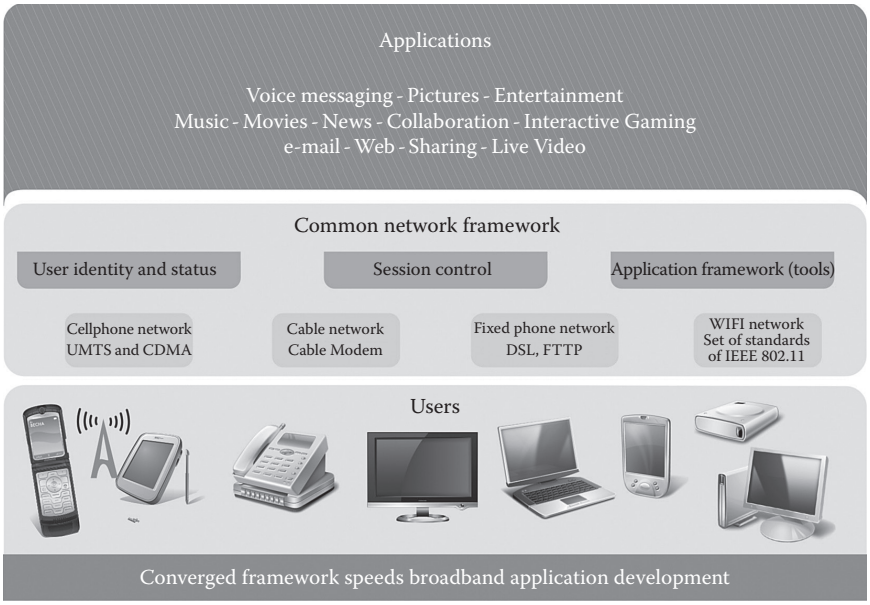


Figure 17.2 With IMS, the networks converge in a common network framework.

17.3 IMS Architecture

The IMS architecture design proposed by the 3GPP is based on functional layers. This solution facilitates the services offered through the IMS network independent of the access network. Thus, any IMS network subscriber that communicates throughout their mobile telephone or PC uses the same presence and group management service, to name a few, regardless of the access technology used. However, the potentials for bandwidth and latency of connections could be different, like the processing speed in the IMS device [R2].

In turn, layer-based design minimizes the dependency between them, facilitating the addition of new access networks to the IMS network. For example, WLAN (wireless local area network) technology was added to the IMS architecture in Release 6 and fixed broadband access in Release 7 [R2].

Figure 17.3 illustrates the layered modular design of the IMS architecture.

17.3.1 The Role of the SIP Protocol

Internet applications typically need to create and maintain sessions between participants of a communication (client-server or peer-to-peer) to facilitate the data exchange between them. In turn, the exchange of multimedia information is key in applications over an all-IP network such as IMS, and session management becomes an essential issue. This led to the adoption of a protocol for session management

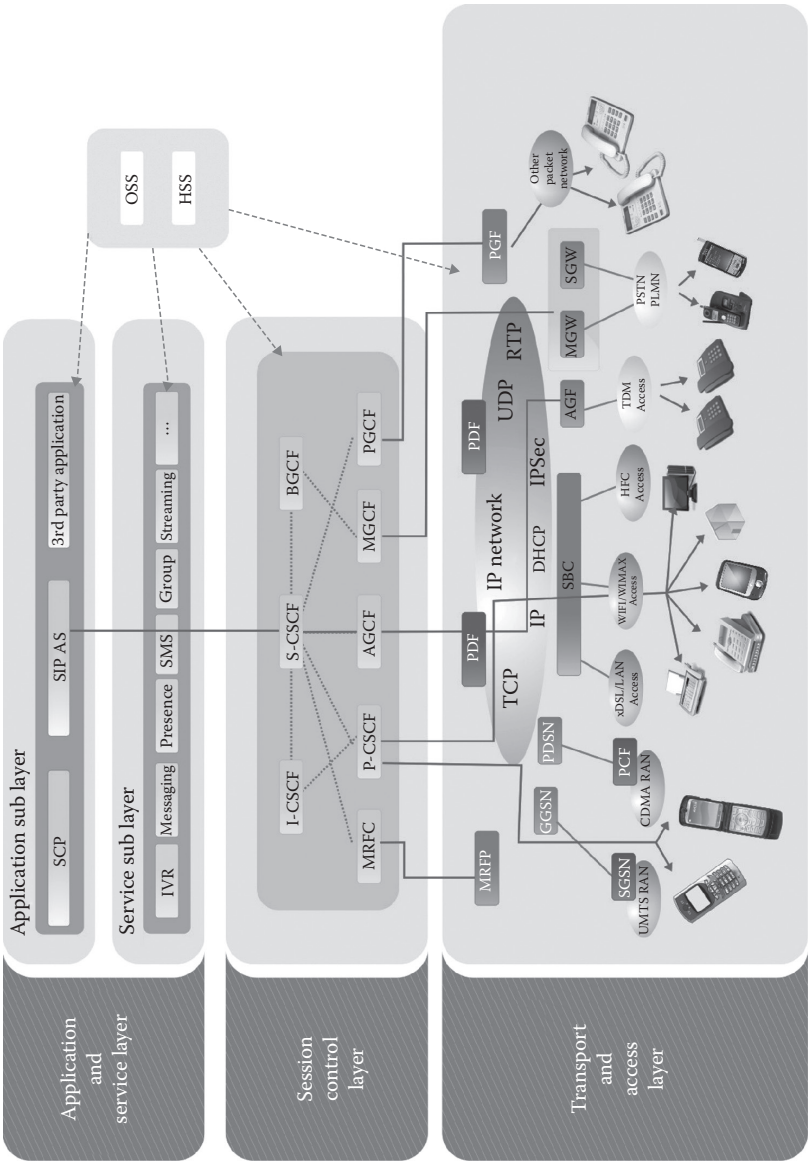


Figure 17.3 IMS architecture–layered design.

characterized by its simplicity, maturity, extensibility, and flexibility regarding mobility such as SIP. SIP (Session Initiation Protocol) is a protocol developed by IETF belonging to the application layer of TCP/IP model and is used to establish, modify, and terminate multimedia sessions in an IP network. The complete specification is available on the IETF RFC 3261 [L12]. Typical applications are related to video, voice, gaming, messaging, call control, and presence. An SIP session could be a phone call between two or more people or a videoconference. The role of SIP in these applications is, during the establishment of the session, the negotiation of the communication parameters between its participants. This negotiation involves the medium (text, voice, video, or other) transport protocol, typically RTP (Real-time Transport Protocol) for streaming and encoding technology (codec). Once these parameters are agreed upon, participants communicate using the selected method. At this point, SIP delegates the transmission in an appropriate protocol. Once the communication in completed, SIP is used again to end the session.

One of the design features of SIP is its ability to operate in collaboration with other protocols used for specific tasks, such as SDP (Session Description Protocol) to describe the parameters previously mentioned in the negotiation phase and RTP for multimedia data transport.

In a word, the role of SIP in relation to these applications is session management and consensus of its attributes, delegating transport of voice, video, and streaming to other protocols such as RTP (Real-time Transport Protocol)/RTCP (RTP Control Protocol) and SCTP (Stream Control Transmission Protocol). All this is accomplished by exchanging text messages.

User identification in IMS is based on the addressing scheme proposed by SIP. SIP addresses are similar to e-mail, prepending prefix *sip*: or *sips*: (secure SIP) and can also contain additional parameters that may indicate preferences. Some examples of URIs are

sip:tom@domain.com

sips:jerry.brown@example.com

sip:mafalda@linti.unlp.edu.ar; transport=tcp

SIP is a request-response type protocol and its messages have a format, which is similar to HTTP (Hypertext Transfer Protocol) and SMTP (Simple Mail Transfer Protocol), used in Web pages and e-mail distribution, respectively. One of the objectives of SIP design was to transform telephony into another Internet service, which is why it was based on two of the most disseminated protocols. The SIP message format consists of three parts: start line, header, and body. The start line enables to distinguish between the SIP messages that represent requests and those that represent responses. In the request SIP messages, the beginning line contains the method name, the request SIP URI, and the protocol version (SIP/2.0 now). The heading is composed of multiple fields that contain information related with the request, i.e., who initiates it, who is the receptor, and the CALL-ID (identifies a SIP dialogue). The information in these fields is the name-value type, enabling the

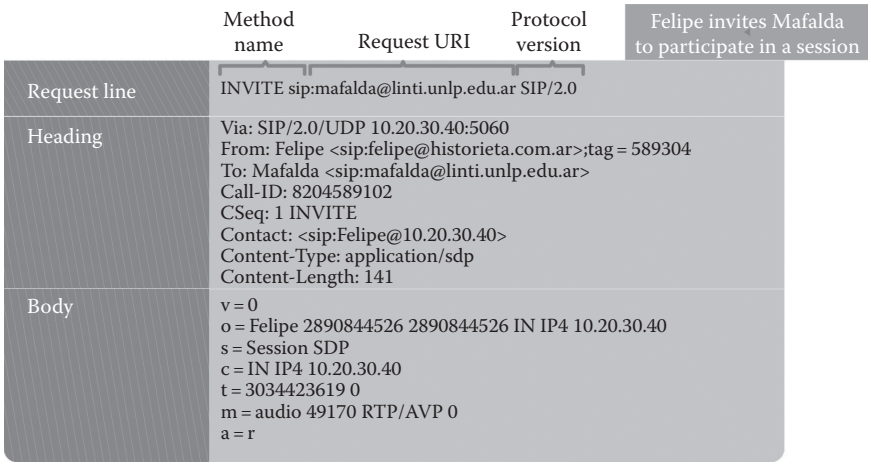


Figure 17.4 SIP Invite message.

existence of multi-value fields, like in the case of the field *Via* and *From*. Figure 17.4 presents an example of SIP INVITE request message, which in this case expresses the case of a user named Felipe who wishes to initiate a conversation with Mafalda.

The request type messages provided by the SIP protocol are described in Table 17.1 detailing the IETF RFCs in which they were defined. The main specification of the SIP RFC 3261 [L12] defined the six basic methods for session management. However, these were not sufficient to provide support services that are widely accepted today such as instant messaging and presence, which is why SIP was extended in the RFC 3428 [L17], 3265 [L15], 3856 [L20], and 3903 [L13] to include four more methods. In addition, other request type messages compete the range of messages provided by the SIP protocol defined in the RFC 3311 [L16], 2976 [L18], and 3262 [L19].

SIP response messages contain a starting line called state line conformed by the protocol version, state code, and a description of the error code (a reason phrase). The state codes are grouped in six categories also established in the RFC 3261 [L12] and summarized in Table 17.2.

Figure 17.5 shows an SIP OK response message example, which expresses that Mafalda accepts Felipe's invitation.

Next, we see the SIP dialogue between Mafalda and Felipe, held for a voice communication. The diagram in Figure 17.6 presents the sequence of SIP messages sent to establish the session, delegate the voice transport on the RTP protocol, and finally end the session.

Table 17.3 describes in detail the sequence of SIP messages exchanged between Felipe and Mafalda to establish the session, which will enable them to "speak," the delegation on the specific protocol that will transport (RTP) the voice and finally, the end of the session.

Table 17.1 SIP Request Messages

Name	*Meaning*
Specified in RFC 3261	
Invite	Establishes a session
Cancel	Cancels a pending session
Bye	Ends a session
Register	Maps a public URI with the current location of the user
Ack	Acknowledges the reception of an end response originated by INVITE
Options	Consults a server about its capabilities
Specified in RFC 3265, RFC 3856 Y RFC 3903	
Publish	Updates information on a server
Subscribe	Requests notifications about particular events
Notify	Notifies the User Agent about the occurrence of a particular event
Specified in RFC 3311	
Update	Modifies some session characteristics
Specified in RFC 3428	
Message	Transport PSNT telephone signaling
Specified in RFC 2976	
Info	Transport PSNT telephone signaling
Specified in RFC 3262	
Confirms the reception of a provisional response	

Components or elements of the SIP architecture:

User Agent (UA): A logical entity that can act as both a user agent client and user agent server.

User Agent Client (UAC): A user agent client is a logical entity that creates a new request, and then uses the client transaction state machinery to send it. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of

Table 17.2 Family of State Codes in the SIP Response Messages

Code	*Class*	*Functions*	*Examples*
1XX	Provisional informational	Request has been received and is being processed	100 trying 180 ringing 183 session in progress
2XX	OK	Successful, understood, accepted	20 OK 202 accepted
3XX	Redirect	Redirect or revise request	300 moved 305 use proxy
4XX	Client error	Error detected by the client, syntax error	401 unauthorized 404 not found 415 unsupported media type
5XX	Server error	Error detected by the server, cannot fulfill a valid request	500 not implemented 501 server timeout
6XX	Global network error	Request cannot be fulfilled by any server	600 busy everywhere 603 decline

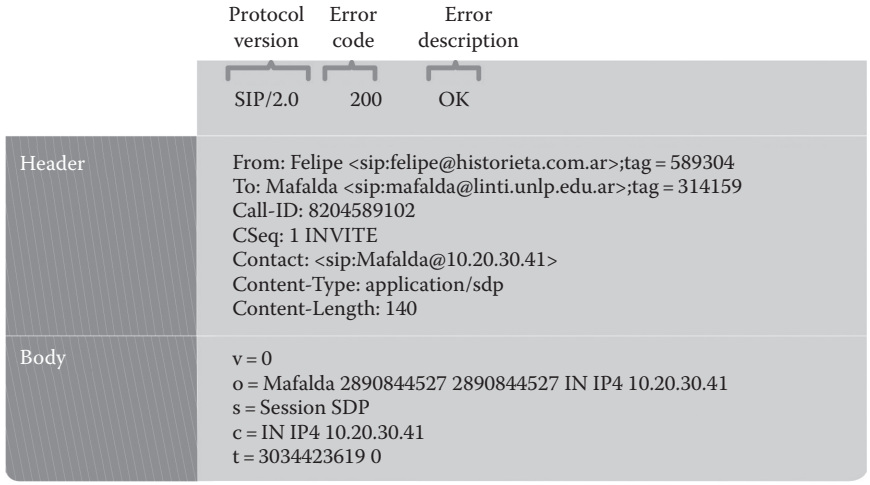


Figure 17.5 SIP OK message.

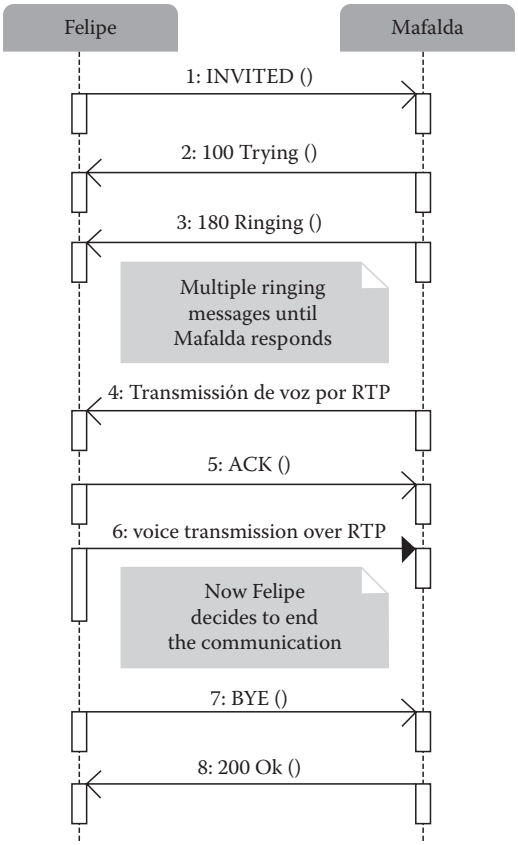


Figure 17.6 SIP dialogue between Felipe and Mafalda.

that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction.

User Agent Server (UAS): A user agent server is a logical entity that generates a response to a SIP request. The response accepts, rejects, or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the duration of that transaction. If it generates a request later, it assumes the role of a user agent client for the processing of that transaction.

The role of UAC and UAS, as well as proxy and redirect servers, are defined on a transaction-by-transaction basis. For example, the user agent initiating a call acts as a UAC when sending the initial INVITE request and as a UAS when receiving a BYE request from the callee.

Similarly, the same software can act as a proxy server for one request and as a redirect server for the next request.

Table 17.3 Sequence of SIP Messages between Felipe and Mafalda

INVITE sip:mafalda@linti.unlp.edu.arSIP/2.0 Via: SIP/2.0/UDP 10.20.30.40:5060 From: Felipe <sip:felipe@historieta.com.ar>;tag=589304 To: Mafalda <sip:mafalda@linti.unlp.edu.ar> Call-ID: 8204589102 CSeq: 1 INVITE Contact: <sip:Felipe@10.20.30.40> Content-Type: application/sdp Content-Length: 141 v = 0 0 = Felipe2890844526 2890844526 IN IP410.20.30.40 s = Session SDP c = IN IP410.20.30.40 t = 3034423619 0 M = audio 49170 RTP/AVP 0 a = rtpmap:0 PCMU/8000	Felipe sends an INVITE message to Mafalda telling her that he wishes to communicate with her. The body of the message contains the negotiation parameters for voice transmission in SDP format.
SIP/2.0 100 Trying From: Felipe <sip:felipe@historieta.com.ar>;tag=589304 To: Mafalda <sip:mafalda@linti.unlp.edu.ar> Call-ID: 8204589102 CSeq: 1 INVITE Content-Length: 0	Mafalda's device responds automatically with a Trying message to signal that the INVITE has been received.
SIP/2.0180 Ringing From: Felipe <sip:felipe@historieta.com.ar>;tag=589304 To: Mafalda <sip:mafalda@linti.unlp.edu.ar>;tag=314159 Call-ID: 8204589102 CSeq: 1 INVITE Content Length: 0	While Mafalda's device rings, it automatically sends ringing messages to inform of this situation.

(*continued*)

Table 17.3 (continued) Sequence of SIP Messages between Felipe and Mafalda

SIP/2.0 200 OK From: Felipe <sip:felipe@historieta.com.ar>;tag=589304 To: Mafalda <sip:mafalda@linti.unlp.edu.ar>;tag=314159 Call-ID: 8204589102 CSeq: 1 INVITE Contact: <sip:Mafalda@10:20.30.41> Content-Type: application/sdp Content-Length: 140 v = 0 o = Mafalda 2890844527 2890844527 IN IP410.20.30.41 s = Session SDP c = IN IP410.20.30.41 t = 30344236190 m = audio 3456 RTP/AVP 0 a = rtpmap:0 PCMU/8000	Mafalda takes Felipe's call sending the OK response message. The body of the message contains the agreed parameters for voice transmission in SDP format.
ACK sip:mafalda@linti.unlp.edu.ar SIP/2.0 Via: SIP/2.0/UDP 10.20.30.41:5060 Route: <sip:Mafalda@10.20.30.41> From: Felipe <sip:felipe@historieta.com.ar>;tag=589304 To: Mafalda <sip:mafalda@linti.unlp.edu.ar>;tag=314159 Call-ID: 8204589102 CSeq: 1 ACK Content-Length: 0	Felipe confirms with an ACK message. This is the only request message that cannot be replied and is only sent for INVITE.
Now Mafalda and Felipe are talking through a voice channel established with the SDP parameters which were agreed upon by means of the OK method. RTP packets of audio data are going in both directions over ports 49170 and 3456 using PCMU/8000 encoding.	

Table 17.3 (continued) Sequence of SIP Messages between Felipe and Mafalda

BYE sip:mafalda@linti.unlp.edu.ar SIP/2.0 Via: SIP/2.0/UDP 10.20.30.41:5060 To: Mafalda <sip:mafalda@linti.unlp.edu.ar>;tag=314159 From: Felipe <sip:felipe@historieta.com.ar>;tag=589304 Call-ID: 8204589102 CSeq: 1 BYE Content-Length: 0	To end the communication, any of the users, either Mafalda or Felipe, can send a BYE message to finalize the SIP session. In this case, Felipe decides to end the session.
SIP/2.0 200 OK To: Mafalda <sip:mafalda@linti.unlp.edu.ar>;tag=314159 From: Felipe <sip:felipe@historieta.com.ar>;tag=589304 Call-ID: 8204589102 CSeq: 1 BYE Content-Length: 0	Mafalda receives the BYE request and responds with an OK message, terminating the communication.

The most relevant functions of each layer of the IMS network are detailed below.

17.3.2 Application and Service Layer

In this layer, the services offered by the IMS network are available, whether provided in a standard way or as the new services that arise from the adoption of this new converging network. Typically, all the applications and services over the IMS network are running on SIP application services (SIP AS) located in the *Application* sub-layer. Its important to highlight the flexibility of the IMS architecture for the deployment of new services through third-party application servers, available as well through a peer-to-peer architecture based on SIP that do not require a central server. The Open Mobile Alliance (OMA) comprised by the main mobile operators and equipment manufacturers promotes the use of interoperable services among different devices, geographic locations, service providers, operators, and networks, named by OMA as *IMS enablers*. They provide specific Internet services with rich capabilities related to communities, such as presence information, group management, location, instant messaging (IM), push-to-talk over cellular, and IP conferencing. The IMS enablers collaborate on the development of multimedia applications from the *Services* sub-layer.

One of the key features of the IMS architecture is its flexibility to incorporate new services to end users. It is possible to use network resources (such as caller ID, user location service) and provide reliable services with feature-rich Web 2.0-type, among other things for entertainment and games. Currently, these collaboration capabilities and real-time communications provided by the IMS enablers are being incorporated into Internet services and applications. The generation of this new ecosystem of new applications is the great challenge of IMS. Multimedia push, real-time video sharing, real-time peer-to-peer multimedia streaming service, interactive gaming, and videoconferencing can be considered in the category of services.

17.3.3 Session Control Layer

This layer basically implements the control session. This aspect covers from the user registration, the SIP session routing, to the maintenance and management of the user data and policies that ensure service quality. The SIP session routing could consist on requests from a device, toward specific services hosted in an AS or toward another user in the network (peer-to-peer) of the same provider or doing roaming from another network or toward a predefined component which meets calls in eventual situations.

The core of the IMS architecture is the CSCF (call session control function) and its main functions include registration, session establishment and SIP routing strategies, which it carries out through the following components: P-CSCF (proxy-call session control function), I-CSCF (interrogating-call session control function), and S-CSCF (serving-call session control function) [R2]. In turn, the IMS architecture specifies a main database server called HSS (home subscriber server) which stores information of all the subscribers and data related to the services. P-CSCF is the first contact point of the end user with the IMS network; this implies that all SIP signaling traffic between the user device and the network passes through the P-CSCF.

The main function of I-CSCF is the recovery of the S-CSCF or the AS name that will attend the SIP request. It is carried out using the subscriber's profile information stored in the HSS. Another function of I-CSCF is to provide concealment of the topology among networks from different operators. S-CSCF is responsible for the registration, the SIP routing decisions, and the maintenance of the session states.

17.3.4 Transport and Access Layer

The most important goal of IMS is the convergence between fixed and mobile networks by creating a new paradigm in telecommunications services, where the system focuses on the user as opposed to the current paradigm focused on devices [R2]. This convergence was designed to give end users new communication

experiences provided across multiple geographic locations, devices, access technologies and services. The integration of fixed and mobile world presents the user the best of each, offering the convenience and availability of mobile services, and the reliability and quality of fixed [R2].

As shown in Figure 17.3, one of the most prominent features of IMS is the capacity to separate services from transport technology; thus a 3G mobile telephone connects to IMS network using protocols from the IP and SIP family in the same way that a PC does through the DSL. More significantly, in a mobile environment in which the user is able to move geographically, the IMS independent access not only enables the user to roam between different providers but the device could allow the user to change the connection method between different access technologies, making better use of available types of connections. For example, a telephone with Wi-Fi technology could change in a transparent way between the 3G and Wi-Fi access, and the users could also change the device, i.e., between a cell phone and the PC, maintaining the same session and user [R4]. In this layer, we can find the traditional Internet protocols and devices.

With a look into the OSI model, we have in the physical and data link layers the access technologies DSL, 3G, Cable Modem, WiFi, Ethernet, etc., in the network layer the IP protocol, essential for the interconnection of all these underlying technologies, and in the transport layer the protocols TCP, UDP, RTP/RCTP, etc.

The core of the IMS network is based on the same pillars as the Internet, as illustrated in Figure 17.3. IP is the core of this converging network architecture. In turn, the IMS design was conceived as well taking into account the service security and quality as essential elements. Security in IMS networks is applied in three different scenarios: in the first contact point of the user with the IMS network, which is between the user device and the IMS network, among the different devices of the diverse layers of the operator's core network and among the core networks of the different operators for the case of the use of *roaming*. Originally, IMS security support was provided by the IPSec protocol (RFC 2401, RFC 2406, RFC 2407, and RFC 2409) for access from both the user's device and for routing between different networks, then the TLS protocol (Transport Layer Security—RFC 2246) was added also for the same scenarios. Moreover, the protocol annexed AKA (Authentication and Key Agreement) for user authentication when accessing the network.

The service quality (QoS) is a key component of IMS; for a particular session, it could be determined by multiple factors, among them, the maximum bandwidth assigned to a user based on his subscription or the current state of the network. IMS supports multiple models of point-to-point quality service, the user's devices can use specific protocols of a link layer for resource reservation, i.e., PDP Context Activation for parameter reservation of QoS in mobile networks, RSVP (Resource ReSerVation Protocol, RFC 2205) or DiffServ (Differentiated Services, RFC 2475 and RFC 3260).

17.4 The Challenge Proposed by IMS for New Applications

The advantage of having a converging network of a higher bandwidth and service quality support enables us to build new multimedia applications and services which enhance user experience.

The possibility of using multimedia services in a simultaneous way encourages the construction of content rich applications, interactive and, in general terms, more suited to people's new communication behaviors that only consume what they are interested in (iTunes in an example), that register themselves in virtual communities, etc. The IMS network user can manage their own contents, share them with other users, such as "live video." Thus, IMS facilitates "user–user" and "user–content" communication.

Cell phones bring more and more benefits, their screens are more accurate and larger, they have photograph and video cameras incorporated, they reproduce music, and have GPS navigators. They are mobile devices with the capacity of being always on, representing a challenge for new applications that stop being isolated entities that exchange information through the user interface. The new generation of applications will be peer-to-peer, enabling to share music, games, boards, live video, etc.

17.5 Java Support for IMS and SIP

The neutrality of the Java platform, the ability to be available "everywhere" from mobile devices with limited hardware resources including servers to desktop PCs, laptops, etc., and the standardization of APIs to access the incorporated functionalities and build new ones without depending on manufacturer's particular implementations, positions Java as an ideal and convenient development platform for building new converging applications through IMS. Another important Java feature is its availability in different IDEs (integrated development environment) of the free software community, like the Eclipse and NetBeans initiatives.

JCP (Java Community Process) has various activities focused on the definition of multiple JSRs (Java Specification Request) that provide support for the development of applications based on SIP in the different Java technology *flavors*.

In turn, the JAIN (Java APIs for Integrated Networks) initiative of JCP defines APIs (application programming interface) that enable the use of JAVA technologies to develop telecommunication services in converging networks [L21].

The JAIN initiative extends Java technology to provide service portability (write once, run anywhere), network independence (any network), and open development (by anyone) promoting a chain of open value ranging from the network equipment, computer and device manufacturers to the outsource service providers, having impact in the technological structure and telecommunication company businesses.

The goal of this initiative is to change from closed and proprietary systems to open environments, having an influence in the same way JEE has done in the IT industry. This way, the communication operators can extend their portfolio services by getting faster, simpler, and less expensive applications [L22].

Within the JCP, multiple APIs JAVA evolve in relation to SIP and IMS [L23–L30]; they are described in Table 17.4:

Java is the standard platform for both server-side and client-side IMS application development. Server-side applications require an SIP servlet container compatible with JSR 289 [L25] (or the earlier JSR 116 [L31]). Client applications are implemented using the JSR 281 [L29] and JSR 325 [L30] specifications.

The most significant initiative in relation to IMS client development is the ICP (IMS Client Platform) [R5], included in Ericsson SDS (Service Development Studio) [L32]. The ICP was proposed as a standard Java in 2005 under JSR 281, led by Ericsson and BenQ. In its final state, the JSR 281 standard was divided into two specifications: JSR 281 and JSR 325. The first one, called *IMS Service API*, approved as standard in July 2008, contains basic IMS features like logging and setting up audio and video sessions, as well as a generic framework to access IMS services. JSR 325, called *IMS Communication Enablers* and currently in progress, provides an interface to access specific IMS service enablers such as presence, group management, and instant messaging. This part was removed from the original version of JSR 281. The ICP, together with the related JSRs, simplifies the development of IMS client-side applications.

As for the development of IMS server-side applications, JCP developed the JSR 116 specifications called SIP Servlet API 1.0 and the JSR 289 SIP Servlet API 1.1. Both specifications are an abstraction of the SIP protocol based on the Java Servlet API. SIP Servlets are the Java components that run in a Servlet container within a server. The SIP container simplifies the creation of SIP applications managing the life cycle of the SIP servlets and providing support for the interaction between SIP servlets and SIP clients (User Agent) through SIP request and reply message exchange. A container can enqueue messages, manage states in the server, and transfer control to the components or the SIP servlets, which are responsible for addressing messages. As the SIP protocol is based on the popular HTTP Web protocol, Java SIP Servlets share a common nature with Java HTTP Servlets. Thus, the programming interface available in the SIP Servlet API will be familiar to developers of Web applications in Java, particularly for those who make use of the Servlet API.

In connection with the implementation of the JSR 289 and JSR 116, the SailFin [L33] project is the most important initiative within the free software community. SailFin is a java.net project contributed by Ericsson, which extends the JEE application server, GlassFish [L34], SIP Servlet technology, included in the Ericsson SDS (Service Development Studio) [L32]. GlassFish is an open-source server implemented by Sun Microsystems and, like SailFin, has dual license—GNU-GPL Version 2 and CDDL Version 1.0. There are other proprietary products with similar capabilities as

Table 17.4 JCP Java APIs for handling SIP and IMS

API	*JSR*	*Java Platform*	*Description*
JAIN SIP	JSR32	JSE	Low-level API for SIP. Requires extensive knowledge of the SPI protocol. Provides call control management.
JAIN SIP Lite	JSR 125	JSE adaptable a JME	Abstraction of the SIP protocol which does not require a comprehensive understanding of the protocol. Being a lightweight API, it adapts to devices with scarce computing and memory capabilities.
SIP Servletq	JSR 289	JEE	Abstraction of the SPI protocol based on Java servlets.
			The SIP servlets are run and managed by a servlet container that hides the complexity of the protocol.
SIP for JME	JSR 180	JME	SIP for mobile devices with limited resources.
JAIN SIMPLE Presence	JSR 164	JME y JSE	API that allows clients and SIMPLE/ SIP servers to exchange presence information between a client and a server SIMPLE/SIP.
JAIN SIMPLE IM	JSR 165	JME y JSE	API that allows instant messaging between clients SIMPLE/SIP. Generally applications use JSR 164 and JSR 165 Implementations together.
IMS services	JSR 281	JME	High-level abstraction of IMS technology and protocols to facilitate JME applications development.
IMS communication enablers	JSR 325	JME	High-level abstraction of IMS technology and protocols to facilitate application development JME. Provides access to the IMS enablers (presence, group management, Instant Messaging, etc.).

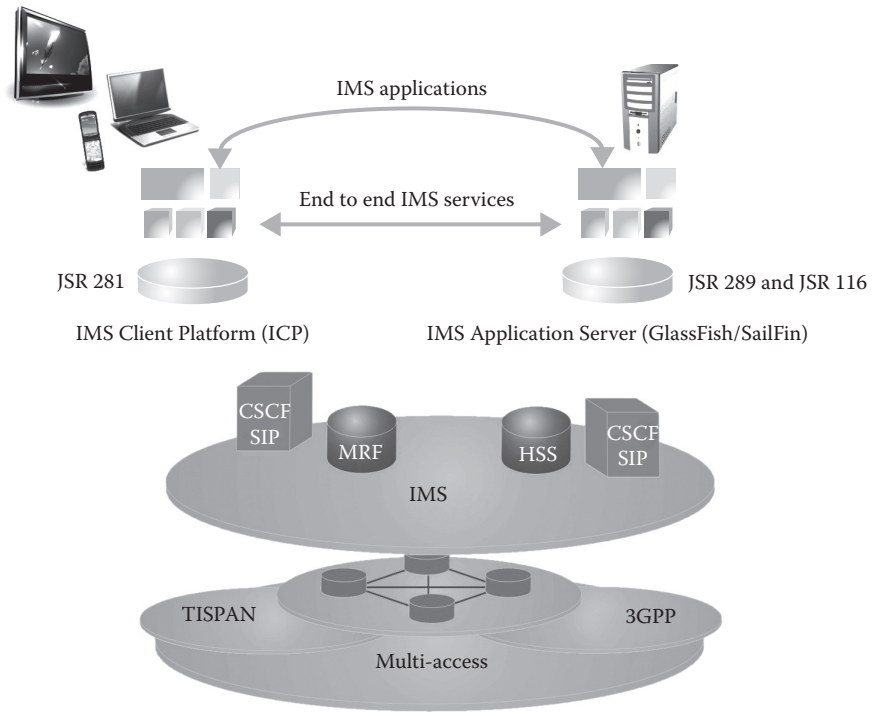


Figure 17.7 Implementations for Java IMS application development.

SailFin, among them, OCMS Oracle, WLSS BEA, and IBM Websphere; however, SailFin is the first one contributed by the free software community.

Figure 17.7 illustrates the two most relevant implementations described above that facilitate the development of client-side and server-side Java applications in IMS. They both provide high-level abstraction for the developer, hiding the particularities of IMS technology.

17.6 Java Development Environments for IMS Applications

Our school has over 10 years of experience training highly qualified students in development on Java technology through various graduate and postgraduate courses. The agreement of the Computer Science School with Ericsson-Argentina enabled us to leverage the strengths of both teams. Ericsson Company is a world leader in 2G and 3G mobile technologies and is now betting heavily on the convergence of multiservice networks to promote IMS technology and foster the

development of innovative applications that exploit the facilities provided by this technology, being one of the leaders in the standardization of Java APIs for application development on IMS-capable mobile devices. Ericsson has established the SDS [L32], which is an integrated development environment based on Eclipse open access [L35] that facilitates the construction of IMS services and applications. We believe that Ericsson SDS is the best IDE that accompanies the evolution of the different standardizations promoted by the JCP in relation to IMS; also, being based on Eclipse turns it into a popularly adopted tool by the Java developer community.

SDS is a comprehensive tool for development, testing, and deployment of IMS services and applications, both on the client side and the server side. It is possible to develop client-side applications using ICP, Ericsson's implementation of JSR 281, and on the other end, SDS integrates and supports the GlassFish/SailFin server that implements the JSR 289 for development of IMS services. To facilitate testing of mobile applications, SDS also includes a set of mobile device emulators.

Ericsson SDS simulates a complete IMS infrastructure, enabling the developers to test their software through an execution environment that emulates all the real IMS network components in an only PC.

17.7 IMS Java for Mobile 2.0

17.7.1 Java Initiatives for Mobile Applications

Currently, JME is the most accepted technology among cell phone manufacturers, wireless connection carriers, and mobile application developers. It has been adopted as a standard development and application execution platform in most cell phones. The JME platform has evolved in response to the growing capabilities of the devices, which have been introduced in the mobile world, based on the MIDP (Mobile Information Device Profile) specification. Today, the tendency in the software industry is toward standardizing the access to these new hardware capabilities in mobile devices and to the new services provided through the Internet. JCP supports this tendency with the elaboration of a specification built over MIDP called MSA (Mobile Service Architecture).

The MIDP 2 specification defined by JCP is the JSR 118 [L36] is designed to operate on CLDC (Connected, Limited Device Configuration) 1.0 (JSR 30 [L37]), 1.1 (JSR 139 [L38]) and subsequent versions. This last specification defines some basic APIs and a virtual machine for mobile devices with limited resources, such as the cell phone case. MIDP is the foundational piece of JME technology; it defines a platform to deploy in a dynamic and secure way for optimized applications, with graphic capacities and with connectivity possibilities through different technologies (Wi-Fi, Bluetooth, Infrared, USB, etc.).

Today, the mobile world is not limited to cell phones; it also includes devices such as netbooks, smartbooks, Amazon Kindle, PlayStation 3, etc. In response

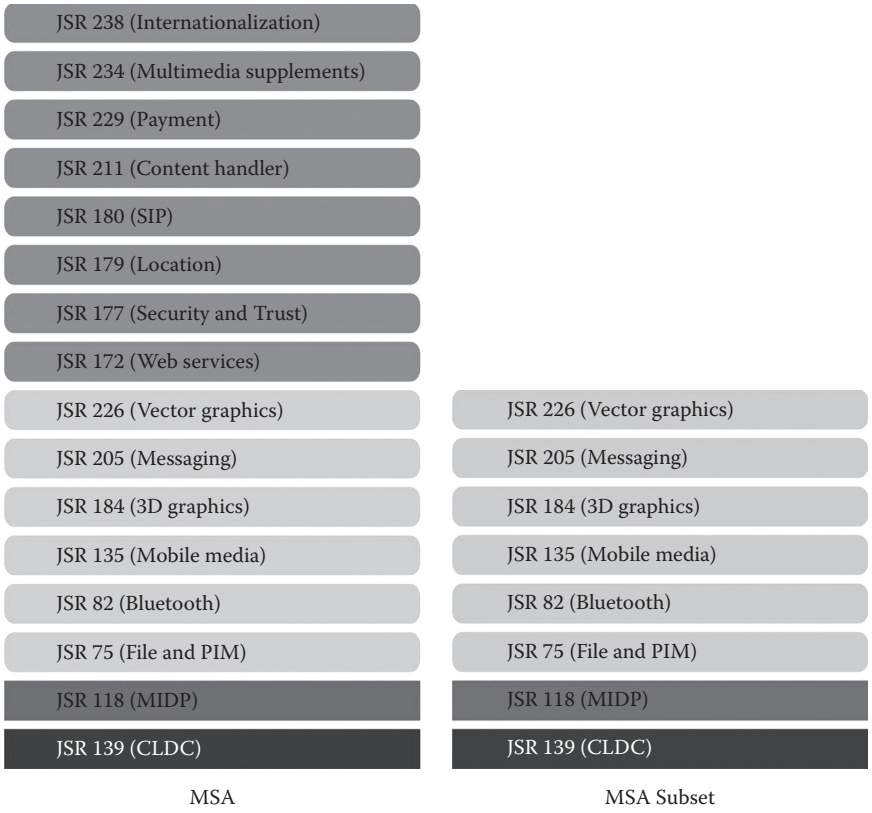


Figure 17.8 MSA 1.0 stack.

to this evolution, the Java ME platform was expanded to consider the new hardware and software capabilities of these devices, giving way to the JTWI (Java Technology for the Wireless Industry) platforms [L39] and later to MSA [L40], which are now a standardized environment for building mobile applications. These two new JME platforms include MIDP 2. Figure 17.8 shows the specification stack of the MSA and MSA Subset platform. The difference between both specifications is related with the option of including conditionally mandatory APIs that depend on the hardware capacity of the device, as is the case of the Location API (JSR 179) that depends on the existence of a GPS incorporated to the device.

17.7.2 Java Initiatives for IMS in Mobile Applications

As described, the MSA 1 specification [L40] for JME defines a set of standard functions for mobile devices. JCP continued to work on a new version of MSA called

MSA Advanced or MSA 2 (JSR 249) [L41], which improved the standard functionality by incorporating current technologies with a good future prospect, especially in the multimedia area such as the case of JSR 281 called IMS Services. Figure 17.9 shows the new APIs that were added to MSA to version 2 in relation with version 1. The incorporation of JSR 281 IMS Services API [L29] to MSA 2 standard shows that IMS technology will be available in the JME platform, promoting its adoption by mobile device manufacturing firms and with a favorable impact in the development of applications for these devices. Mobile application developers can take advantage of IMS multimedia capabilities such as QoS, single login to access multiple services, with the confidence that their applications will run on any device compatible with the MSA 2 platform.

17.7.3 Mobile Development JME: MIDlets

As already mentioned, MSA is based on MIDP and from its origins MIDP applications are called MIDlets. These applications are written with JME APIs that comprise the MSA standard and run on a mobile computing environment. MIDlets require a special execution environment given by a specific piece of software in the device called *Application Management System* (AMS) that controls the installation, execution, and life cycle of the MIDlet.

To build a MIDlet it is necessary to define a class that extends *javax.microedition.midlet.MIDlet* and overwrites at least the following three methods:

1. *startApp()*: It is invoked by the AMS to initiate the MIDlet or to resume the MIDlet execution in pause state.
2. *pauseApp()*: It is invoked by the AMS when certain events take place, for example, an incoming call.
3. *destroyApp()*: It is invoked by the AMS to release the resources allocated by the application.

A MIDlet has three different states that determine its functioning: Paused, Active, and Destroyed. These states correspond with the three methods earlier described called collectively as MIDlet life cycle methods. The AMS is responsible for controlling the MIDlet life cycle by invoking the life cycle methods. Figure 17.10 describes the MIDlet states and the methods invoked by the AMS that cause the passage between the different states.

The AMS decides when to invoke the life cycle methods: beginning the execution, the MIDlet passes to *Active* state, but what happens if it receives a call or a message while running? The AMS is in charge of changing the MIDlet state according to the external events that are produced. In this case, the AMS temporarily stops the MIDlet execution to attend the call or read the message, passing it to a *Paused* state.

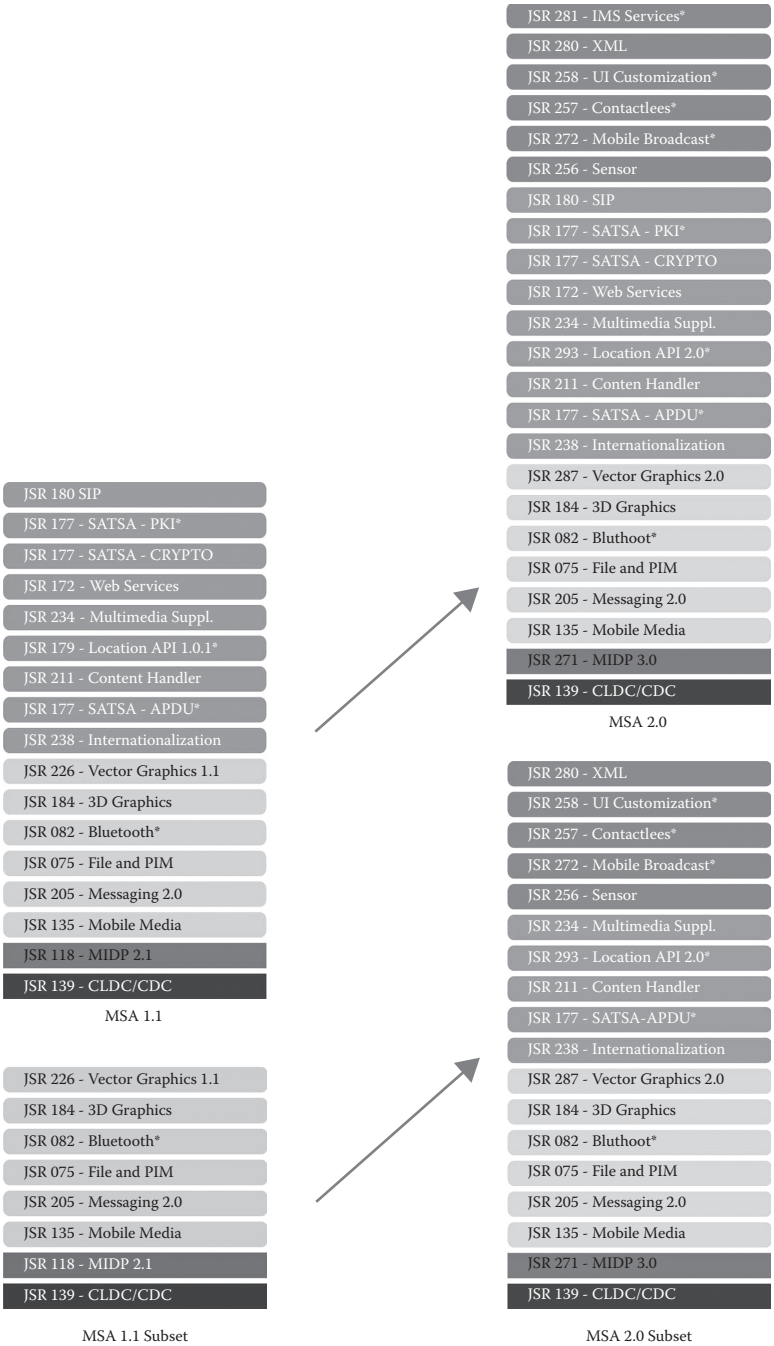


Figure 17.9 MSA 2.0 stack.

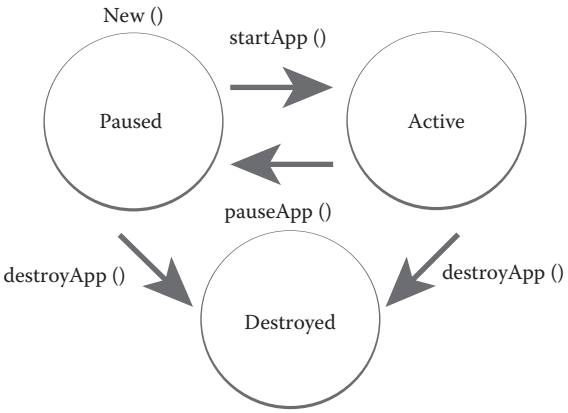


Figure 17.10 The MIDlet life cycle.

17.8 Development of a Simple Chatroom on IMS Using Ericsson SDS

17.8.1 Introduction

Below, we describe an example of a simple chat application, *IjcuChat*, taken from the *Ericsson Developer Connection** Web site. Despite continuous growth, interest, and expectations regarding the development of Java applications on IMS networks, we must consider that this is a software development technology that is not yet mature and there is no consolidated documentation on the issues to take into account for the development of applications. While the IMS architecture design promotes the abstraction of the network architecture for application development, it currently requires the programmer to be aware of some key elements and some basic configurations of the components that constitute it. Today, programming an IMS Java application using the SDS requires management of the specific APIs to access the IMS network and services and the assembly of a simulated execution context of an IMS network by setting its key elements (CSCF, HSS, DNS, etc.). That is why in this description we also consider the commonly requested configuration aspects.

17.8.2 General Architecture of the IjcuChat Application

IjcuChat is a JME (CLDC 1.1/MIDP2.0) client implemented through a Java Midlet that connects to the application server called *Twitty* by means of the IJCU platform. The basic functionality of this application is the registration of users, the

* http://www.ericsson.com/developer/sub/open/technologies/ims_poc/tools/sds_40

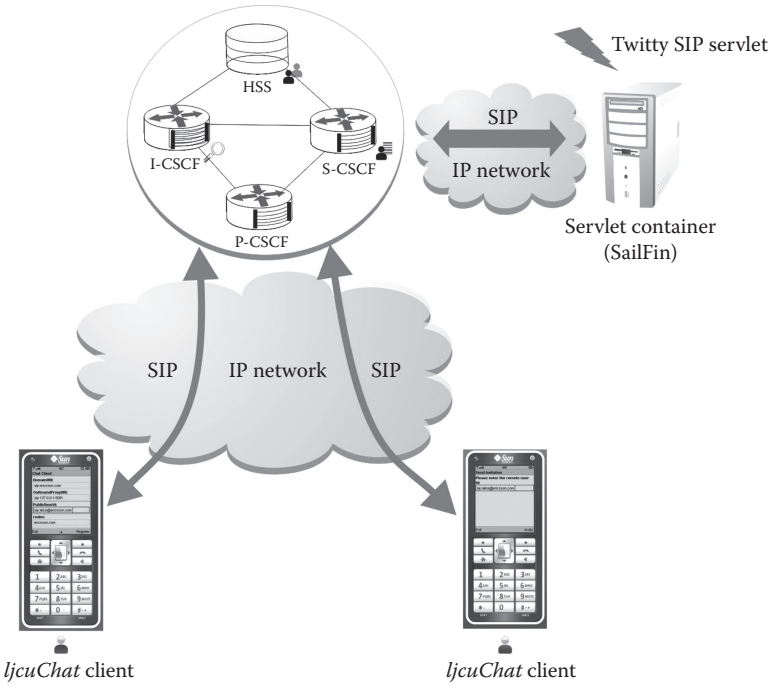


Figure 17.11 Conceptual *IjcuChat* architecture.

invitation from one user to another to start a conversation, and exchange messages between them. Figure 17.11 shows a conceptual scheme of the main components and the communication protocols used in the IMS *IjcuChat* Java application.

The user interface is a simple form that allows users to complete text fields required for the registration in the IMS network, the invitation and exchange of messages between two users. Figure 17.12 shows screenshots of the screens that enable the user to interact with *IjcuChat application; in this case*, *we will call the users of this application Bob and Alice*.

The Twitty server is a Java SIP server-side application implemented through a SIP Servlet (JSR 116) that runs in a SailFil application server. It provides a simple functionality that consists of sending the invitation message from one user to another to establish a conversation and forward the exchange of messages.

17.8.3 IjcuChat Functioning in a Simulated IMS Network in the Ericsson SDS

The DSD has a feature that facilitates the view of the exchange of SIP messages that pass through the CSCF through a flowchart. This feature is available in two ways, one is online and shows the exchange of SIP message during the application

(a) Registración de ALICE (b) Invitación de BOB a ALICE (c) Mensaje de BOB a ALICE (d) Mensaje de ALICE a BOB

Figure 17.12 *IjcuChat* screenshots.

execution, and the other is offline and shows the exchange recorded in a CSCF log file. This diagram is very educational to understand how the different actors in the IMS network interact; it is also a very useful tool when testing the application behavior in the different user interactions. Figure 17.13 shows an image capture of the SDS during the execution of the *IjcuChat* application and the *Twitty* server.

The flowchart in Figure 17.13 is described in detail in relation to the main actors of an IMS network and this way we can understand what happens in the *IjcuChat* application "backstage." Figure 17.13 shows the SIP methods used for the *IjcuChat application*.

SIP REGISTER: when the application starts, the first screen prompts the user registration in the IMS network (Figure 17.12), which results in the sending of an SIP REGISTER message to the CSCF component; this is the Alice and Bob case. Considering that the CSCF has three main components, P-CSCF, I-CSCF, and S-CSCF, in this interaction, the S-CSCF plays a main role. The functionality provided by the S-CSCF is to facilitate the establishment and ending of the SIP session with help from the HSS. The latter is the one that contains the subscriber's permanent data and the most relevant temporal data related with the users that are connected. Figures 17.14 and 17.15 show the screenshots from the *Provisioning* perspective of the SDS where it is possible to view the IMS network subscriber's permanent information and the temporary information, respectively; both aspects of the user management are stored in the HSS. In particular, the information presented is related with the registration of Alice and Bob in the IMS network. In conclusion, when the S-CSCF receives the request, SIP REGISTER delegates the

AQ2

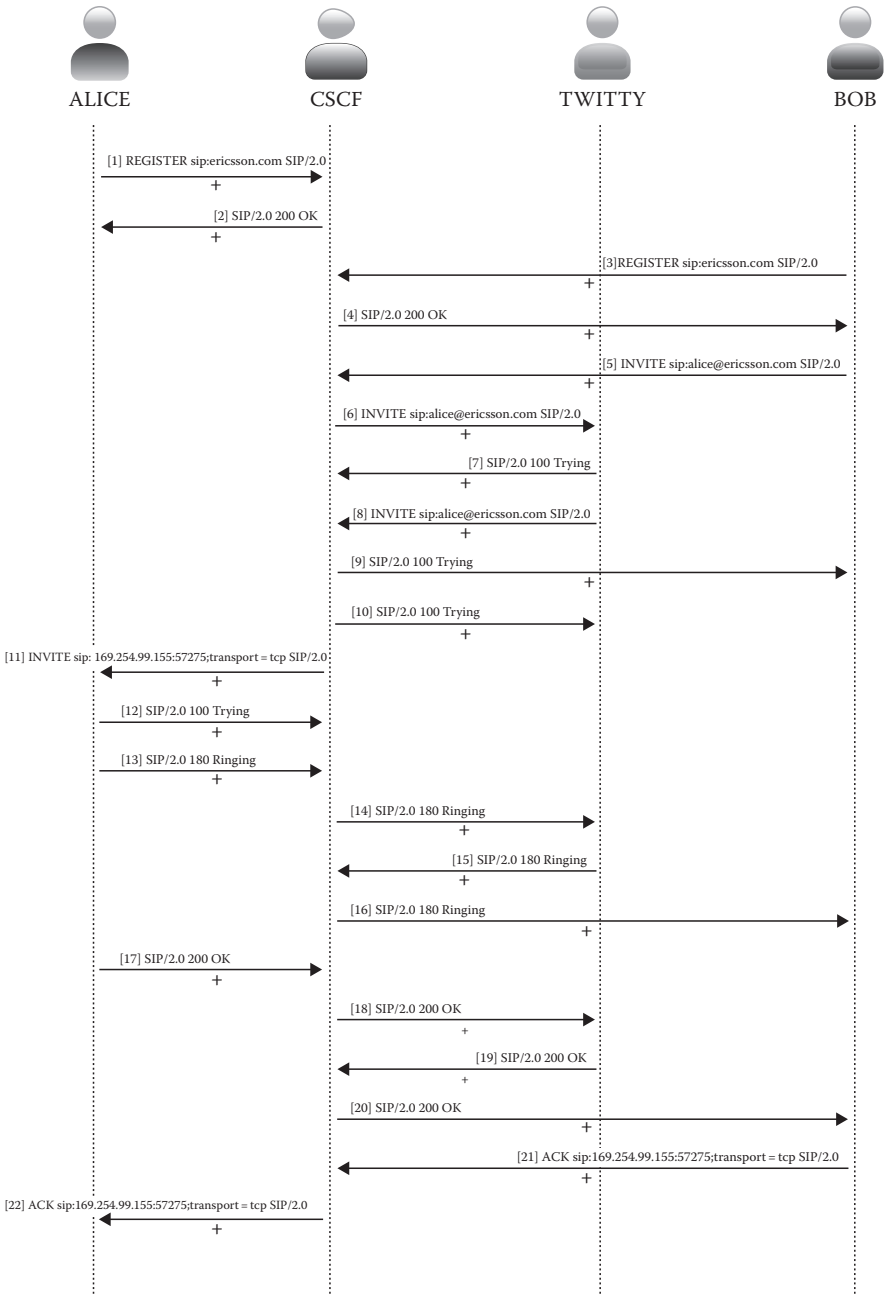


Figure 17.13 Flowchart of the *IjcuChat* application and the Twitty server.

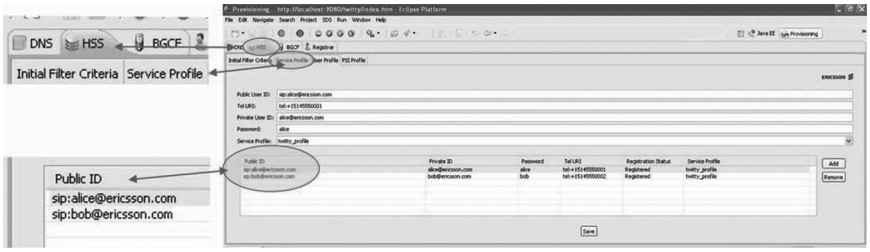


Figure 17.14 Provisioning perspective that shows the subscribers loaded in the HSS.

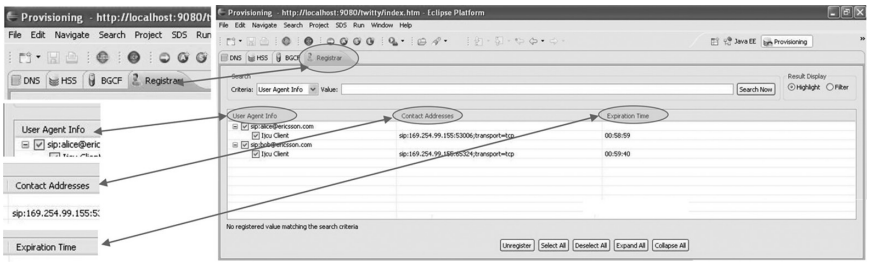


Figure 17.15 Provisioning perspective showing the online registered users in the HSS.

specific function of registration of the SIP session to later build and send the SIP answer to the user. In the flowchart in Figure 17.13, these interactions are summarized in the messages exchanged between the users Bob and Alice in the CSCF.

SIP INVITE and SIP MESSAGE: Once Bob and Alice are registered, Bob decides to invite Alice to a chat session. For this purpose, in the invitation screen (Figure 17.12), he indicates Alice's SIP address. With this user request, the application generates a message and sends a SIP INVITE message whose final receptor is Alice. In the IMS networks, this message in canalized through the CSCF that again plays a central role. The S-CSCF component is the most intensive processing node of the IMS nucleolus because it is responsible for determining what services will the users have available according to their profiles and which will be the application servers in charge of attending those requests. Figures 17.16 through 17.19 show the HSS configuration screen that relates the user's profiles with the services that will attend their requests. Particularly, each time a user interacts with an application that generates a SIP INVITE or SIP MESSAGE request message, they will be attended by the service identified as *twitty_profile*. Figures 17.18 and 17.19 show these configurations, called iFC (Initial Filter Criteria) in the IMS vocabulary. In our example,

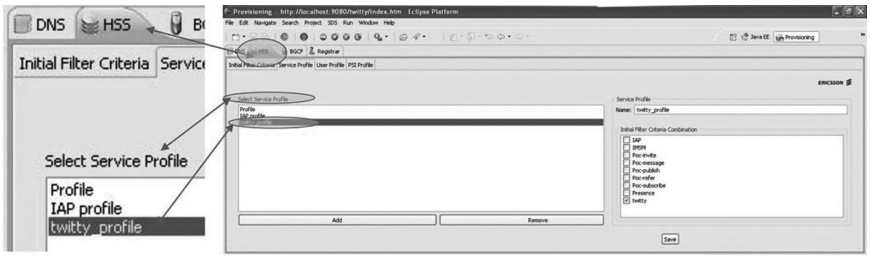


Figure 17.16 HSS configuration: Definition of the service profile twitty_profile.

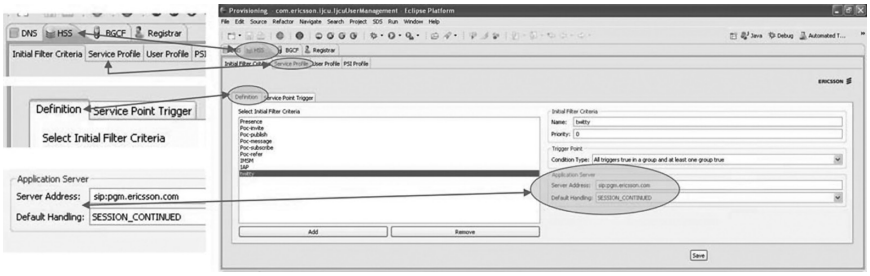


Figure 17.17 HSS configuration: Definition of the Initial Filter Criteria twitty.

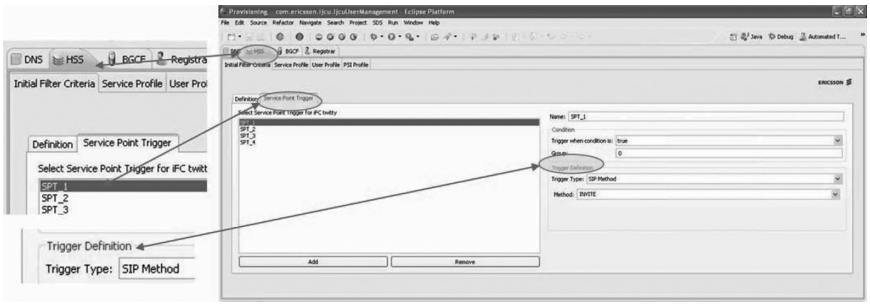


Figure 17.18 HSS configuration: iFC twitty INVITE criteria.

the established iFC is called *twitty*. In the flowchart in Figure 17.13, we can see that the SIP INVITE message sent from Bob to Alice is headed toward the CSCF who internally, as we explained previously, uses the S-CSCF component to determine through the iFC what server will attend the request; in this particular case, *Twitty*. Figure 17.17 shows the relationship between the iFC *twitty* and the server sip:pgm.ericsson.com where the *Twitty* service is hosted, and Figure 17.20 shows the resolution of the IP address in the DNS. As described before, *Twitty* provides a very AQ3

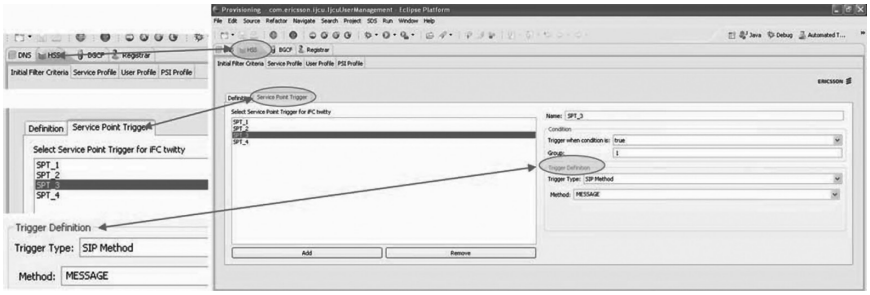


Figure 17.19 HSS configuration: iFC twitty MESSAGE criteria.

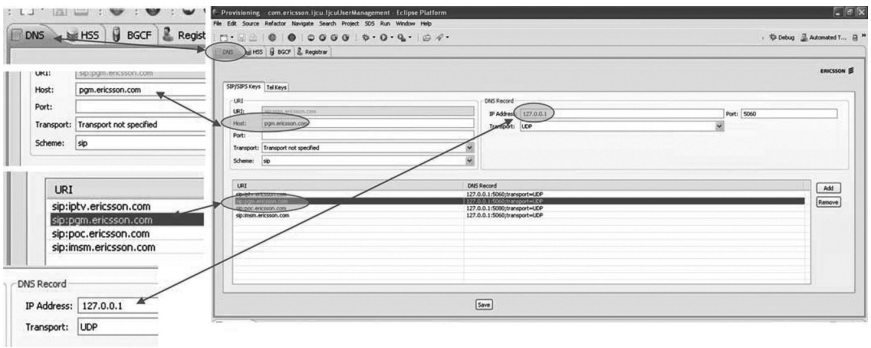


Figure 17.20 DNS configuration in the provisioning perspective.

simple functionality that will forward the SIP messages received addressed to Alice, once again through the CSCF. In relation to the SIP MESSAGE, the message flow between Bob and Alice is similar to the one described previously on the SIP INVITE message. The difference between them is that both Bob and Alice generate this type of message each time they chat using the *IjcuChat* client.

17.8.4 Analyzing the Use of JSR 281 in the IjcuChat

As mentioned before, *IjcuChat* is an application for mobile devices implemented by a MIDlet that uses the IMS network to communicate with the server and its set up in the chat session. Figures 17.21 through 17.28 show the most relevant code segments in which the API from JSR 281 is used to access the communication services provided by the IMS network. MIDP defines a mechanism called Push Registration that enables, among other things, to register MIDlets that that correspond to network events such as incoming connections. IMS applications must declare the capacities it can manage though Push Registration. After an incoming

```
public class ljcuChat extends MIDlet {
   private StringItem chatArea;
   private TextField domainURIField;
   private TextField outboundProxyURIField;
   private TextField publicUserIdField;
   private TextField realmField;
   private TextField privateUserIdField;
   private TextField mobilePhoneNumberField;
   private TextField passwordField;
   private Session session;
   private FramedMedia framedMedia;
   private String scheme = "imscore://";
   private String iari = "com.ericsson.client.ljcuChat";
   private String serviceId = "chat";
   private String aliceUserId = "sip:alice@ericsson.com";
   private String IARI_PREFIX = "urn%3Aurn-xxx%3A";
   private String transport = "tcp";
   private static String contentType = "text/plain";
   private CoreService coreService;
   public void startApp() throws MIDletStateChangeException {
      showRegistrationForm();
   }
```

Registration form fields in the IMS network

Application capability declaration fields

The application initiates with the display of the network registration form

Figure 17.21 Declaration of registration fields in the IMS network and of application capabilities.

```
private void showRegistrationForm(){
   Form registerForm = new Form("Register");
   Command registerCommand = new Command("Register", Command.OK, 1);
   Command exitCommand = new Command("Exit", Command.EXIT, 1);
   registerForm.addCommand(registerCommand);
   registerForm.addCommand(exitCommand);
   getDisplay().setCurrent(registerForm);
   domainURIField = new TextField("DomainURI:", "sip:ericsson.com", 40, TextField.ANY);
   outboundProxyURIField = new TextField("OutboundProxyURI:", "sip: 127.0.0.1:5081", 40, TextField.ANY);
   publicUserIdField = new TextField("PublicUserId:", aliceUserId, 40, TextField.ANY);
   realmField = new TextField("realm:", ericsson.com", 40, TextField.ANY);
   privateUserIdField = new TextField("PrivateUserId:", "alice@ericsson.com", 40, TextField.ANY);
   mobilePhoneNumberField = new TextField("MobilePhoneNumber:", "5143457900", 40, TextField.DECIMAL);
   passwordField = new TextField("Password", "alice", 10, TextField.PASSWORD);
   passwordField.setString("alice");
   registerForm.append(domainURIField);
   registerForm.append(outboundProxyURIField);
   registerForm.append(publicUserIdField);
   registerForm.append(realmField);
   registerForm.append(privateUserIdField);
   registerForm.append(mobilePhoneNumberField);
   registerForm.append(passwordField);
   registerForm.setTitle("Chat Client");
```

Creation and display of IMS network registration form

```
   registerForm.setCommandListener(new CommandListener() {
      public void commandAction(Command command, Displayable displayable) {
         if(command.getCommandType() == Command.OK) {
            RegisterThread registerThread = new RegisterThread();
            Thread thread = new Thread(registerThread);
            thread.start();
         }
         else if(command.getCommandType() == Command.EXIT) {
            exit();
         }
      }
   });
}
```

Initiate of the IMS network registration process

Figure 17.22 Registration form display.

```
protected class RegisterThread implements Runnable {
   public void run() {
      register();
   }

   public void register() {
      try {
      ljcuUserManagement ijcuUserManagement = ljcuUserManagement.getInstance();
      ijcuUserManagement.setDomainURI(domainURIField.getString());
      ijcuUserManagement.setOutboundProxyURI(outboundProxyURIField.getString());
      ijcuUserManagement.setPublicUserId(publicUserIdField.getString());
      ijcuUserManagement.setRealm(realmField.getString());
      ijcuUserManagement.setPrivateUserId(privateUserIdField.getString());
      ijcuUserManagement.setPassword(passwordField.getString());
      ijcuUserManagement.setTransport(transport);
      ijcuUserManagement.setPlatform(ljcuChat.this);

         Configuration myConfiguration = Configuration.getConfiguration();
         String [][] properties = new String [][] {{"Framed", contentTyper},
                                  {"CoreService", serviceId, IARI_PREFIX + iari, "",""}};
         myConfiguration.setRegistry(iari, ljcuChat.class.getName(), properties);

         ConnectionState connectionState = ConnectionState.getConnectionState();
         ConnectionState.setListener(new ConnectionStateAdapter());

         String url = scheme + iari + ";serviceId=" + serviceId;
         coreService = (CoreService) Connector.open(url);
         CoreServiceClientAdapter coreServiceAdapter = new CoreServiceClientAdapter();
         coreService.setListener(coreServiceAdapter);

         initializeSession();
      }
      catch (IllegalArgumentException e3) {
         alert("Can not register the phone: receive IllegalArgumentException");
      }
      catch (ConnectionNotFoundException e2) {
         alert("Can not register the phone: receive ConnectionNotFoundException");
      }
      catch (IOException e1) {
         alert("Can not register the phone: receive IOException");
      }
      catch (Exception e) {
         alert("Can not register the phone: receive Exception");
      }

   }
}
```

The application capabilities are declared using Push Registration

IMS network registration transaction triggers sending a SIP Register to CSCF

Inicio de formulario de invitación

Figure 17.23 Registration of the application capabilities and network registration.

call, the AMS is responsible for launching the MIDlet whose declared capabilities match the ones required by the incoming call.

The Push Registration can be carried out statically or dynamically. Static registration consists on declaring the capabilities through properties in the MIDlet's (JAD) description file. In our example, we use dynamic registration that consists of declaring the capabilities through an array of strings, which will be established as a parameter to the method *Configuration.setRegistry()*, as shown in Figure 17.23.

After declaring the application capabilities, the MIDlet registers the user in the IMS network invoking the *Connector.open()* method, and thus the application can initiate and receive invitations for chat and text message exchange. The *Connector.open()* method performs the registration through the message exchange of the SIP REGISTER transaction displayed graphically in Figure 17.13. This method may

```
private void initializeSession()
 {
  Form form = new Form("Initialize Session");
  form.setTitle("Send invitation");
  final TextField remoteUserId = new TextField("RemoteUserId", "sip:alice@ericsson.com", 40, TextField.ANY);
  remoteUserId.setLabel("Please enter the remote user Id:");
  form.append(remoteUserId);
  Command inviteCommand = new Command("Invite", Command.OK, 1);
  Command exitCommand = new Command("Exit", Command.EXIT, 1);
  form.addCommand(inviteCommand);
  form.addCommand(exitCommand);
```

Creation and display of the chat incitation form

```
    form.setCommandListener(new CommandListener()
    {
       public void commandAction(Command command, Displayable displayable)
      {
        if(command.getCommandType() == Command.OK)
        {
          StartSessionThread startSessionThread = new StartSessionThread(remoteUserId.getString());
          Thread thread = new Thread(startSessionThread);
          thread.start();
        }
        else if(command.getCommandType() == Command.EXIT)
        {
          exit();
        }
      }
    });

    getDisplay().setCurrent(form);
  }
```

Initiation of the chat session invitation process

Figure 17.24 Invitation form display.

```
private class StartSessionThread implements Runnable {
    private String remoteUserId;
    private StartSessionThread(String remoteUserId) {
      this.remoteUserId = remoteUserId;
    }
    public void run() {
      invite(remoteUserId);
    }
    private void invite(String remoteUserId) {
      if(!isUriValid(remoteUserId)){
        displayAlert("Please enter a valid URI", true);
      }
      try {
        session = coreService.createSession(publicUserIdField.getString(), remoteUserId);
        SessionAdapter sessionAdapter = new SessionAdapter();
        Session.setListener(sessionAdapter);

        framedMedia = (FrameMedia) session.createMedia("FrameMedia", Media.DIRECTION_SEND_RECEIVE);
        framedMedia.setAcceptedContentTypes(new String[]  {"text/plain"});

        FramedMediaAdapter outgoingFramedAdapter = new FramedMediaAdapter();
        framedMedia.setListener(outgoingFramedAdapter);

        session.start();
      }
      catch (IllegalStateException e) {
        displayAlert("Problem starting a session-IllegalStateException", true);
      }
      catch (ImsException e1) {
        displayAlert("Problem starting a session-ImsException", true);
      }
      catch (ServiceClosedException e2) {
        displayAlert("Problem starting a session-ServiceClosedException", true);
      }
    }
```

Creation of an object that represents a session for media exchange and their configuration

Chat session invitation transaction that triggers a SIP INVITE

Figure 17.25 Session setting and media configuration.

```
private class SessionAdapter implements SessionListener {
   public void sessionAlerting(Session arg0) {}
   public void sessionReferenceReceived(Session arg0, Reference arg1) {}
   public void sessionStartFailed(Session arg0) {
     displayAlert("Session can't be started", true);
     terminateSession();
   }
   public void sessionStarted(Session arg0) {
     startSession();
   }
   public void session terminated(Session arg0) {
      initializeSession();
   }
   public void sessionUpdateFailed(Session arg0) {}
   public void sessionUpdateReceived(Session arg0) {}
   public void sessionUpdated(Session arg0) {}
}
```

Established session alert that triggers the message exchage

Figure 17.26 Session events listener.

end successfully with the user registered in the network, or with an encapsulated failure as an exception. A code of nonsatisfactory answer as a result of the SIP REGISTER request is abstracted in an IOException.

Once the users are registered in the network, the MIDlet presents the form which allows the user to identify another user to start a chat session with. Figure 17.24 shows the code for the creation and display of this form.

Figure 17.25 shows the code related with the session establishment between two connection ends. The *CoreService.createSession()* method enables the creation of the session through the IMS network for the media exchange. In our example, we use the *FramedMedia* type, which represents a media transference connection in which the content is sent in packets and may be used for instant messaging as well as object serialization or file transference. Through the constant *Media. DIRECTION_SEND_RECEIVE*, it specifies that the media exchange will take place in both directions.

The object that represents the session is associated with a *listener* in charge of listening and processing the events it generates. In our example, the class *SessionAdapter* in Figure 17.26 implements this *listener*.

When the user at the other end of the session accepts the invitation, the implementation of the API JSR 281 generates an event that represents this situation and results in the invocation to the *sessionStarted()* method, which in our implementation invokes the MIDlet *startSession()* method that displays the form for entering and sending messages during the chat session, as shown in Figure 17.27.

Finally, Figure 17.28 shows how the message sending is done using the *FramedMedia* object created in the *StartSessionThread* class of Figure 17.25.

```
private void startSession() {
  Form form = new Form("Chatting");
  final TextField message = new TextField("Enter a message to sent:", "", 40, TextField.ANY);
  chatArea = new StringItem("\r\nMessages and status:\r\n","");
  chatArea.setText("");
  form.append(message);
  form.append(chatArea);
  Command byeCommand = new Command("End", Command.EXIT, 1);
  Command sendCommand = new Command("Send", Command.OK, 1);
  form.addCommand(sendCommand);
  form.addCommand(byeCommand);
```

Creation and display of the incoming messages form

```
  form.setCommandListener(new CommandListener() {
     public void commandAction(Command command, Displayable displayable) {
      if(command.getCommandType() == Command.OK) {
        SendThread sendThread = new SendThread(message.getString());
        Thread thread = new Thread(sendThread);
        thread.start();
      } else if(command.getCommandType() == Command.EXIT) {
        EndSessionThread endSessionThread = new EndSessionThread();
        Thread thread = new Thread(endSessionThread);
        thread.start();
       }
     }
  });
  Display.getDisplay(this).setCurrent(form);
}
```

Start of the message sending process to the other end of the session

Figure 17.27 Display of the message sending form.

```
private class SendThread implements Runnable {
  private String message;
  public SendThread(String message) {
     this.message = message;
  }
  public void run() {
     send(message);
  }
  private void send(String message) {
    try {
      framedMedia.sendBytes(message.getBytes(), "text/plain", null);
    }
     catch (Exception e) {
       displayAlert("Problem sending message!", true);
    }
  }
}
```

Message sending through FrameMedia object

Figure 17.28 Message sending.

17.9 Conclusions

IMS (IP Multimedia Subsystem) is an intelligent network architecture standardized by 3GPP, with multiple converging services such as mobile and fixed telephony, cable and satellite TV, and access to the Internet (usually provided by different networks), through an IP common data transmission infrastructure. Since it is based on standard interfaces and protocols (i.e., SIP for session management), IMS favors an "open" chain of values that includes network, computer and device manufacturers, as well as third-party service providers promoting its adoption by telecommunication companies.

The IMS layer design facilitates, on one hand, the availability of those services offered independently in the access network and, on the other hand, their integration, making the creation of new services and application more agile. Also, the growth of mobile devices for their processing capabilities, connectivity, and functionality and its broad adoption by users will promote an interesting market for the adoption of a network with the characteristics of IMS. IMS fills the gap between the two most successful communication paradigms, cellular and Internet technology.

The vision of IMS of "any service, any screen, anywhere," enables users to share contents "live" with multifunctional devices through multiple virtual systems, promoting the creation of enriched Web 2.0 applications.

References

[R1] R.J.M. Tejedor, Convergencia total en IMS: IP multimedia subsystem. *Revista Comunicaciones World*, 214, 50–53, 2006, ISSN 1139-0867, Versión digital: http://www.idg.es/comunicaciones/articulo.asp?id=178173 (última visita 20/02/09).

[R2] M. Poikselkä, G. Mayer, H. Khartabil, and A. Niemi, *The IMS*: *IP Multimedia Concepts and Services*, 2nd edn., John Wiley & Sons, Chichester, U.K., ISBN 0-470-01906-9, 2006.

AQ4 [R3] R. Copeland, *Converging NGN Wireline and Mobile 3G Networks with IMS*, CRC Press, Boca Raton, FL, ISBN 978-08493-9250-4, 2009.

[R4] O. Rashid, P. Coulton, and R. Edwarts, Implications of IMS and SIP on the evolution of mobile applications, *2006 IEEE Tenth International Symposium on Consumer Electronics*, *2006* (*ISCE'06*), Petersburg, Russia, ISBN 1-4244-0216-6, 2006.

AQ5 [R5] P. Kessler, IMS client platform, *Ericsson Review*, No. 2. Año 2007.

Web Sites

[L1] NAPSTER: http://free.napster.com/

[L2] KaZaA: http://www.kazaa.com/

[L3] eMule: http://www.emule-project.net/

[L4] ICQ: http://www.icq.com/

[L5] Skype: http://www.skype.com
[L6] MSN: http://www.msn.com
[L7] Twitter: http://twitter.com/
[L8] GPRS: General Packet Radio Service. http://www.3gpp.org/article/gprs-edge
[L9] Code Division Multiple Access. http://www.tiaonline.org/standards/technology/cdma2000
[L10] 3GPP2: 3rd Generation Partnership Project 2. http://www.3gpp2.org/public_html/specs/index.cfm
[L11] TISPAN: Telecoms & Internet converged Services & Protocols for Advanced Networks. http://www.etsi.net/tispan/
[L12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session initiation protocol. June 2002. http://www.ietf.org/rfc/rfc3261.txt
[L13] A. Niemi. RFC 3903: Session initiation protocol (SIP) extension for event state publication. October 2004. http://www.ietf.org/rfc/rfc3903.txt
AQ6 [L14] J. Rosenberg. RFC 3857: A watcher information event template-package for the session initiation protocol (SIP). August 2004. http://www.ietf.org/rfc/rfc3857.txt
[L15] A. B. Roach. RFC 3265: Session initiation protocol (SIP)-specific event notification. June 2002. http://www.ietf.org/rfc/rfc3265.txt
[L16] J. Rosenberg. RFC 3311: The session initiation protocol (SIP) UPDATE method. September 2002. http://www.ietf.org/rfc/rfc3311.txt
[L17] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle. RFC 3428: Session initiation protocol (SIP) extension for instant messaging. December 2002. http://rfc.dotsrc.org/rfc/rfc3428.html
[L18] S. Donovan. RFC 2976: The SIP INFO method. October 2000. http://www.ietf.org/rfc/rfc2976.txt
[L19] J. Rosenberg and H. Schulzrinne. RFC 3262: Reliability of provisional responses in the session initiation protocol (SIP). June 2002. http://www.ietf.org/rfc/rfc3262.txt
[L20] J. Rosenberg. RFC 3856: A presence event package for the session initiation protocol (SIP). August 2004. http://www.ietf.org/rfc/rfc3856.txt
[L21] Java Community Process. JSRs de la iniciativa JAIN. http://jcp.org/en/jsr/summary?id=jain
[L22] JAIN and Java in Communications. Sun White Paper. March 2004. http://java.sun.com/products/jain/reference/docs/Jain_and_Java_in_Communications-1_0.pdf
[L23] JSR 32: JAIN SIP API. http://jcp.org/en/jsr/detail?id=32
[L24] JSR 125: JAIN SIP Lite. http://jcp.org/en/jsr/detail?id=125
[L25] JSR 289: SIP Servlet v1.1. http://jcp.org/en/jsr/detail?id=289
[L26] JSR 180: SIP API for J2ME. http://jcp.org/en/jsr/detail?id=180
[L27] JSR 164: SIMPLE Presence. http://jcp.org/en/jsr/detail?id=164
[L28] JSR 165: SIMPLE Instant Messaging. http://jcp.org/en/jsr/detail?id=165
[L29] JSR 281: IMS Services API. http://jcp.org/en/jsr/detail?id=281
[L30] JSR 325: IMS Communication Enablers (ICE). http://jcp.org/en/jsr/detail?id=325
[L31] JSR 116: SIP Servlet API v1.0. http://jcp.org/en/jsr/detail?id=116
[L32] SDS (Service Development Studio) de Ericsson. http://www.ericsson.com/developer/sub/open/technologies/ims_poc/tools/sds_40
[L33] Proyecto SailFin. http://sailfin.dev.java.net
[L34] GlassFish. https://glassfish.dev.java.net
[L35] Eclipse. http://www.eclipse.org

[L36] JSR 118: Mobile Information Device Profile 2. http://jcp.org/en/jsr/detail?id=118
[L37] JSR 30: Connected, Limited Device Configuration. http://jcp.org/en/jsr/detail?id=30
[L38] JSR 139: Connected Limited Device Configuration 1.1. http://jcp.org/en/jsr/detail?id=139
[L39] JSR 185: JavaTM Technology for the Wireless Industry. http://jcp.org/en/jsr/detail?id=185
[L40] JSR 248: Mobile Service Architecture. http://jcp.org/en/jsr/detail?id=248
[L41] JSR 249: Mobile Service Architecture 2. http://jcp.org/en/jsr/detail?id=249

Author Queries

[AQ1] The sentence starting: "Examples of applications on IMS" seems incomplete. Please check.
[AQ2] Figures 17.19 and 17.20 have been renumbered to maintain sequential order of appearance. Please check.
[AQ3] Figure 17.15 has been changed to Figure 17.20 as per the caption. Please check if this is ok.
[AQ4] Please provide in-text citation for [R3].
[AQ5] Please provide page range in Ref. [R5], if appropriate.
[AQ6] Please provide in-text citation for [L14].