A VOICE OVER IP SERVICE ARCHITECTURE for Integrated Communications

DANIELE RIZZETTO AND CLAUDIO CATANIA
Hewlett-Packard Laboratories

A novel architecture supports flexible integration of IP-based telephony services with legacy circuit-switched services by defining the means for convergence at the communications-control layer of the network, rather than the transport layer.

Voice over IP has paved the way for a global approach to designing communication service platforms, but the real driver is the convergence between heterogeneous communication network technologies. Schulzrinne[1] considered some of the problems arising from this convergence and described possible scenarios for the evolution of the current communication world. The integration needs of existing systems slow down the introduction of new technologies and communication tools. Integrating heterogeneous networks at the communication-control layer can speed up this process. Integrating the intelligent network (IN)[2] and the Internet is a key step in this direction.

This article advances a high-level abstraction for integrating IP-based telephony intelligent services with legacy circuit-switched telephony IN services. We do not propose new protocols; rather, we present a novel service architecture, its components, and their behavior. We focus on International Telecommunication Union Recommendation H.323 for VoIP[3] (see the sidebar "H.323: An Overview" on the next page) and its central network component, the gatekeeper. In particular, we describe the architectural design and findings of the Hewlett-Packard Laboratories Bristol (HPLB) gatekeeper, an experimental prototype implemented on top of an H.323 platform. We also describe a practical demonstration of interoperation between the Internet and the IN.

ARCHITECTURE RATIONALE

Our proposed architecture takes advantage of VoIP's flexibility by defining the means for convergence at the communications-control network layer, rather than the transport layer. The ever-growing availability of communication networks that offer similar functionality using different technologies suggests the need for a high-level abstraction to execute the services inde-

 1089-7801/99/$10.00 ©1999 IEEE

pendent of the user's network connection. Our approach defines a network-independent service environment into which different communication stacks can be plugged. Uncoupling the protocols from the platform is the first step toward convergence at the control layer.

With VoIP, communications services are more logically implemented in terminals than in centralized service components.[4] This is because IP terminals are far more powerful and intelligent than traditional telephones and provide better user interfaces. As the migration to IP terminals gets under way, more services will be implemented in devices at the edge of the network. Nevertheless, some services implement functionality that cannot run on the client side. Among them are call routing, directories, and services that suffer from the "always-on" problem of client end terminals, which stems from the fact that VoIP terminals are not always on and connected to a network. Our approach, although essentially server-based, acknowledges the "intelligence at the edge" vision and supports it by offering an entirely IP-based platform easily accessible by smart terminals.

Integrated service platforms must provide services that end users can easily access and configure.[5] The standardized access provided by IP is instrumental in achieving this goal, but the service model has to enable this capability. Our agent-based approach to provisioning communication services lets users modify the logic of their services and decide how service

H.323: AN OVERVIEW

H.323 is an ITU umbrella Recommendation for multimedia communications over local area networks (LANs) that do not provide a guaranteed quality of service (QoS)[1] (for a concise introductory tutorial see DataBeam's "A Primer on the H.323 Series Standard"[2]). The standard covers point-to-point communications and multipoint conferences. It addresses call control, multimedia management, bandwidth management, and interfaces between LANs and other networks.

Architecture

The Recommendations are still under development (H.323v2 was approved in January 1998), but some basic concepts are widely accepted. The architecture elements are

- *User terminals.* Terminals are the LAN client endpoints that provide real-time two-way communications.
- *Gateways.* GWs translate signaling and media streaming exchanged between H.323 and PSTN endpoints.
- *Multipoint control units.* MCUs enable conferencing.
- *Gatekeepers.* GKs are responsible for call authorization, address resolution, and bandwidth management. They intercept call signaling between endpoints and provide "signaling-based" advanced services.

Terminals, gateways, and MCUs are generically addressed as endpoints. GKs provide those services that cannot be decentralized and implemented by endpoints.

The Registration Admission Status (RAS) protocol is the key GK protocol. RAS messages are carried in User Datagram Protocol (UDP) packets exchanged between an endpoint and its GK. When an endpoint is switched on, it sends an RAS registration request (RRQ) to the GK. This message contains information such as terminal transport address, user alias, and E.164 telephone number. If the GK accepts the registration, it sends a Registration Confirm message (RCF); otherwise, it sends a Registration Reject (RRJ) message. The GK and its registered endpoints are called a zone.

Call Setup Life Cycle

The H.323 call setup life cycle can be split into three phases (named according to the protocol used):

RAS. To make a call, an H.323 endpoint sends an RAS Admission Request (ARQ) message to the GK. This message contains the destination alias, which is the name or phone number of the user to be contacted. The GK may grant permission for the call by sending back an Admission Confirm (ACF) message containing the actual transport address associated with the called party alias. The GK may also reject the request with an Admission Reject (ARJ) message for a variety of reasons, such as "not enough bandwidth" or "security violated." Therefore, during this phase the GK accomplishes three functions: address translation, call authorization, and bandwidth management.

Q.931. This phase is derived from ISDN end-to-end call setup signaling (SETUP, PROCEEDING, ALERTING, CONNECT) and provides the logical connection between the two endpoints—the calling party and the called party. In H.323, Q.931 is implemented on top of TCP.

H.245. During this phase, the two endpoints exchange capabilities. They agree on the nature of the information

objects will behave when reacting to events sent from the network and dialoguing with other entities involved in the communication session. A similar approach is the call management agent.[6]

Letting end users configure their own services requires careful analysis of the language used to implement them because badly programmed logic can harm the network's overall functionality. A well thought-out language that is user-friendly, can handle complex and advanced services, and can offer the appropriate reliability features required by communication networks has yet to be devised. Our proposal, while offering a simple rule-based language to program the service logic, implements peer-to-peer negotiation in the service platform among all the service objects of the parties involved in the communication session. We believe this approach solves the above-mentioned problem and takes a step beyond the traditional telephony world service model, in which the called party plays a dominant role in triggering and executing services.

SERVICE ARCHITECTURE

The main components of the service architecture are shown in Figure 1. The GK platform provides a high-level application programming interface (API) that implements an abstraction layer for executing services independent of the underlying networks. Each GK platform has a local service platform in which service objects are executed. A service object implements ser-

they will exchange through the media channel (audio, video, or data) and its format (for example, compression or encryption). H.245 is implemented on top of TCP.

After these three phases, the Real-Time Protocol/Real-Time Control Protocol (RTP/RTCP, running on top of UDP) media channels between the two endpoints are opened according to the capabilities exchanged, and the actual media communication starts. Data communications are based on the T.120 specification. During the call, dual-tone multifrequency (DTMF) touch tones are transmitted over the LAN through the H.245 User Input Indication message.

Call signaling can be routed through the GK or routed directly between the endpoints. RAS is, by nature, GK-routed. A GK can decide to route Q.931 and H.245 through itself, so that it can act as a proxy between endpoints. If the GK intercepts the signaling it can perform call management, maintaining a list of ongoing H.323 calls in order to keep endpoints' state or to provide information for the bandwidth management function. In any case, the media flows directly between endpoints because the GK is just a signaling entity and cannot be called.

New Features

H.323v2 uses the H.235 standard to address authentication, integrity, privacy, and nonrepudiation.

It has also introduced the Fast Connect procedure, which allows endpoints to expedite the exchange of terminal capabilities by encapsulating them in Q.931 messages, thus avoiding the slow H.245 negotiation.

Supplementary services have been defined by the H.450 series. H.450.1 is the signaling protocol between endpoints for the control of supplementary services; H.450.2 defines call transfer of an established call; and H.450.3 defines call diversion for implementing call forwarding unconditional, call forwarding busy, call forwarding no reply, and call deflection. New services can be introduced through the standardization of a new H.450.X series. The series puts in place signaling mechanisms to control services, but do not define how the logic behind them should be implemented.

Beyond Signaling

The Internet Engineering Task Force proposal for VoIP signaling is the Session Initiation Protocol.[3] SIP goes beyond the simple signaling of voice communications. It focuses on setting up generic communication sessions, separating this phase from channel allocation and media transmission. SIP uses the REGISTER message to register to a SIP server and the INVITE message to initiate a call. This message can be sent directly to a SIP client or SIP server, which is comparable to a H.323 GK. If the SIP server is a proxy server, it will route the call through itself; if it is a redirect server, it will redirect the call to the actual destination.

REFERENCES

1. ITU-T Rec. H.323, "Visual Telephone Systems and Equipment for Local Area Networks which Provide a Nonguaranteed Quality of Service," Geneva, Switzerland.
2. DataBeam Corp., "A Primer on the H.323 Series Standard," http://www.databeam.com/h323/h323primer.html.
3. M. Handley, H. Schulzrinne, and E. Schooler, "SIP: Session Initiation Protocol," Internet draft, IETF, Jan. 1999; work in progress.

vices for a particular user, who can modify and configure them at any time. For each user there is a service object stored in a home server, which is shared among several users. The server can be located anywhere in the network. When a user changes location, the local GK-service platform accepts the registration and downloads the user's service object from the home server in which the object is stored.

HOME SERVER

The home server contains the service objects and provides them when requested by a service platform. It plays an essential role in supporting user and service mobility, synchronizing multiple instances of the service objects, and providing access to service creation environments.

A service object is located using the alias address of the user to which it belongs. We use the addressing scheme adopted by the Internet Engineering Task Force's Session Initiation Protocol (SIP),[7] in which users are identified by a structured e-mail-like alias: userName@HomeServer. When a service platform needs to download a user's service object, it uses this structure to locate HomeServer and download the service object of userName.

Obviously, this location mechanism is not valid when the user's alias address is outside the adopted addressing scheme. This is the case for E.164 telephone numbers, which in the Internet do not belong to any domain and therefore have to be translated into meaningful structured aliases. Various standards bodies—in particular, the European Telecommunications Standards Institute (ETSI) Telecommunications and Internet Protocol Harmonization over Networks (TIPHON) technical body and its Working Group IV[8]—are addressing this PSTN/IP integration issue, and we expect to use their results.

The home server is also responsible for synchronizing multiple requests for the same service object (for example, when there is more than one call attempt to the same user). This problem stems from the fact that the same service object can be accessed by different service platforms. It can be solved using a centralized approach (the home server is responsible for synchronization) or a distributed one (the service objects communicate among themselves to avoid inconsistencies).

The home server is also where service objects are created. In the traditional telephony network, terminals provide a very poor interface (an alphanumeric keypad with some extra buttons), and the only way to configure services is to type cumbersome sequences of digits. In our solution, a Web-based service creation environment (SCE) can offer a user-friendly GUI-based service configuration. Mizuno et al. describe a similar approach.[9]

SERVICE PLATFORM

The service platform controls service objects' execution life cycles. Once downloaded from the home server, the objects are executed locally and receive events from the underlying communication stacks.

In Figure 1 the GK platform provides the abstraction layer for events coming from a generic network. This abstraction requires the identification of commonalities between existing and emerging communication paradigms adopted by protocols used in wireless, fixed, and VoIP networks. Observe that virtually all of their functionality can be grouped into three main areas: a *registration* phase during which terminals communicate their presence and related data; an *admission* phase during which they ask for a called party's number translation; and a *call signaling* phase during which entities exchange messages to control the actual communication. This distinction allows us to define a level of abstraction that enables different communication stacks to be combined into a single platform.

The user objects encapsulated in service objects (Figure 1) receive fast-changing user information such as the current user location, the user status (busy, free, in a call setup, and so on), or the status of the calls in which the user is involved.

We categorize service objects into three classes.

- *Installed* service objects are downloaded at registration time. They live in the platform until the user unregisters (Figure 1).
- *Downloaded* service objects are downloaded during a call attempt. They live until the call terminates.
- *Replica* service objects are downloaded when other instances of the same service object are already installed into or downloaded to some other service platforms. They live in the service platform in which they are downloaded until the call terminates.

When a user registers with a local GK, the service platform sends a Notify message to the home server, downloads the appropriate service object, and installs it. The Installed service object gets a copy of the user object (created by the GK as soon as the user registers) and initiates it to receive events from the network via the GK and service platform. The service logic behavior is driven by such events, which carry information about the user status and status of

the possible calls in which the user is involved. Examples of services that can take advantage of this updating process are:

- *International call barring*. When a user tries to place a call, the call request is passed to the service object, which uses its logic and the data stored in the user object to do prechecking and decide whether the call can progress.
- *Presence services*. Once the user registers, the service object forwards the user's location to other applications, which can start prescheduled calls or deliver waiting messages.

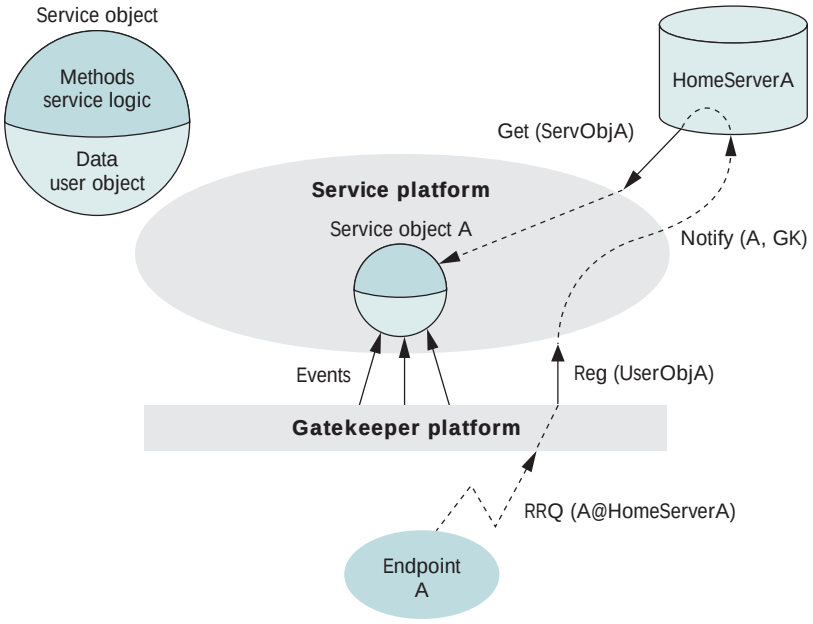

Figure 1. Service architecture (registration time).

When a registered user (for example, user A@HomerServerA in Figure 2) wants to place a call, the GK with which the individual is registered alerts the associated service platform. This platform then checks if the called party's service object is installed locally; if it is not, the appropriate service object is downloaded from the called party's home server. At this stage, two scenarios are possible.

In the first scenario, the called party is registered with a different GK (GK Y in Figure 2). In this case the home server of user B (the called party) creates a called party Replica service object, which contains a reference to the Installed called party service object running in the service platform associated with GK Y. The replica is downloaded into service platform X and retrieves an updated user object from the Installed service object, which is also responsible for the synchronization between itself and its replica.

In the second scenario, the called party is not registered with another GK, so its downloaded service object does not need an updated user object. The called party's status indicates that the user is disconnected. (This is typical when the service logic routes the call to a voice-mail system.) Synchronizing between possible Replica service objects, due to simultaneous call attempts, is left to home server B.

To summarize, the goal is to involve all service objects in a call in the same service platform, which is the one associated with the GK with which the calling party is registered.

An interesting evolution of the proposed architecture is intelligent edge devices running a lightweight version of the service platform. This implies moving it from the service execution layer to the user layer (Figure 2) and removing the GK platform between the device and its service platform. This lets devices access other service platforms directly, avoiding GK mediation.

SERVICE OBJECT BEHAVIOR

Once the service platform has involved all the service objects in the call, the peer-to-peer call negotiation between the objects can start. The calling party's service object invokes a "locate" method (part of the interface of every service object), which chooses the correct destination transport address of the called party. This process could use the user object's fast-changing data.

The service logic implemented by the locate method consists of a set of "if" conditions then actions. The condition field contains information such as time of day, call originator ID, and devices the user is currently using. Possible actions include return transport address, reject, and request further information.

The called party's service object can return a single response or a list of possible options. For example, when the user is unavailable, the calling party's service object can either stop the call or proceed, leaving a message with a voice mail system. When the calling party's service object gets the response, it can execute more logic (address postchecking, availability, and so on) and

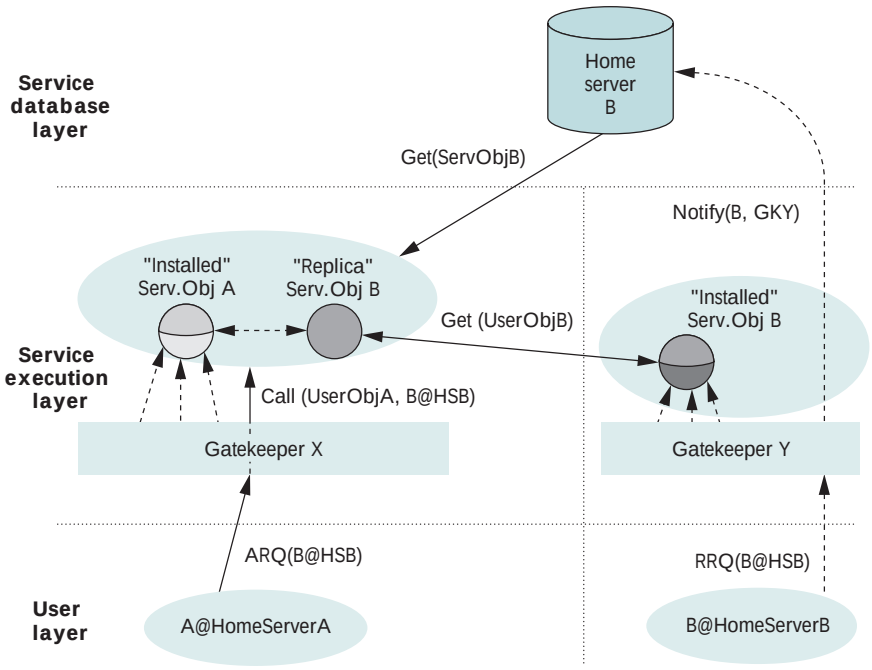

Figure 2. Call placing (from A to B).

then passes the response to the calling party's terminal through the service platform.

For privacy and security reasons, the called party's service object does not have direct access to the calling party's user object. However, it can ask the calling party's service object for information about its user. During this negotiation phase, the two parties agree on the media communication type and format.

As described above, the negotiation phase requires more than a single invocation of the locate method; therefore, it is more reasonable not to call the method remotely on the home server but to transport the whole object through the network.

If $T_{download}$ is the service object downloading time, T_{remote} is the transmission delay for a single message, T_{local} the transmission delay within the platform, and n the number of messages exchanged between the two service objects, the negotiation phase overall delay is $T_1 = T_{download} + 2*n*T_{local}$ if the whole object is downloaded and $T_2 = 2*n*T_{remote}$ if it is not. Assuming T_{local} is negligible and $T_{remote} \cong T_{download}$, then $T_1 << T_2$ when n is significantly large. This is the main reason for downloading the called party's service object, even if this could have an impact on postdial delay.

In addition, service mobility has some security implications. Since service objects are downloaded into a foreign service platform, their internal data and methods should not be completely exposed. To overcome this problem, service objects could be implemented as empty proxies and their methods executed via remote calls. This could be a feature home server providers make available. Obviously, security advantages are obtained at the expense of the performance benefits gained by downloading the service objects.

Alternatively, host service platforms have to shield local resources to prevent malicious service objects from accessing local information and communicating relevant data back to the home server or to other applications. This goal can be achieved by using authentication and signed code techniques prior to the execution. Obviously, interoperability between service platforms and home server providers requires well-defined communication interfaces and service-level agreements between them.

PRACTICAL IMPLEMENTATION

Figure 3 (next page) shows the implementation of the service architecture. Because we wanted our prototype to interoperate with currently available VoIP clients, the most popular being Microsoft NetMeeting, we chose H.323 as the VoIP protocol.

The H.323 stack handles messages coming from the IP network. The GK core logic stores information about ongoing calls (call objects) and registered users (user objects) into the respective databases (Figure 3). The components located above the dashed line have been implemented using Java, taking advantage of the built-in remote method invocation (RMI) for communication between these components.

HPLB GATEKEEPER

The HPLB Gatekeeper implements the functionalities described by the H.323 standard except for bandwidth management (Figure 4). Its internal architecture is composed of three functional blocks: the H.323 stack, the interface layer, and the GK, which is divided into the Java communication API and the core logic.

The Java communication API, which sits between the core logic and interface layer, was designed to be independent of the underlying communication protocols. This creates an additional level of abstraction for registration and call admission (RAS) and Q.931 (call signaling). Thanks to

this abstraction a variety of protocols can be plugged into the platform, allowing seamless integration between different networks.

The interface layer between the H.323 stack API, which is written in C, and the Java communication API, is based on the standard Java Native Interface (JNI) mechanism. This layer can be split into two major functions: the message dispatcher and the native methods. The message dispatcher is triggered by callback functions coming from the stack API. It instantiates Java objects with the parameters contained in the received messages and invokes the appropriate communication API methods to pass these objects. Native methods simply map requests from the Java GK logic into H.323 stack API method invocations.

The GK core logic has access to the user and call databases (Figure 4) that contain data related to registered users and ongoing calls, respectively. The user database contains user objects. They contain user status and basic properties such as user alias, E.164 telephone number, and IP address. The status can be DISCONNECTED, REGISTERED, BUSY, or SETUP (if the user is in the call setup phase). The call database contains call objects, consisting of a call-ID, the references of the user objects of all the parties involved in the call, and the call status. The call status reflects the Q.931 messages exchanged between the two endpoints through the GK.

When the GK receives an ARQ, the service platform is invoked to start the service objects' negotiation. After the response is sent back by the service platform and the permission for the call is granted, the Q.931 signaling starts to flow through the GK, updating the related users and call objects. We route H.245 directly between the endpoints because it involves the exchange of a large amount of information of little interest to the GK itself.

The HPLB Gatekeeper core logic implements a simple internal admission policy based on resource availability. Two parameters, based on network capacity, can be set up at launch time: maximum number of registered users and maximum number of ongoing calls. Registrations or call requests that could cause exceeding these parameters are rejected. Future admission policies can be based on more complex interactions with firewalls or bandwidth management servers. These policies could require the interception of the H.245 flow. In fact, if the GK must allocate resources to guarantee QoS for a particular media connection, it should know the UDP port where the stream takes place. The GK can get this parameter by intercepting H.245 messages. This parameter is also useful for monitoring and measurement.

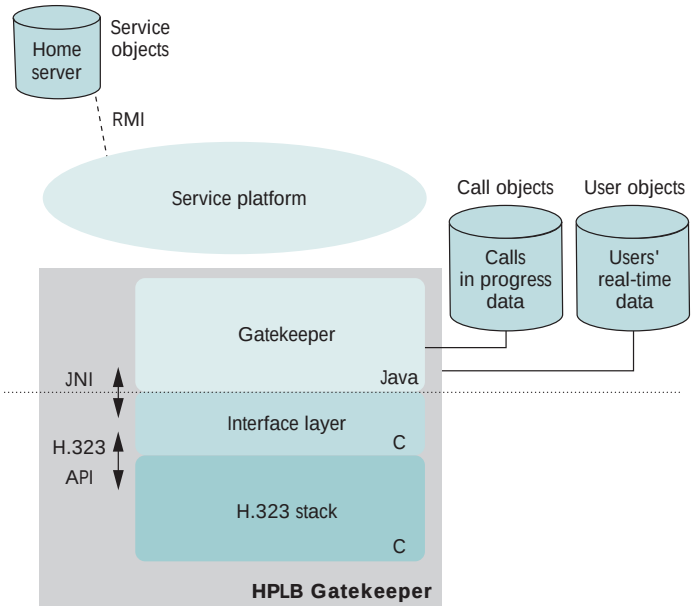

Figure 3. Practical implementation.

SERVICE OBJECTS

Service objects are stored and provided by the home server through the following simple RMI interface:

```
getServiceObject().
releaseServiceObject().
```

The mechanism for synchronizing multiple instances of the same service object is implemented within the home server. The service object Java interface is composed of the following methods:

- locate() is invoked by the calling party's service object onto the called party's service object to start call negotiation. In the current implementation this method has one parameter: calling party ID (E.164 number or, preferably, the structured alias when available, that is, when the call comes from an IP terminal).
- die() is used to terminate the object life cycle. Normally, this method is invoked by the service platform when the user unregisters, or by the calling party's object to force stop call negotiation. In the first case, the object is released and the home server notified. In the second case, the behavior is more complicated and depends on whether the object is installed, downloaded, or a replica.
- freeze() is invoked by an installed service object located in the service platform associated with the user's current GK onto a replica of it (Figure

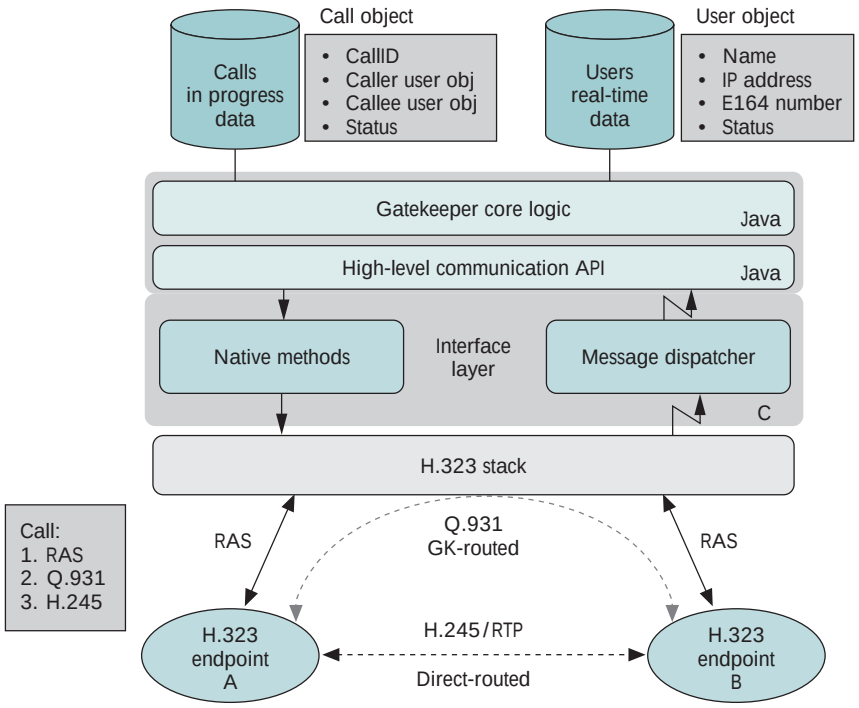

Figure 4. The HPLB Gatekeeper platform.

2). This method is used to temporarily freeze the replica to avoid conflicts between the decisions made by the two objects, which may affect the user.

- unfreeze().
- release() is invoked by a replica service object to notify the original installed object that it no longer exists.
- getUserStatus().
- getServiceStatus(). The service status can be normal or frozen. If a service object is frozen, it is likely that the other party's service object will stop the call.
- updateUserStatus() and updateCallStatus() are invoked by the service platform.

H.323v2 can benefit from the service object negotiation. In fact, its Fast Connect procedure coupled with the prior service object's call negotiation ensures that media channels with the optimal characteristics—chosen during this call negotiation—can be opened just after call setup. This requires passing terminal capabilities to the GK at registration instead of exchanging them during the H.245 phase, after service object negotiation. While this feature is already part of the SIP protocol, H.323 will provide it in future versions. This is one of the problems that must be solved when integrating different protocol stacks under a common level of abstraction.

It is also worth pointing out the relationship between H.450 and our proposal. The former is a signaling mechanism to control services implementation; the latter, the logic behind it.

Since service objects are essentially Java objects, we use Java as the service logic language, with service logic scripts interpreted by the Java Virtual Machine. Thus we do not need a new virtual machine for interpreting a new scripting language.

There are several proposals for a scripting language that handles calls. The most interesting of them, from a VoIP point of view, is the Call Processing Language (CPL)[10] for Internet telephony, under development by the IPTel IETF Working Group (see the article "Programming Internet Telephony Services" in this issue).

The two main objections to using a normal programming language like Java for service setup by end users are that service creation could be difficult, and that common languages are too complex for handling calls. This complexity can easily generate service misbehaviors. These drawbacks can be overcome by using controlled and simple service creation environments. In this context, we are exploring JavaBeans. JavaBeans can be manipulated graphically, which can generate an interesting evolution of the SCE. In fact, home server providers may make available different classes that can be directly customized by end users using JavaBeans. Different classes of services may be offered, from simple static call redirection (to a number decided by the user) to provision of the whole service object by an end user with good programming skills (which actually implements the whole logic).

DEMONSTRATION

The demonstration described here was presented at the Hewlett-Packard Laboratories review in Bristol last summer. Its aim is to show the ease of deploying personalized user services and the potential interaction with IN components, in particular with the service control point (SCP), the platform currently used by telecom operators to offer value-added services.[2]

The architecture, depicted in Figure 5, provides Internet call waiting, allowing users to receive incoming calls through a VoIP gateway when they are connected via dial-up and their telephone line is engaged.

The SCP is simulated by a Java application running on a machine connected to a private branch exchange (PBX), through specialized cards supporting the British signaling standard DPNSS. This "virtual SCP" communicates with the GK through Java RMI, but call requests reach the GK through the same high-level communication API used by the H.323 stack. A plug-in makes the virtual SCP interoperate with the GK through this API. The H.323 gateway runs on a different machine connected to the same PBX. The range of telephone numbers served by the PBX, connected to the normal PSTN, has been split: a subrange for the GW and another subrange for the virtual SCP. Each extension in the SCP subrange corresponds to an Internet call-waiting subscriber.

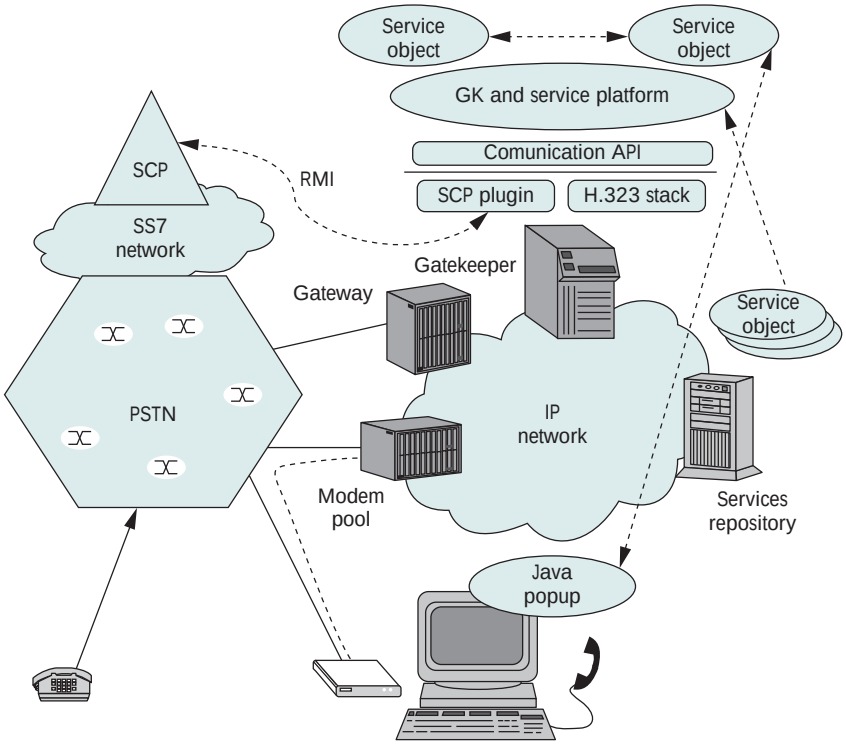

Figure 5. A practical demonstration.

When a call to a subscriber is attempted, the PBX tries to contact the subscriber telephone number. If the number is engaged, the virtual SCP is alerted. The virtual SCP then contacts the GK to find out if the subscriber is registered (note that the user must have registered with the same E.164 number). If so, the process goes on as described: The service platform is invoked and the personalized user service executed.

The demonstration is limited to Internet call waiting, so we assume that whenever the called user is connected via dial-up to an Internet service provider, the user is also registered to the GK associated with the SCP (there is a well-known fixed binding between them). If the called party is not registered to that GK, it means that the user is busy in a PSTN call, and the calling party will get the busy tone (alternatively, the normal IN call-waiting service can be activated).

In our demonstration, the logic in the called party's service object is programmed to examine its own user status. If the user is not busy in a VoIP call, its IP address is returned. Otherwise, the service contacts a small Java application launched at registration running on the current user machine and delivers a popup window, asking the user whether to take the call (leaving to the client the responsibility to manage multiple calls), divert it to another IP address, divert it to a telephone number, or reject it. To do this, the service platform security manager implements a dynamic policy that allows a service object to open a remote connection to the current machine where the registered user is located.

If the user returns an IP address, the GK allocates a "phantom" number on the range associated with the GW. The GK gets this range from the GW at registration through H.323. The GK returns the number to the SCP, keeping the association between the phantom number and the IP address. The virtual SCP communicates with the PBX to divert the incoming call to the GW phantom number. When the GW receives the call, it sends an RAS ARQ message to the GK with the phantom number as destination; the GK knows the IP address associated with that phantom number and connects the H.323 call.

In our demonstration we assume a static relationship between the SCP and the GK-service platform, and every time a user is connected via dial-up to the Internet the user registers with that GK. If the called party can be registered with a GK other than

that associated with the SCP, E.164 number translation is not a trivial task. In particular, the GK associated with the SCP receives, from the SCP, call requests containing just two parameters: caller ID and dialed number. It is important to stress that, in this context, the dialed number is a user identification and not a classic E.164 number. If there are no users registered locally with the E.164 numbers provided by the SCP, the GK must translate E.164 numbers into structured e-mail-like aliases in order to locate the home servers and then download the user objects and service objects. The GK can perform this translation in several ways. It can perform a lookup in a central database or ask other GKs through the Inter-Gatekeeper Communication Protocol (IGCP); see H.323 Annex G.[3]

If the GK can translate these E.164 numbers and locate the called party's and calling party's home servers, service execution proceeds as usual. If not, two scenarios are possible:

- If either the called party's home server is not located or the called party is not registered to any GK, the call is rejected.
- If either the calling party's home server is not located or the calling party is not registered to any GK, a dummy service object is created to interact with the called party's service object.

CONCLUSIONS

VoIP networks have come of age, with significant investment from key industry players. New network and communications technologies are dramatically changing the way services are deployed. There is now a critical need for services that span heterogeneous networks, and VoIP, with its extreme openness, will be a key factor in enabling the communication control convergence. In the VoIP context, using this type of active networking through objects downloadable into gatekeeper-like servers seems to be a reasonable way to address the problem.

The architecture presented in this article is a first step in this direction, and more work in this area is expected in the future. Users, carriers, and equipment vendors must map future service requirements and opportunities into an architecture from which protocols, technologies, and products will be determined. The transition from a closed, circuit-switched, usually monopoly- (or oligopoly-) owned telephone network to a much more open and user-configurable packet network poses many difficulties.

Our experience shows that integration between the control elements of traditional and emerging networks is a powerful tool for rapidly deploying and offering new services. We believe that approaches like ours are suitable for the so-called next-generation service providers and will replace the existing closed solutions. ■

REFERENCES

1. H. Schulzrinne, "Re-engineering the Telephone System," *Proc. IEEE Singapore Int'l Conf. Networks*, Singapore, Apr. 1997.
2. J. Garrahan et al., "Intelligent Network Overview," *IEEE Comm.*, Vol. 31, No. 3, Mar. 1993.
3. ITU-T Rec. H.323, "Visual Telephone Systems and Equipment for Local Area Networks which Provide a Nonguaranteed Quality of Service," Geneva, Switzerland.
4. D. Isenberg, "The Dawn of the Stupid Network," *ACM netWorker*, Vol. 2, No. 1, Feb.–Mar. 1998, pp. 24–31.
5. C. Low, "Integrating Communication Services," *IEEE Comm.*, Vol. 35, No. 6, June 1997.
6. O. Kahane and S. Petrack, "Call Management Agent System Requirements Function Architecture and Protocol," IMTC VoIP Forum, Seattle, Wash., Jan. 1997; work in progress: ftp://ftp.imtc-files.org/imtc-site/VoIP-AG/VoIP97-010.doc.
7. M. Handley, H. Schulzrinne, and E. Schooler, "SIP: Session Initiation Protocol," Internet draft, IETF, Jan. 1999; work in progress.
8. "Telecommunications and Internet Protocol Harmonization Over Networks," TIPHON, http://www.etsi.org/tiphon.
9. O. Mizuno et al., "Advanced Intelligent Network and the Internet Combination Service and Its Customization," *IEICE Trans. Comm.*, Vol. E81B, No. 8, Aug. 1998.
10. J. Lennox and H. Schulzrinne, "Call Processing Language Requirements," Internet draft, Internet Engineering Task Force, July 1998; work in progress.

Daniele Rizzetto is a member of the technical staff at Hewlett-Packard Laboratories, Bristol, UK. He graduated in computer science from the University of Padua, Italy, in 1995, and earned a postgraduate degree in network systems from Cefriel-Politecnico di Milano, Italy, in 1997. He is interested in the impact of Internet telephony on traditional telephony and next-generation service platforms for telecommunications.

Claudio Catania is a senior member of the technical staff at Hewlett-Packard Laboratories, where he focuses on the impact of disruptive technologies on advanced communication services. He received the Laurea degree in electronic engineering from Politecnico di Milano, Italy, in 1995, and a post-graduate specialization degree in information technology from Cefriel (Politecnico di Milano), where he studied mathematical models for performance evaluation of interconnection networks.

Readers can contact Rizzetto at daniele_rizzetto@hp.com and Catania at cld@hpl.hp.com.