IMS Application Servers

Roles, Requirements, and Implementation Technologies

The IP multimedia subsystem (IMS) defines a generic architecture to support communication services over a Session Initiation Protocol (SIP) infrastructure. In the IMS architecture, application servers host and execute the IMS service logic. These servers can be SIP application servers, open services architecture (OSA) application servers, or a customized applications for mobile networks using enhanced logic (Camel) service environment. Some technologies used in telephony and voice-over-IP (VoIP) application servers are also applicable to IMS application servers, but such servers have some unique requirements that could limit the extent to which these technologies can meet them.

Hechmi Khlifi
Dialexia

Jean-Charles Grégoire
National Institute of Scientific Research, Canada

The Third-Generation Partnership Project (3GPP)-defined IP Multimedia Subsystem[1] is becoming the de facto standard for real-time multimedia communication services. IMS defines a generic architecture for offering communication services such as multimedia telephony, push to talk, and instant messaging over a Session Initiation Protocol (SIP)[2] infrastructure.

Among the most important components of the IMS architecture are the application servers, which host and execute the logic of the IMS services (for example, how services are invoked, what signaling and media actions are involved, and how services interact with each other). IMS defines three types of application servers: SIP application servers, open services architecture (OSA) application servers, and a customized applications for mobile networks using enhanced logic (Camel) service environment. This article discusses the first two server types.

Several technologies to build telephony and voice-over-IP (VoIP) application servers have been proposed. Some of these technologies are relevant to IMS application servers and can meet their unique requirements. We assume that the reader is familiar with the SIP protocol and the roles of its architectural elements, especially the SIP proxy.

Overview

IMS is a set of specifications that defines a complete framework for enabling the convergence of voice, video, and data communications over an IP-based infrastructure using SIP. The 3GPP

1089-7801/08/$25.00 © 2008 IEEE

originally designed IMS for mobile networks. Other standards bodies, including the European Telecommunications Standards Institute,[3] have adopted it for fixed networks. It has also become part of the International Telecommunication Union (ITU) Next-Generation Networking (NGN) vision.

IMS Benefits

The IMS architecture has generated a lot of interest from both wireline and wireless service providers. This interest stems from its service delivery framework as well as from the services themselves.

Integrated delivery of multimedia services. IMS, through the use of SIP, brings the Internet's power to the communication world. It provides a flexible way to build high added-value services on top of the common signaling infrastructure. Service providers can integrate voice, video, and data services and provide them on a single platform.

Modular architecture. Because IMS is highly modular, service providers can integrate different components or modules from different solution providers into the same system. This establishes vendor independence and optimizes investment.

Mobility and roaming. IMS provides access to a user's specific set of services, independently of their location and network-access serving operator.

Extra features. Finally, IMS incorporates integrated QoS, security, flexible charging, and lawful intercept mechanisms, making it a complete service platform for next-generation networks.

IMS Architecture

Figure 1 shows the NGN functional architecture. It has three main layers: transport, control, and service. IMS is at the architecture's core. It consists of several functions (such as serving call session-control function [S-CSCF], home subscriber server (HSS), and multimedia resource function control [MRFC]), interconnected through standardized interfaces. We give an overview of each layer in the following sections. An in-depth tutorial appears elsewhere.[4]

Transport layer. The transport layer is the network-access layer. It lets different IMS devices and user equipment connect to the IMS network. IMS devices connect to the IP network in the transport layer using various technologies, including fixed access (DSL, cable modems, Ethernet, and so on), mobile access (wideband code-division multiple access [W-CDMA], CDMA-2000, Global Packet Radio Service [GPRS], and so on) and wireless access (such as wireless local area network [WLAN] or WiMax). The transport layer also lets IMS devices place and receive calls to and from the public switched telephone network (PSTN) or other circuit-switched networks through the media gateway as Figure 1 shows.

The transport layer establishes the user equipment's IP connectivity. Once the user equipment has an IP address and can exchange SIP messages (either directly or through a gateway), it will be responsible for its own IMS interactions, independent of the underlying network-access technology. For example, K. Daniel Wong and Vijay K. Varma show how IMS can be used in Universal Mobile Telecommunications System (UMTS) networks.[5]

Control layer. All control-layer functions are specified as part of the IMS architecture. At the layer's core is the CSCF. Three SIP signaling servers handle this function: the *proxy CSCF* (P-CSCF), the *interrogating CSCF* (I-CSCF), and the S-CSCF mentioned previously.

The P-CSCF is a SIP proxy that is the first point of contact for the IMS terminal. IMS terminals discover their corresponding P-CSCF as part of their network connectivity procedure (that is, through the Dynamic Host Configuration Protocol [DHCP]). The P-CSCF sits on the path of all signaling messages of the IMS terminal and passes SIP registration to the correct home network (that is, the subscriber's administrative IMS domain) and SIP session messages to the correct S-CSCF after registration. The P-CSCF can also perform message compression or decompression and security enforcement.

The I-CSCF is a SIP proxy located at the administrative IMS domain's edge. Its IP address is published in the domain's DNS server, so remote servers (such as a P-CSCF in a visited domain or an S-CSCF in a foreign domain) can find it and use it as an entry point for all SIP transactions to this domain.

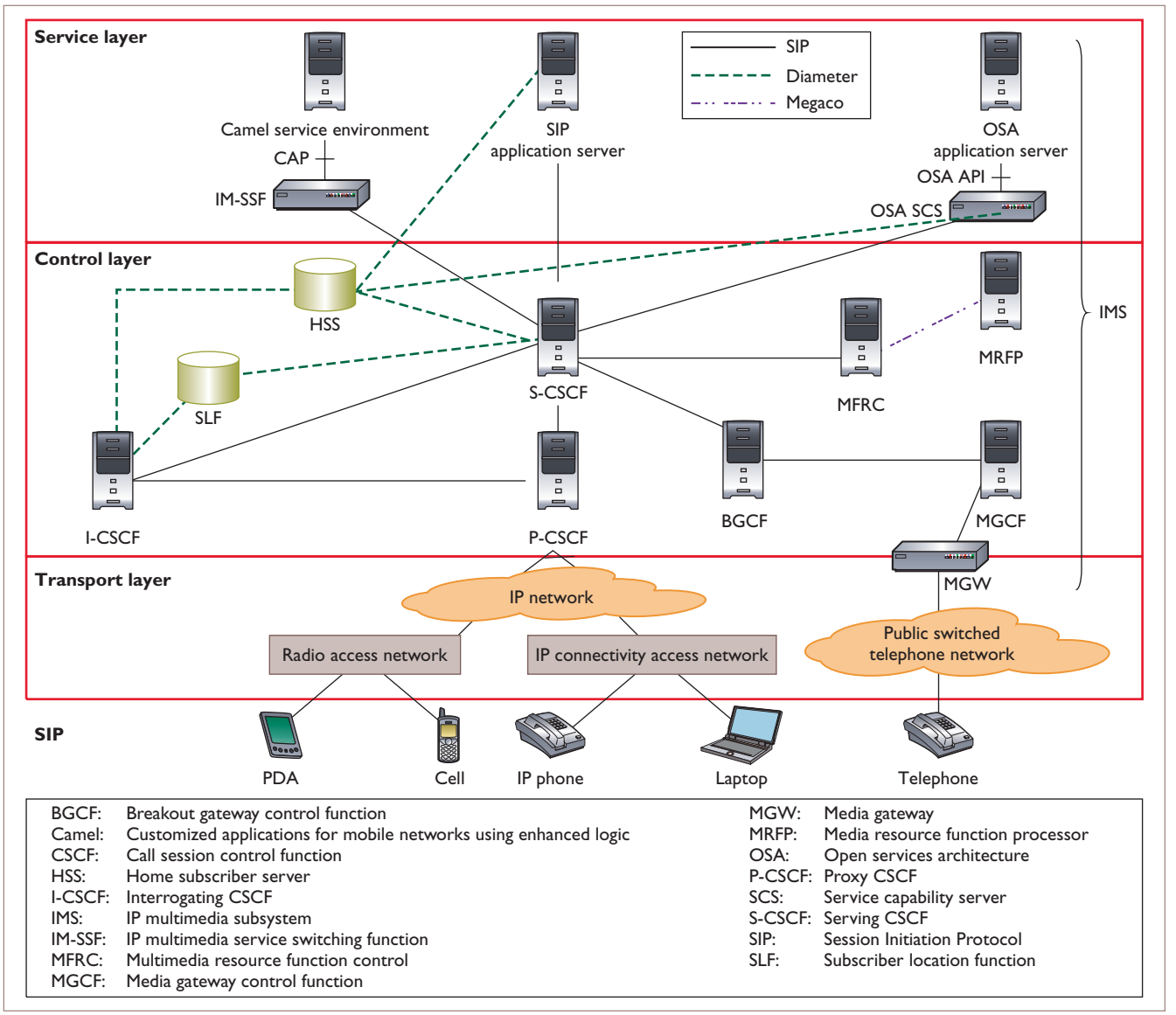

Figure 1. Overall Next-Generation Networking (NGN) functional architecture. The NGN architecture has three layers — transport, control, and service — with the IP multimedia subsystem (IMS) at the architecture's core.

The S-CSCF is the signaling plane's central node. It registers users and provides services to them. It routes SIP requests, provides billing information to mediation systems, maintains session timers, and interrogates the HSS to retrieve authorization, service-triggering information, and user profiles.

The HSS is the master user database. It supports the IMS network entities that handle the calls or sessions. It contains subscription-related information (that is, user profiles), authenticates and authorizes users, and can provide information about users' locations. An IMS domain needs a subscriber location function (SLF) when it uses multiple HSSs. Both the HSS and SLF implement the Diameter[6] protocol – an authentication, authorization, and accounting (AAA) protocol that the other elements of the IMS network can use to upload and download to and from the HSS/SLF.

The *media resource function* provides media functions in the IMS architecture (such as playing announcements or recording voicemails). The IMS standard decomposes the function into two elements.

The *media resource function controller* interprets information from the S-CSCF (for example, a session for media processing such as playing announcements and mixing streams). The S-CSCF uses SIP to communicate with the

MRFC, which uses Megaco/H.248,[7] a master–slave media-control protocol, to control the *media resource function processor* (MRFP).

The *breakout gateway control function* (BGCF) determines the next hop for routing SIP messages that can't be routed by the S-CSCF. It selects a media gateway control function (MGCF) that will route the call to the PSTN through a media gateway.

Service layer. The transport and control layers provide an integrated and standardized network platform to let service providers offer various multimedia services in the service layer. Application servers host and execute the services and provide the interface with the control layer using the SIP protocol. As we stated previously, IMS defines three types of application servers: the SIP application server, OSA application server, and the Camel service environment.

The Camel service environment is a set of mechanisms that let mobile network operators provide subscribers with operator-specific services, even when they're roaming outside their home network service environments. In the IMS architecture, the IP multimedia service switching function interface translates requests between SIP and the Camel application part. The implementation of the Camel service environment is outside this article's scope.

IMS Application Servers

An IMS application provides a specific service to the end user. IMS end-user services include multiparty gaming, videoconferencing, messaging, community services, presence, and content sharing.

Depending on its implementation, an IMS application server can host one or many IMS applications. In both cases, the application server handles and interprets the SIP messages forwarded by the S-CSCF (either directly or through the OSA service capability server [SCS]) and translates end-user service logic into sequences of SIP messages, which it sends to the parties involved, again through the S-CSCF.

The IMS architecture lets an IMS service provider deploy multiple application servers in the same domain. Different application servers can be deployed for different application types (for example, telephony or presence application servers) or different groups of users. The S-CSCF decides whether it should forward an incoming initial SIP request to a given application server. The decision it makes is based on filter information received from the HSS. The HSS stores and conveys this filter information on a per-application-server basis for each user.

When the HSS transfers the name and address of more than one application server, the S-CSCF must contact each application server in the order provided. The S-CSCF uses the first application server's response as input to the second application server.

The application server uses filter rules to decide which of the many services deployed on the server should handle the session. During the service logic's execution, the application server can communicate with the HSS to get additional information about a subscriber or to learn about changes in the subscriber's profile.

The application server (SIP application server or the OSA SCS) uses the Diameter protocol to communicate with the HSS. Diameter transports user-profile-related data, but can also transport transparent data — that is, data for which the exact representation of information isn't understood by the HSS or the protocol (for example, service-related data or user-related information).

An Example IMS Application

To illustrate the interactions between key elements of the IMS architecture and the IMS application servers, we implemented a small IMS audioconference application.

To join a conference, a participant calls the service identified by its uniform resource identifier (URI) — for example, `sip:ConfServiveURI@emt.inrs.ca`. When the service receives the call, it plays a greeting and asks the caller to enter the conference code. If the code is valid, it asks the caller to enter his or her PIN. If the PIN is also valid, the service lets the caller join the conference; otherwise, it asks the caller to retry.

SIP Application Servers

SIP application servers can act as redirect servers, proxy servers, originating user agents, terminating user agents, or back-to-back user agents. They have SIP signaling capabilities and are directly involved in the call's signaling flow. They receive SIP messages from the S-CSCF and parse them. Similarly, they generate SIP messages and send them to the S-CSCF.

The application server translates an IMS

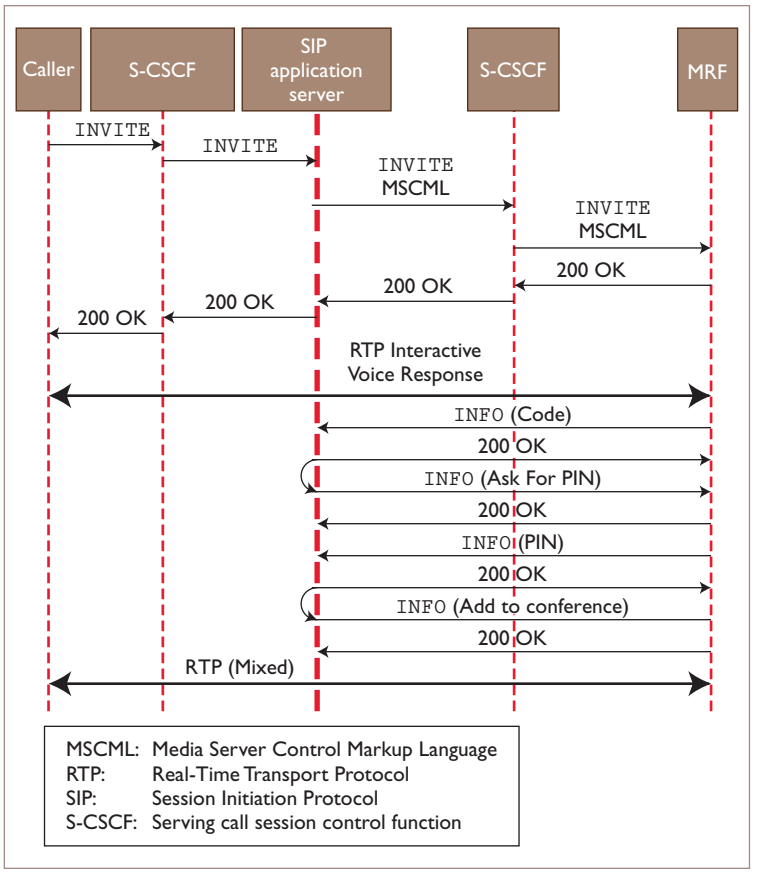

Figure 2. The audioconference service's call flow, managed by a SIP application server. This server invokes the multimedia resource function control's capabilities using SIP.

service's execution in the SIP application server into a sequence of SIP procedures, such as sending invite requests and reacting to SIP responses. For services requiring media interaction, the SIP application server invokes the multimedia resource function control (MRFC) capabilities. The IMS specifications don't explain exactly how to perform this invocation, but several proposals exist for letting the SIP application server use SIP to control the MRFC (for example, basic network media services with SIP,[8] Media Server Control Markup Language [MSCML],[9] Media Objects Markup Language, and Media Sessions Markup Language).

Figure 2 shows the audioconference service's call flow when managed by a SIP application server. We assume that the MRFC and the MRFP are colocated in the same entity (MRF in the figure). We also assume that media control messages are transported in the body of the SIP `invite` and `info` requests. (An `info` request is a SIP request used to carry session-related control information generated during a session.) For simplicity, the figure doesn't show some provisional SIP messages, such as 100 Trying and 180 Ringing.

When the SIP application server receives the first `invite` from the caller, it creates a new SIP session with the MRF. The body of the `invite` asks the MRF to play a greeting to the caller and ask the caller to enter the conference code. The MRF forwards the conference code received from the user to the application server in an `info` SIP request. If the code is valid, the SIP application server sends an `info` request back to the MRF, asking it to ask the caller to enter his or her PIN. The SIP application server and the MRF thus continue to exchange SIP messages until the authentication process ends, and the caller's user agent starts receiving the media stream from other participants' input. As Figure 2 shows, all SIP messages transit through the S-CSCF.

OSA Application Servers

OSA application servers can provide the same services as the SIP application server but have no signaling capabilities and aren't directly involved in the SIP calls' signaling flow. They communicate with the S-CSCF through the OSA SCS, which maps SIP messages into invocations of the OSA API (also called Parlay) and back. From an S-CSCF perspective, the SIP application server and the OSA SCS exhibit the same behavior.

The OSA application server also has access to the HSS data, but only through the SCS. The SCS implements the Diameter protocol, so it can read and update data records based on the OSA application server's requests.

The OSA application server mainly implements external services that could be located in a visited network or a third-party platform. Figure 3 shows the call flow of the audioconference service managed by an OSA application server. The interactions between the OSA SCS and the MRF are similar to those of the SIP application server and MRF. However, the OSA application server tells the OSA SCS which action to perform (for example, using `sendInfoReq()method` calls), and the OSA SCS notifies the OSA application server of the events reported by the MRF (for example, using `sendInfoRes()method` calls).

IMS Application Server Implementation Technologies

The IMS specifications don't define the application servers' internal architecture. To un-

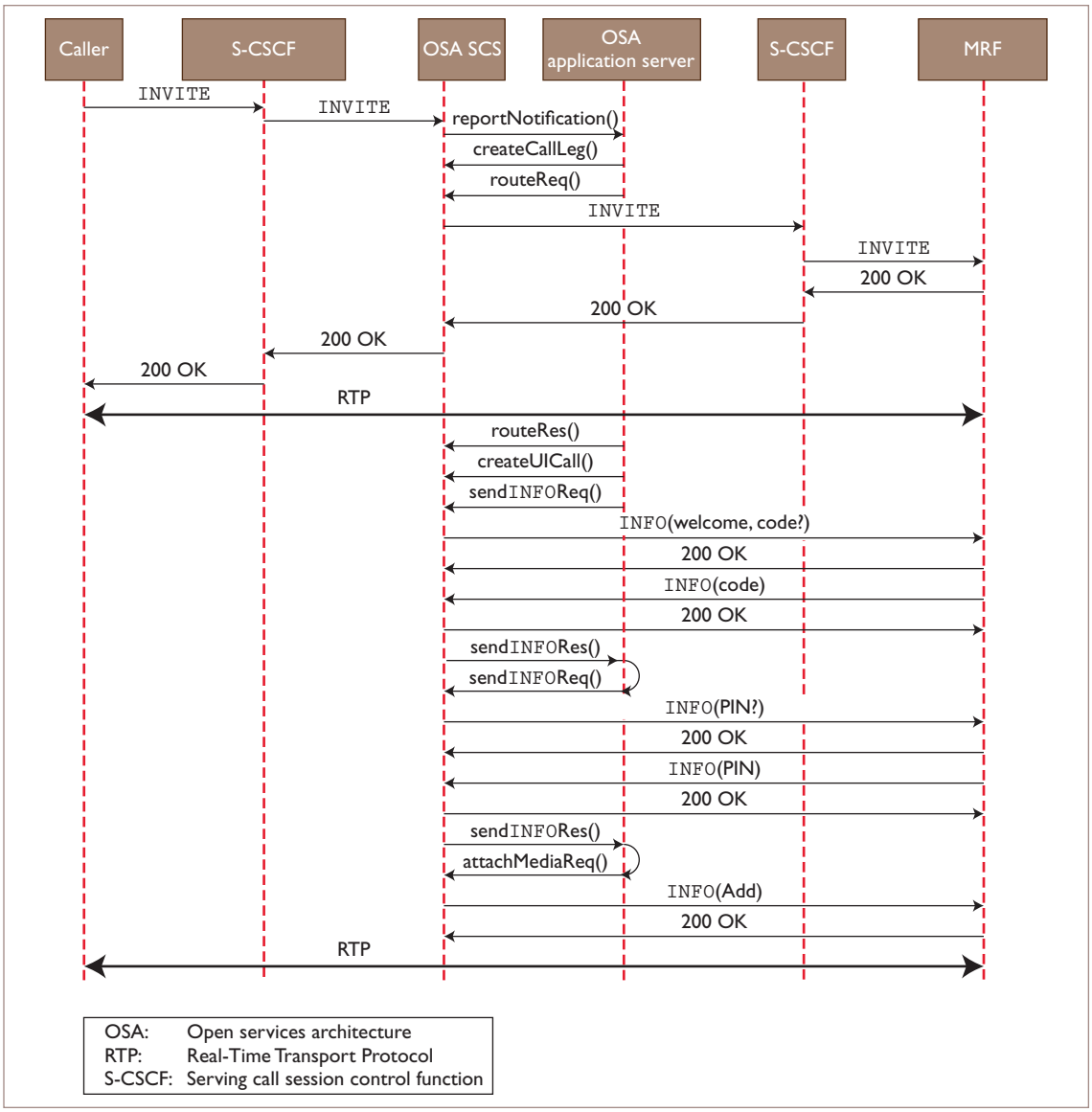

Figure 3. The audioconference service's call flow, managed by an OSA application server. The OSA service capability server (SCS) maps SIP messages into invocations of the OSA API (also called Parlay) and back.

derstand how to build application servers and deploy IMS applications and end-user services, we must examine the technologies that we could use to build them.

Existing technologies for building IMS applications aren't all equivalent. Some can be used to implement a complete application server, others can be used to implement just a layer of the application server. Some of them are SIP-dependent, so they can be used only to build SIP application servers. Others are protocol independent, so they can be used to build both SIP and OSA application servers. We classify these application server technologies into three families: SIP programming techniques, APIs, and service execution environments.

SIP Programming Techniques

SIP programming techniques let application programmers access basic SIP functionalities to program the SIP applications. The two techniques in this category are the SIP common gateway interface (SIP CGI)[10] and the SIP servlet.[11] These techniques are SIP-dependent, so they can only be used for SIP application servers.

SIP-CGI. SIP CGI was inspired by HTTP CGI, a tool for creating dynamic content for the Web.

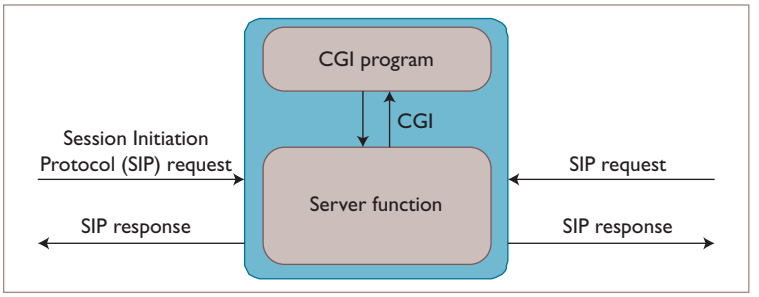

Figure 4. CGI-based SIP application server. When a server receives a SIP request, it invokes a SIP CGI script. The script performs the required processing and generates signaling instructions for the server to execute.

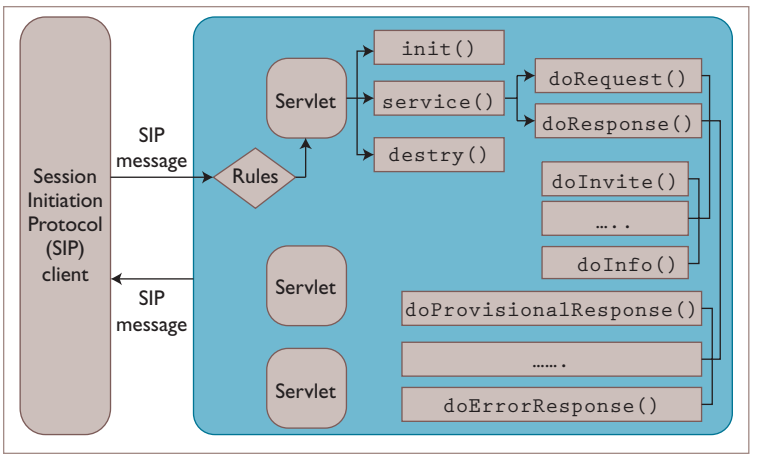

Figure 5. Servlet-based SIP application server. The container passes objects representing SIP messages to the servlet. The servlet has access to all the SIP messages' headers through those objects and, with this information, decides how to respond to a message.

When a server receives a SIP request, it invokes a SIP CGI script. The server passes the message body to the script through its standard input and sets environmental variables containing information about the message headers, user information, and server configuration. The script performs some processing and generates signaling instructions for the server to execute. The SIP CGI script can instruct the server to generate a SIP response, proxy a request, create a new request, or change a request's headers. Figure 4 shows the functional model of a CGI-based SIP application server.

In a CGI-based IMS SIP application server, IMS end-user services are written as CGI scripts. The server calls a CGI script for each incoming call. The script runs the service logic and returns to the server a list of actions to perform on the SIP request. Using IMS terminology, we can consider the CGI scripts IMS applications.

SIP servlet. The SIP servlet API is a Java extension API for SIP servers, inspired by the HTTP servlet concept. A SIP servlet is a Java-based application component that's managed by a container and performs SIP signaling. These containers, sometimes called *servlet engines*, are server extensions that provide servlet functionality. Servlets interact with SIP clients by exchanging request and response messages through the servlet container. The container passes objects representing SIP messages to the servlet. The servlet has access to all the SIP messages' headers through those objects and, with this information, decides how to respond to a message. Servlets can answer or proxy requests, create or forward responses, and initiate new SIP transactions. The container provides many services that the SIP servlet can exploit, such as automatic retries, message dispatching and queuing, forking and merging, and state management.

The servlet itself only manages high-level message handling and service logic. Figure 5 shows the relationship between the servlets and the container as well as the different methods of the servlet interface.

An IMS servlet-based SIP application server is a SIP servlet container. IMS applications are SIP servlets (or a group of SIP servlets). When the server receives a new request, it applies some preconfigured rules to select the servlet (the application) to process the request. The servlet executes the service logic and invokes the container capabilities to send and receive SIP messages.

APIs

Several APIs for building communication application servers can be used as part of an IMS application server. These APIs wrap up network and protocol functionalities into an easy-to-use abstract software component.

The APIs provide high-level object-oriented interfaces that let programmers implement communication applications. Figure 6 shows an API-based IMS application server. When the API and application reside on different machines, a remote-procedure call mechanism can allow interaction between them.

Parlay (www.parlay.org) is a set of API specifications for managing network services such as call control, messaging, and content-based charging. The Parlay group first proposed it for

telephone networks, and the 3GPP later adopted it as the OSA API to give the OSA application server access to the IMS network functionalities. The Parlay API supports all call-control functionalities that previous telephony APIs provided and offers some new features such as mobility, presence, and data session control. Moreover, Parlay is the only API that includes VoIP and SIP systems in its specifications.

Two versions of the OSA API exist:

- the standard version, a simple API specification that can be programmed using any object-oriented programming language; and
- the Web service-based version, or Parlay X.

Although the IMS documentation implies that the OSA API is for building OSA application servers only, nothing prevents a SIP application server from being Parlay-based. In this case, the Parlay API will likely reside on the same machine as the application server and will be invoked locally.

In addition to Parlay, developers can use three well-known APIs for information technology and PSTN integration to build an IMS application server:

- *Telephony API* (TAPI), introduced in 1993 by Microsoft and Intel and limited to Windows-based systems;
- *Telephone services API* (TSAPI), developed by Novell and Lucent Technologies; and
- *Java telephony API* (JTAPI), a specification for Java-based computer-telephony applications.

TAPI, TSAPI, and JTAPI all define methods that let telephony applications set up and tear down calls, monitor progress, perform identification, and activate features such as hold, transfer, conference, call park, and call pickup. They can redirect and forward calls, answer and route incoming calls, and generate and detect DTMF signals.

All three APIs hide network details and protocols from application programmers and provide them with an easy programming interface with which to build sophisticated telephony services. These APIs specify a set of packages, classes, and methods, which network elements should expose and applications should invoke to let the applications access network functionalities. The API implementation notifies the application of events occurring in the network and instructs the network components to execute the application's commands.

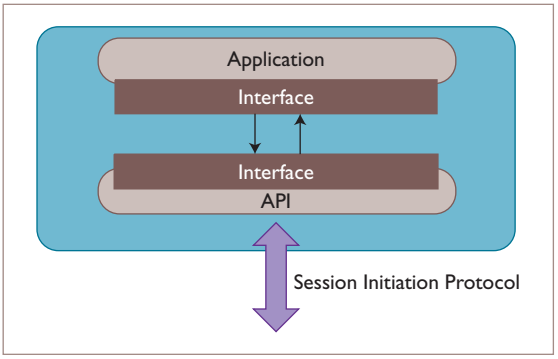

Figure 6. API-based IMS application server. The API wraps network and protocol functionalities into an easy-to-use abstract software component.

Even though these three APIs can theoretically be used to build IMS SIP application servers, they've stirred little interest because the new features and capabilities that SIP provides have superseded them. Of course, you can't use any of them to build an OSA application server because such servers require the OSA API.

Service Logic Execution Environments

A service logic execution environment (SLEE) is a high-throughput, low-latency event-processing application environment designed for communication applications that can be used to build IMS application servers.

Jain SLEE (http://jainslee.org) is the Java specification of the SLEE concept. To our knowledge, it's the only industry standard specification of a SLEE. It specifies the runtime execution environment, called a *container*, and communication services' internal architecture. A Jain SLEE service is a collection of reusable object-oriented components — *service building blocks* (SBBs) — running inside the container. Figure 7 shows the architecture of a Jain SLEE-based application server.

An SBB is a software component that sends and receives events and performs computational logic. An external resource, such as the communications protocol stack, the SLEE container, or another SBB event can generate events.

A Jain SLEE application server interacts with external resources, such as network protocols and TAPIs, through resource adapters, which adapt resources to SLEE requirements.

An IMS SLEE-based application server is a

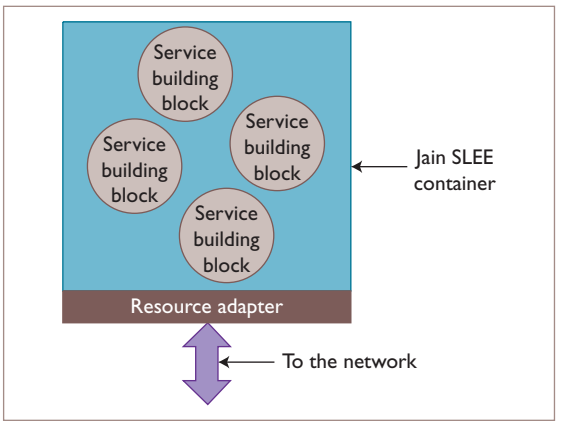

Figure 7. Jain service logic execution environment application server. A Jain SLEE service is a collection of reusable object-oriented components — service building blocks (SBBs) — running inside the container.

Jain SLEE container, and IMS applications are SBBs. Jain SLEE isn't related to any signaling protocol. It can be used for both IMS SIP application servers and OSA application servers. For a SIP application server, it needs a resource adapter for SIP, and for an OSA application server, it needs a resource adapter for the OSA API.

IMS Application Server Requirements

IMS application servers should fulfill several requirements:

- *Support for a wide range of end-user services.* By supporting a wide range of services, application servers can provide a single solution platform.
- *Rapid service creation and deployment.* Rapid service creation is crucial to success in the marketplace. Service providers should be able to rapidly specify, design, test, and install new services.
- *Easy service customization and tailoring.* Service providers must be able to change the service logic rapidly and efficiently. Customers also demand control of their own services to meet their individual needs.
- *Independent evolution of services and network infrastructure.* Services should be defined independently of a specific network technology (SIP). Conversely, the service architecture's flexibility should facilitate the exploitation of new technologies.
- *Support for multiplayer (or open) environments.* The application server should support services, software, and hardware components from different vendors while maintaining interoperability.
- *Universal service access.* Users must be able to access services independently of the physical location and types of terminals being used.

These requirements are rooted in the Telecommunications Information Networking Architecture Consortium's work, the most in-depth work on communication service architectures to date (see www.tinac.com). Because of the requirements' generic character, researchers have used them beyond their original scope in intelligent networks — for example, to discuss Internet telephony service architectures[12] — so we feel strongly that they can also apply to IMS.

To relate the IMS application server requirements to the technologies presented, we studied which of the requirements are intrinsically met by the SIP application server and OSA application server, which can be met using the presented technologies, and which require the use of other technologies.

Support for a Wide Range of Services

Both the SIP and OSA application servers can provide any SIP-based communication service. The OSA API, however, lacks some capabilities that are available in SIP, such as call forking. The absence of these capabilities might prevent the OSA application server from supporting some services. 3GPP and Parlay working groups are addressing these issues.

Rapid Service Creation and Deployment

Using SIP CGI or SIP servlets to program the SIP application server services requires knowledge of the SIP, which makes it relatively difficult. Still, using the SIP servlet approach has some advantages over using the CGI approach:

- Containers ease application development by handling some of the SIP complexity (automatic retrial, provisional responses, and so on).
- The API approach provides more convenient access to various structures (such as SIP URLs and contact addresses) by representing them as abstractions rather than untyped strings.
- The servlets have ready access to a wide variety of Java APIs, such as directories, databases, and security algorithms.

- Servlet deployment is more convenient because it uses well-defined XML files, whereas the deployment of CGI scripts isn't standardized.

Programming OSA application server services requires only general programming skills; no protocol knowledge is needed. This facilitates service creation in the OSA environment. Also, in the SIP application server, adopting a layered programming approach that hides SIP details from service programmers could ease service creation. You could do this using, for example, TAPI or Parlay. Parlay would be the best choice because it supports SIP functionalities. Using Parlay will result in a SIP application server that is similar to the OSA application server.

With Jain SLEE, which can be used for both SIP and OSA application servers (but requires resource adopters), service creation and deployment is easy because services are created as standalone SBBs and deployed in a standard manner.

Easy Service Customization and Tailoring

Easy service customization is an important requirement for service providers. Scripting is a good approach to ensure a high customization level of IMS services. Generally, solution providers offer proprietary scripting languages. However, the Voice Browser Call Control language,[13] an XML-based language, offers an alternative. CCXML was proposed to script the logic of telephony applications and has potential in both SIP and OSA application servers. It provides a standard way for controlling routing, bridging, outbound calling, and conferencing actions, as well as for executing Voice Extensible Markup Language (VoiceXML)[14] dialogs. Unfortunately, CCXML is only designed for telephony, so you can't use it to customize other IMS services, such as presence and instant messaging.

The application server should also provide a mechanism for customers to set and modify their preferences. Most approaches use Web interfaces and interactive voice response menus to do so. However, you could also use scripting languages such as the Call Processing Language[15] for this purpose. CPL is an XML-based scripting language that lets users control their Internet telephony services. End users can use CPL to describe their preferences, such as call forwarding based on time of day or call rejection based on caller identity. Like CCXML, CPL can be used in both SIP and OSA application servers and is designed for telephony services.

Finally, using CPL and CCXML requires interactions between the application server technology and the CCXML or CPL scripts and, as far as we know, is usually done in an ad hoc manner.

Independent Evolution of Services and Network Infrastructure

The SIP application server has signaling capabilities and is directly involved in the calls' signaling flow. So, it doesn't meet the requirement of independent evolution of services and network infrastructure. This limit restricts the services provided to the SIP technology. Changes in SIP might imply modifications to the services on the SIP application server, and the migration to another network protocol (such as H.323 or any future protocol) will imply the reimplementation of these services. The OSA application server doesn't have this restriction.

A layered programming approach can satisfy the rapid service creation and deployment requirement.

Support for a Multiplayer Environment

The IMS architecture is modular and lets service providers integrate different elements from different vendors. We can push this modularity further if the application server software architecture is also standardized. Using Jain SLEE, for instance, would let services providers run different services from different vendors in the same Jain SLEE containers.

The TAPIs and Parlay also provide a certain multilayered environment support for the SIP application server. They split the application server into two software layers that different vendors can provide. Similarly, using the SIP servlet approach to build a SIP application server allows for the deployment of services from different vendors into the same servlet container.

Universal Service Access

Both SIP and OSA application servers meet this requirement. SIP-based terminals can access, independently of their physical location, the services both server types provide.

Analysis

As Table 1 shows, from a purely technical per-

Table 1. Comparison of application server implementation technologies.

Feature	SIP programming techniques		Programming interfaces		Service logic execution environments
	CGI	Servlet	Telephony	Parlay	Jain SLEE
Support for a wide range of services	Yes	Yes	Limited	Yes	Yes
Rapid service creation and deployment	No	Medium	Yes	Yes	Yes (but high learning curve)
Easy service customization and tailoring	Possible	Possible	Possible	Possible	Possible
Independent evolution of services and network infrastructure	No	No	Yes	Yes	Yes
Support for a multiplayer environment	No	Limited	Medium	Medium	High
Universal service access	Yes	Yes	Yes	Yes	Yes

spective, the Jain SLEE architecture most closely meets the TINA-C requirements. However, the programming community perceives it as complex, which will certainly negatively affect its adoption. Moreover, large-scale adoption of Jain SLEE would require standardizing a Parlay resource adapter for OSA application servers and a SIP resource adapter for SIP application servers. This leads us to a second point — that the SIP application server should also be built over Parlay. A Parlay-based SIP application server both closely meets the TINA-C requirements and lets service providers deploy the same applications on both SIP and OSA application servers when necessary.

Some of the application server technologies that we've presented, such as Parlay and Jain SLEE, are complementary and can be used together. For example, Figure 8 shows how you can build a SIP application server using Jain SLEE and Parlay. In addition, the choice of an IMS application server implementation technology doesn't depend on the service to be built over it. You can use all the technologies we've presented (except TAPIs) to build any IMS service.

Implementation Experience

As noted, we implemented an audioconference service. Although we didn't originally design our media server and SIP application server for IMS (we describe an early version of the media server elsewhere[16]), their migration to IMS is well under way.

We implemented our SIP application server using Java servlet technology and the interaction with the media server (MRF in IMS terminology) using SIP and MSCML. We deployed the servlet that manages the conference service — ConfServlet — in the same container as the other services' servlets, such as the BasicCallServlet, IVRServlet, and VoiceMailServlet.

In our application, calling a specific URI triggers the ConfServlet. For calls coming from the PSTN, the gateways must map the external phone number called to the specific conference URI.

SIP servlets are powerful and easy to implement for programmers with HTTP servlet experience and SIP knowledge. However, they introduce the risk of mixing front-end functions with business logic. The SIP servlet environment lacks a rigid component model that separates call control from the business logic classes and persistence layer (similar to the MVC frameworks used in Web development). Finally, unit testing SIP servlet code (as well as any SIP application server code) isn't easy because it requires simulating the communication protocol. Alleviating this inconvenience will require frameworks similar to those used for HTTP.

We can deduce two important points from this review. The first is the absence of an abstract standard internal architecture for IMS application servers. The different technologies have different visions. Second, from a purely technical perspective, we believe that the Jain SLEE architecture can become the standard IMS application server architecture.

These points notwithstanding, we note that

the SIP servlet approach is currently the most popular application server technology. Both the Web and open source communities are encouraging its adoption, so it will likely be the major platform for at least the first generation of IMS SIP application server services.

However, the technologies we presented aren't an end point in the evolution of SIP service environments. In future work, we plan to investigate the use of OSGi (www.osgi.org) to build IMS application servers. OSGi is a successful architecture with a high level of modularity and easy service deployment that can be successfully applied to communication systems.

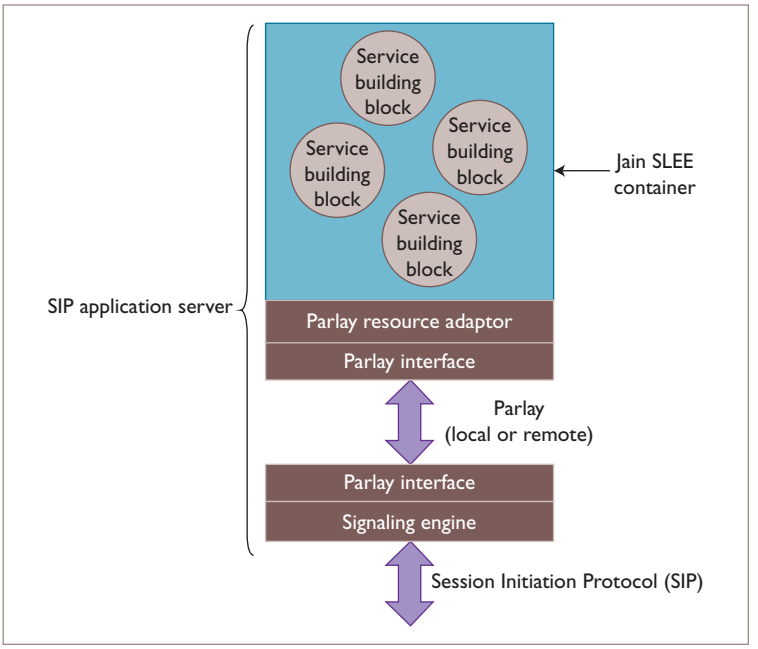

Figure 8. Jain SLEE/Parlay integration. Parlay and Jain SLEE are complementary and can be used together to build a SIP application server.

References

1. Third Generation Partnership Project, "Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS), Stage 2 (Release 7.5.0)," Sept. 2006.
2. M. Handley et al., *SIP: Session Initiation Protocol*, IETF RFC 3261, June 2002; www.ietf.org/rfc/rfc3261.txt.
3. ETSI, "Telecoms and Internet Converged Services and Protocols for Advanced Networks (TISPAN): NGN Functional Architecture Release 1," Aug. 2005; http://portal.etsi.org.
4. G. Camarillo and M.A. Garcia-Martin, *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds*, John Wiley & Sons, 2nd ed., 2006.
5. K.D. Wong and V.K. Varma, "Supporting Real-Time IP Multimedia Services in UMTS," *IEEE Comm.*, Nov. 2003, pp. 148–155.
6. P. Calhoun et al., *Diameter Base Protocol*, IETF RFC 3588, Sept. 2003; www.rfc-editor.org/rfc/rfc3588.txt.
7. F. Cuervo et al., *Megaco Protocol Version 1.0*, IETF RFC 3015, Nov. 2000; www.rfc-editor.org/rfc/rfc3015.txt.
8. E. Burger, J. Van Dyke, and A. Spitzer, *Basic Network Media Services with SIP*, IETF RFC 4240, Dec. 2005; www.rfc-editor.org/rfc/rfc4240.txt.
9. E. Burger, J. Van Dyke, and A. Spitzer, *Media Server Control Markup Language (MSCML) and Protocol*, IETF RFC 4722, Nov. 2006; www.rfc-editor.org/rfc/rfc4722.txt.
10. J. Lennox, H. Schulzrinne, and J. Rosenberg, *Common Gateway Interface for SIP*, IETF RFC 3050, Jan. 2001; www.rfc-editor.org/rfc/rfc3050.txt.
11. JSR Expert Group, "SIP Servlet API Specification Version 1.0," Feb. 2003; http://jcp.org/aboutJava/communityprocess/final/jsr116/.
12. R.H. Glitho, "Advanced Services Architectures for Internet Telephony: A Critical Overview," *IEEE Network*, July 2000, pp. 38–44.
13. R.J. Auburn et al., "Voice Browser Call Control: CCXML Version 1.0," World Wide Web Consortium (W3C) recommendation, June 2005; www.w3.org/TR/ccxml.
14. S. McGlashan et al., *Voice Extensible Markup Language (VoiceXML) Version 2.0*, World Wide Web Consortium (W3C) recommendation, Feb. 2003; www.w3.org/TR/voicexml20.
15. J. Lennox, X. Wu, and H. Schulzrinne, *Call Processing Language (CPL): A Language for User Control of Internet Telephony Services*, IETF RFC 3880, Oct. 2004; www.rfc-editor.org/rfc/rfc3880.txt.
16. H. Khlifi and J.C. Grégoire, "Design and Performance of a Stand-Alone Media Server," *Proc. 2005 Systems Comm.*, IEEE CS Press, 2005, pp. 147–152.

Hechmi Khlifi is a consultant with Ericsson Canada. His research interests include Internet real-time applications, voice over IP, and telecommunications service engineering. Khlifi has a PhD in telecommunications from the National Institute of Scientific Research, University of Quebec. Contact him at khlifi@emt.inrs.ca.

Jean-Charles Grégoire is a professor at the Energy, Materials, and Telecommunications Center of the National Institute of Scientific Research, Canada. His research interests include all aspects of telecommunication systems engineering, including protocols, distributed systems, network design, and performance analysis. Grégoire has a PhD in technical sciences from the Federal Polytechnic School, Lausanne, Switzerland. Contact him at jean-charles.gregoire@emt.inrs.ca.