A Programmable Architecture for the Provision of Hybrid Services

Constant Gbaguidi, Jean-Pierre Hubaux, Maher Hamdi
Institute for computer Communications and Applications (ICA)
Swiss Federal Institute of Technology (EPFL)
1015 Lausanne, Switzerland
{constant.gbaguidi, jean-pierre.hubaux, maher.hamdi}@epfl.ch

and Asser N. Tantawi
IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, NY 10598, USA
tantawi@us.ibm.com

Abstract

The success of new service provision platforms will largely depend on their ability to blend with existing technologies. The advent of Internet telephony, although impressive, is unlikely to make telephone customers suddenly turn in favor of computers. Rather, customers display increasing interest in services that span multiple networks (especially Internet Protocol-based networks and the telephone and cellular networks) and open new vistas. We refer to these services as *hybrid services* and propose an architecture for their provision. This architecture allows for programming the service platform elements (i.e., network nodes, gateways, control servers, and terminals) in order to include new service logics. We identify components that can be assembled to build these logics by considering a service as a composition of features such as address translation, security, call control, connectivity, charging and user interaction. Generic service components are derived from the modeling of these features. We assure that our proposal can be implemented even in existing systems in return for slight changes: These systems are required to generate an event when a special service is encountered. The treatment of this event is handled by an object at a *Java Service Layer*. Java has been chosen for its platform-neutrality feature and its embedded security mechanisms. Using our architecture, we design a hybrid closed user group service.

1. Introduction

Until very recently, communication services have been largely tied to specific network technologies. Voice services have been mainly carried out by General Switched Telephone Networks (GSTN) and cellular networks. Likewise, the Internet and technologies such as X.25 have been designed to support data services. However, the barriers that appeared over time among network technologies have been progressively eliminated. The GSTN is used to access the Internet and the Internet is being provided with telephony capabilities. The old philosophy, consisting in inventing new technologies for new services, should now be replaced by a novel approach that addresses the provision of services spanning multiple, existing, as well as forthcoming, network technologies. We refer to such services as *hybrid services*. We should avoid making these services blind re-implementations of existing services. Rather, they will provide seamless access to services that reside in any network, carry new vistas and empower customers with advanced control and management capabilities. Checking an electronic mail box or doing electronic shopping from any telephone set will become common practice, as well as receiving missed calls on a unique voice mail box, regardless if the calls were addressed to a computer or a telephone. The success of novel services, including telephony over the Internet Protocol (IP), will largely depend on their ability to blend together with other networks. Telephone customers, almost one billion around the world, will probably keep their old equipment if IP telephony does not carry any new vistas.

The credibility of any architecture for the provision of hybrid services rests partly in the way that this architecture maps to the existing service infrastructures. Ultimately, such an architecture must be deployable on any network in return for minimal adaptations. This simply expressed requirement is difficult to fulfill because existing service infrastructures use different philosophies: For the Internet, it is a common understanding that services are implemented in end-systems (*end-system-centric vision*). On the other hand, services on the GSTN are almost entirely implemented within the network (*network-centric vision*). Engineers in either community (Internet or Telecommunications) generally display little knowledge of the other. Arming them with an architecture that they can easily understand and use is the challenging issue that we address in this article. Our solution promotes a *service-centric* vision that treats end-systems and network equipment equally: They are all service platform elements that can be tuned to meet the requirements of each specific service. This tuning is called *programmability.* The provision of a service is the result of interactions among service components that reside on service platform elements, especially user terminals, network nodes, control nodes and gateways between differing network technologies. Service components on each platform element dictate the behavior expected from this element. Several problems then arise:

- How to determine a sufficient set of components from which a wealth of services can be built?
- How to map these components onto the service platform elements?
- How much could existing service platforms enjoy the features of our approach?

We propose answers to these questions. Unlike many architecture proposals, which do not show any smooth migration path from the currently used infrastructures to more “futuristic” ones, our approach incorporates the existing systems, while remaining evolutionary.

The article is organized as follows:

- We provide background information for the understanding of the article.
- We present the principles that underlie our service architecture, as well as an overview of it. More importantly, we discuss how current systems can take advantage of our architecture.
- We specify generic service components that can be put together to create sophisticated services.
- We design a hybrid closed user group service as a case study.
- We discuss some related projects.
- We give concluding remarks and perspectives.

2. Background

2.1 Interoperability across GSTN and Internet

In the past two years, interoperation of the GSTN with IP networks have steered many proposals from both the Internet and Telecommunications communities. From the Internet community, we mention the Media Gateway Control Protocol [1] and a signaling gateway transport architecture [2]. The Telecommunications community released a series of recommendations under the umbrella of H.323 [3,4]. We use this series later in our case study. The primary reason for H.323 is to enable audiovisual interactions among terminals situated in various networks. Key H.323 components are the gateway and the gatekeeper. An H.323 gateway translates call signaling, control messages and multiplexing techniques across the network technologies involved in the call. An H.323 gatekeeper performs network administration, bandwidth negotiation and address translation. The gatekeeper is not required in an H.323 system. However, its presence empowers the network administrator with enhanced control on the network. The gatekeeper is intended to play the role of a call center, where services are created and controlled. A generic model has been proposed for the creation of supplementary services within H.323 systems [5].

2.2 Intelligent Network

The Intelligent Network (IN) [6] is the main telecommunication architecture for service creation and control. Its design is based on a simple principle: separation of service-specific software from basic call processing. Prior to the advent of the IN, each switch manufacturer had his own recipe for incorporating services into his switches. Therefore, introducing new services required modifying the software in each switch in the network. Such a process could take years to complete. Network operators were heavily dependent on their equipment suppliers. The IN eliminated much of this dependency by moving service-specific software into a specialized node called Service Control Point (SCP). Basic call processing is performed in the switches, now called Service Switching Points (SSP). The advent of the IN reduced the time needed for introducing a new service from years down to just a few months. The IN has gained status acceptance due to the multitude of services that it supports and its application to cellular networks. Therefore, it is an inspiring concept for the provision of future services.

3. Architecture: Principles and Overview

3.1 Principles

The service-centric approach developed in this article is built on a fourfold principle. First, it promotes the *programmability* of the service infrastructure whose elements, including the network, can be tuned to meet the specific demands of a service. Second, our proposal considers *smooth migration* from existing infrastructures to more “futuristic” ones. We assure that even existing systems can enjoy some of the features of our proposal in return for very little changes to their current operation. Third, our approach strongly promotes putting services under the *customer’s control*. Creating and managing services should be as easy as writing a Web page. Customers now wish to gain as much control over telecommunication services as over their Web pages, while using the same tools. In order to satisfy this demand, platform-independent tools, such as the Java language [7], are needed. Java is a platform-neutral language; a program written in Java is portable as is onto any platform that runs a Java Virtual Machine (JVM). The JVM translates the program into platform-specific code. As a fourth principle, any service architecture should preserve the *integrity* of the underlying equipment. The service components must incorporate security mechanisms that assure a trusted service execution environment.

3.2 Java and Extension Packages

Java is a simple, object-oriented, distributed, secure, architecture-neutral, and multithreaded language. It is enhanced with several packages, most importantly: Java Remote Method Invocation (RMI) for interactions between distant objects and JavaBeans for component re-use in a variety of environments. The purpose for JavaBeans is to reach a high level of re-usability and portability. For instance, a user might want to include a Java object (bean) into a Word or Visual Basic or ClarisWorks document. Incorporation of objects in a document is done today in a platform-dependent manner: Object Linking and Embedding (OLE) technology is used on Microsoft platforms while UNIX-based systems employ their own techniques. Java could be a unifying technology, provided the design and implementation of classes follow conventions that facilitate their manipulation on any platform. JavaBeans fulfills this requirement.

3.3 Architecture Overview

Fig. 1 illustrates a high level view of the architecture proposed. The customer builds her service out of an archive of Java beans. She then sends the created service to the service provider. The sequence of beans put together by the customer is later introduced into a service factory and a new service instance is created. We refer to the beans that are comprised by the service instance as *service components*. The service instance uploads the components into the controllers of the resources that will be involved in the provision of the service. The controllers know how to interact with the underlying resources. Whenever possible and efficient, service components are uploaded directly into the resources. For instance, an applet can be sent to the user equipment to collect information. The entity that controls this uploading is the controller of the equipment. In other cases (e.g., when the resource does not host a JVM), the component resides within the resource controller. The controller then communicates with the corresponding resource by using an appropriate protocol.

In Fig. 1, each service is assigned with a discriminator (e.g., a prefix such as 1-800). To take advantage of our architecture, an existing system is required to detect these discriminators. When a special service is detected, the system passes the request to a dedicated server. This model applies equally for H.323 gatekeepers, IN SSPs and SCPs. This discrimination mechanism relieves existing systems from handling every detail of the services. This mechanism is similar to the one used in the IN.

We are concerned with two main issues in our architecture: (1) the determination of a sufficient service components archive from which a multitude of services can be built, and (2) the mapping of these components onto the resources where they will be run (communication among the service instance, resource controllers and the resources). Both issues are addressed below. The case study will provide a better understanding, especially of the mapping issue.

As indicated above, components are uploaded into the service platform elements. A major issue with such a model is security. This justifies our use of the Java language, which offers many security guarantees. The Java security model builds on the concept of *Sandbox* [8]. A sandbox defines a domain inside of which a Java program is allowed to execute. By default, a program is not allowed to cross the boundaries of its sandbox. The building and management of the sandbox rest mainly on three Java components: the class loader, the byte-code verifier, and the security manager. The class loader fetches code (Java applets) from a remote machine, creates and enforces a namespace, and prevents applets from invoking system-level methods (e.g., to access the machine resources). The byte-code verifier inspects the binary code and ascertains that the code observes all the rules of the language (e.g., conformance to the language specification, correct use of pointers, legal data conversion, and conformance to resource access rights). The security manager polices the namespaces, as well as the boundaries between the sandboxes. Despite all the security mechanisms built in Java, there are still a number of flaws that might be exploited by an attacker [9]. Many of these flaws are due to choices made when implementing Java specifications. Other attacks exploit flaws, not in Java, but in applications written in other languages.

4. Generic Service Components

As mentioned above, the introduction of new services should be smooth and require little upgrading of the existing service infrastructure. We achieve this by using service discriminators, as proposed above. In engineering terms, the service equipment detects the discriminator of a special service and then sends an event toward a dedicated server. We call the service discriminator the *trigger* of the event. When a trigger is encountered by the service platform element (Fig. 2), an event is fired toward the Java Service Layer which holds (at least) one listener for the event fired [10]. The element that fired the event acts as a trigger event source. An event listener catches and handles the event fired. It is entitled to handle a specific event if its attributes match those of the event. Event listeners are implemented in resource controllers. They hold the logic of the service requested. This logic can require a connection to be set up between two or more endpoints. If so, the trigger event listener fires a connection attempt event (`CnxAttemptEvent`) toward the connection factory. This factory then creates the connection. The result of the creation (`CnxCompleteEvent` if success, or `CnxFailEvent` otherwise) is sent back as an event to the trigger event listener, which keeps the state of the call. In some cases, the connection consists of two "independent" segments that need to be bound by the connection factory. Suppose that a request for a special service arrives from the GSTN at a gateway to an IP network. The gateway issues a trigger event toward a listener, which runs the corresponding logic (e.g., address translation) and initiates a connection to the call destination. After this connection has been established, the two connection segments (one on the GSTN and the second one on an IP network) must be bound together. Practically, this means the association of an input port to an output port within the gateway. Note that this part of the model might be useless if the gateway were configured in such a way that the binding is done automatically. This is the case when the gateway is used in conjunction with a gatekeeper in an H.323 configuration. In a more general situation, the gateway can be connected to a call center that uses our model.

The term "Connection" here may sound awkward, especially for people of the Internet community who generally prefer using "Connectivity" instead. The founding principle of a connection is the establishment of a pipe that conveys traffic between two points. Such a notion does not exist in the Internet yet, as packets sent from a source to a destination are likely to follow very different paths. We define a connection as a virtual binding between two devices, regardless of the paths followed in between. Note that the notion of connection in IP networks could appear in the future when resource reservation strategies, such as integrated services or differentiated services, are implemented. In either case (presence or absence of reservation strategies), our definition of connection is valid.

The above description essentially addresses the creation of the session topology (connection establishment). The logic implemented by the trigger event listener is still unspecified. We gather all logic concerns under the heading of service modeling.

4.1 Service Modeling

We model a service as a composition of features; examples of features are address translation (for call routing services such as call screening or call forwarding), call control (call-on-hold, admission or removal of a new medium or user), security (authentication, confidentiality, data integrity), user interaction, charging, and connectivity. We model each feature using the Unified Modeling Language (UML) [11], which is a technique for representing the object classes of a system as well as the relationships among them. UML uses both graphical and textual notations to describe classes. For the sake of conciseness, we use the graphical notations hereafter, in lieu of Java class descriptions. Every class or relationship is mapped to a Java class by adding its properties (called attributes) and methods (i.e., operations that can be invoked on instances of the class).

The customer defines a service (Fig. 3) and allows users to access this service by setting appropriate permissions. A service offers a number of capabilities (e.g., capabilities of electronic banking are funds transfer, account lookup, and funds withdrawal) and is composed of features. A feature can interfere with another feature in a troublesome relation called *feature interaction,* which is not addressed here. Moreover, a feature can involve other features. For instance, some call control operations, such as admission of new users, involve the connectivity feature. For the sake of brevity, we model only address translation.

4.2 Address translation

Considering that an end-system can embed several devices (e.g., many network interface cards), we associate an address with each device rather than the end-system (Fig. 4). End-system is a generic concept that embraces all equipment, other than network elements, that is involved in communication services. Examples of end-systems are user terminals (e.g., telephones and computers), gateways, gatekeepers, information transcoding equipment, and multipoint control units. A network element implements only the lower three layers of the Open Systems Interconnection (OSI) model.

The term “device” is used here in its broader sense: Any entity that holds and/or processes information. It might be any software piece, or hardware board, or a combination of both. Examples are storages, processors, protocol suites, networking devices, information converters, and smart cards. The user employs devices to communicate. She can also have an address (e.g., Email address) that basically does not relate to the device that she is using.

An important class in Fig. 4 is Address Binding which is used to bind Address objects with one another. This class is defined and managed by the user or an authorized individual. It can be used to provide services such as Abbreviated Dialing, Call Screening and Call Forwarding. For Abbreviated Dialing, an address is bound to another one with fewer digits. Screening a call means associating it with a null address.

5. Case Study: A Closed User Group Service

5.1 Service Definition

We consider a hybrid Closed User Group (CUG) service that gathers users on both a telephone and an IP network. Both networks are controlled by the same operator. A CUG is a group of users participating in a *private* session. The privacy of the session stems from the prevention of outside users from interfering with the CUG users. Users within a CUG can be involved in many application sessions, such as telephone conversations, chats, videoconferences, shared whiteboards and distributed games. When creating a CUG, the customer specifies the CUG members, as well as the applications that they can use. The customer can elect a CUG manager who is responsible for setting up application sessions, admitting and removing users.

The CUG, as defined here, is different from its perception in both the Telecommunications and Internet communities. Telephony is the main service included in an IN CUG, whereas the Internet community defines richer services such as computer videoconference, chat, and games. Our CUG definition goes beyond either perception, as we allow a GSTN user to participate in sessions with Internet aficionados. A typical scenario is a user, having both a computer and a telephone, who wishes to talk on her telephone, while participating in a whiteboard-sharing session (on an IP network).

5.2 The CUG Service as a Hybrid Service

The CUG service is hybrid in two main respects:

- hybridity of the communication infrastructure: The service can be accessed equally from an IP network or the GSTN, even though GSTN users may not enjoy applications such as whiteboard-sharing. This hybridity permits three kinds of interactions: Internet-only sessions, GSTN-only sessions, and cross-network sessions
- uniqueness of the service creation and control platform: The implemented service can be plugged into either the GSTN or an IP network after a slight tuning of its behavior. Moreover, the control of the service is entirely done at the place where the service implementation resides.

5.3 System Layout

The CUG service is based on H.323 on the IP network and on IN on the GSTN (Fig. 5). A CUG Server (CUGS) is in charge of creating private sessions. The main elements in the infrastructure are supervised by their respective controllers.

The creation and management of CUGs are supported by the combined entity made up of the CUG server and its controller. The creation of a group consists in defining a group chair, the members and the set of applications that can be shared. The management of a group consists in dynamically adding to, or removing members and applications from, an ongoing session. The CUGS controller implements the logic of the CUG service and the CUGS provides applications to the users.

5.4 Creation of a CUG

The creation of a CUG can be initiated from either an IP network or the GSTN. Many types of terminals can be connected to the GSTN, including a telephone, an ISDN terminal and a mobile phone. When the creation request comes from a (-n ISDN or IP) terminal, the CUGS controller instructs the CUGS to upload a User Interaction component, in Java applet form, into the computer. This applet, called `ipCugClient`, collects necessary information from the user and sends this information to the controller, which checks the privileges and/or restrictions applicable to the user. If the user is authorized to access the service, the CUGS controller instructs the CUG server to upload another applet, called `ipCugMember`, to the user's computer. This applet corresponds to the user's profile (manager or simple user, as well as available applications) and handles all interactions between the user and the CUG server. It incorporates security mechanisms such as a security protocol, as well as the software to be used for handling the user data.

A more complex scenario is the processing of a CUG creation request initiated from a telephone. The elements of the CUG service system interact as depicted by the event trace in Fig. 5, which is detailed in Fig. 6. Terminal B dials a special number (e.g., an 800 number) that is detected by the SSP (operation 2 in Fig. 5). A request is sent by the SSP to the SCP (operation 3) in order to get instructions on the processing of the call. The SCP controller looks up an internal database and finds the service component referenced by the number dialed. This component is the CUGS Controller (operation 4). The CUGS Controller informs the SCP controller that a connection has to be established between Terminal B and the CUGS. The address of this server is given back to the SCP. Simultaneously, the CUGS controller notifies the server of an upcoming call from Terminal B (operation 5). The notification carries an archive of components, especially an applet named `gstnCugClient`. This applet will reside on the CUGS and interact with Terminal B using Dual Tone Multi-Frequency (DTMF) signaling. Operations 6, 7 and 8 (Fig. 5) establish the connection between Terminal B and the CUGS. The applet `gstnCugClient` runs the same dialog as the above-mentioned `ipCugClient,` except that it can interpret DTMF signaling and possibly voice messages. User B can then create a CUG or join an ongoing session.

It is worth mentioning that the CUG service can be implemented separately on the GSTN and the IP network. An IP-based prototype has been developed, in our Institute, in the frame of the CLosed User group Environment (CLUE) project [12]. The advantage of using the Java Service Layer is the uniqueness of the service implementation, which facilitates its control and management and reduces the time to market.

6. Discussion

Programmability of the service infrastructure has been an interesting research topic since the early 1990's [13]. Its goal was to accelerate service creation, and unify the control and management of advanced multimedia services. Examples of early programmability projects are the Telecommunications Information Networking Architecture (TINA) [14] and the Extended Binding Model [15]. While TINA adopted a rather network-centric philosophy, XBIND rightly considered the network as only one of the elements involved in service provisioning. This element should be as programmable as end-systems. Asynchronous Transfer Mode (ATM) was considered as the main transport technology in both proposals. In contrast, our architecture can be deployed on any network technology. Moreover, it incorporates an engineering process that enables the creation and control of service logics, whereas both TINA and XBIND focus on communication topology issues (how to establish and control a communication between endpoints). Such an engineering process is also missing in more recent works on the interworking between the IN and the Internet [16,17]. Nevertheless, these interworking proposals can be used to reduce significantly the time necessary for the introduction of new services. Lately, a programmable architecture, named Geoplex [18], was proposed with similar objectives as ours. However, it is too early to make any thorough comparison between the two projects.

Other programmability projects, such as Computer-Telephony Integration (CTI) [19], are targeted for private switched networks. They enable a computer to send messages to a Private Branch eXchange (PBX) in order to open a connection between any two terminals (especially a computer and a telephone). Our architecture clearly extends the CTI scheme to backbone networks; a call center that uses our architecture can support a richer set of services, as illustrated in the case study above.

Over time, programmability has evolved to a newer concept, called active networking [13], that goes a step further: Instead of using statically defined interfaces, the customer is allowed to dynamically (i.e., during a call session) instruct the network on the behavior that she expects from it. We believe that injecting program codes in network nodes on a user - or even packet - basis is likely to undermine the efficiency of the network. Hence, we take a more pragmatic approach in this article: Code uploading is performed prior to running the service rather than dynamically. This uploading is controlled by the service provider's call center.

7. Conclusion

We have proposed a programmable architecture that addresses the creation of hybrid services. We have adopted a service-centric vision, following which, service components should execute at any place that delivers better performance and security. This vision considers the service infrastructure as composed of elements, each of which holds a part of the global behavior (or logic) expected from the service. In this way, our architecture can be implemented on top of multiple heterogeneous networks.

The originality of our proposal lies in two achievements. *First*, we have proposed a way of identifying components that can be used to implement a multitude of service logics.The technique used has consisted in considering a service as a composition of features (e.g., address translation, call control, connectivity, and security). Modeling each feature helped us identify a significant set of components. Most existing programmable architectures concentrate on defining interfaces for a subset of service platform elements, without any mapping with the globally expected functionality. In contrast, we propose a way of mapping this global service functionality onto the platform elements. *Second*, we ensure that our proposal blends with currently used systems, while most programmable architectures assume systems with a single technology such as ATM. Existing systems can enjoy some of the features exposed by our architecture, by only being able to detect special service discriminators called triggers. When a trigger is detected, a request is issued to a Java Service Layer where the logic of the service is applied.

We are testing our proposal with services such as Voice-access-to-content, hybrid voice mail, and closed user groups. Our testbed is composed of a PBX, an H.323 gateway and a gatekeeper. We intend to address, in the near future, real-time and mobility services. We expect the implementation of these services to give us much insight so that we can address issues such as performance management. We will also refine the mapping of our architecture onto existing systems by studying interoperation between the IN and H.323 systems.

References

[1] M. Arango *et al.*, Media Gateway Control Protocol (MGCP), Internet Draft <draft-huitema-MGCP-v0r1-01.txt>, Nov. 1998.

[2] N. Glaude, T. Blain and R. Cable, SS7 to IP Signaling Gateway Transport Architecture, Internet Draft <draft-glaude-ss7-gw-00.txt>, Nov. 1998.

[3] ITU-T, *Packet-based Multimedia Communications Systems,* Rec. H.323, 1998, Geneva, Switzerland.

[4] M. Hamdi, O. Verscheure, J. P. Hubaux, I. Dalgiç and P. Wang, Voice Service Interworking for PSTN and IP Networks, To appear in *IEEE Comm. Mag.*, 2nd Quarter of 1999.

[5] ITU-T, *Generic Functional Protocol for the Support of Supplementary Services in H.323,* Rec. H.450.1, Feb. 1998.

[6] ITU-T, *Introduction to Intelligent Network Capability Set 1*, Rec. Q.1211, March 1993.

[7] http://java.sun.com .

[8] L. Gong, M. Mueller, H. Prafullchandra and R. Schemers, Going beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2, In Proc. of the *USENIX Symp. on Internet Technologies and Systems*, Monterey, California, Dec. 1997.

[9] D. Dean, E. W. Felten, D. S. Wallach and D. Balfanz, Java Security: Browsers and Beyond, Technical Report 566-97, Princeton University, Available from http://ftp.cs.princeton.edu/sip/pub/internet-beseiged.html , Feb. 1997.

[10] C. Gbaguidi, J. P. Hubaux, G. Pacifici and A. N. Tantawi, An Architecture for the Integration of Internet and Telecommunication Services, To appear in *IEEE Journal on Selected Areas in Communications*, Special Issue on Service Enabling Technologies for Networked Multimedia Systems, Aug. 1999.

[11] M. Fowler and K. Scott, *UML Distilled - Applying the Standard Object Modeling Language*, (Addison Wesley, 1997).

[12] S. Staamann, L. Buttyan, A. Coignet, E. Ruggiano, U. Wilhelm and M. Zweiacker, Closed User Groups in Internet Service Centres, *2nd IFIP Intl. Conf. on Distributed Applications and Interoperable Systems (DAIS)*, June 1999.

[13] T. M. Chen and A. Jackson (eds.), *IEEE Network Magazine*, Special Issue on Active and Programmable Networks, vol. 12, no. 3, May/June 1998.

[14] http://www.tinac.com .

[15] A. A. Lazar, K. S. Lim and F. Marconcini, Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture, *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, Sept. 1996, pp. 1214-27.

[16] I. Faynberg, M. Krishnaswamy and H. Lu, A Proposal for Internet and Public Switched Telephone Networks (PSTN) Internetworking, Internet Draft, March 1997.

[17] C. Low, Integrating Communication Services, *IEEE Comm. Mag.*, vol. 35, no. 6, June 1997.

[18] http://www.geoplex.com .

[19] C. R. Strathmeyer (ed.), *IEEE Comm. Mag.,* Special issue on Computer Telephony, vol. 34, no. 4, April 1996.

Biographies

Constant Gbaguidi received an M.S. degree in electrical engineering from EPFL, Lausanne, Switzerland, in 1995. Since then, he has worked at EPFL as a research and teaching assistant in telecommunication services and management. His diploma project was on an implementation of a Virtual Private Network (VPN) over the OSF's Distributed Computing Environment (DCE). In the summer of 1996, he was a visiting scholar at Columbia University, New York, where he worked on porting a network prototyping software to CORBA. Currently, his interests are in the integration of telecommunication and Internet-based services.

Jean-Pierre Hubaux, for a photograph and biography, see the guest editorial.

Maher Hamdi was born in Mahdia, Tunisia, in 1969. He received the Diploma of Designer Engineer in Computer Sciences from the Ecole Nationale des Sciences de l'Informatique of Tunis with the Presidential Award, in April 1992. He obtained the Ph.D. degree from the University of Rennes I in November 1996. His doctoral research was carried at the Networks and Multimedia Department of ENST de Bretagne (France) and was concerned with video traffic control in ATM networks. He was working as a senior researcher at the same department. His work is concerned with the design and evaluation of interactive services on packet networks. Since August 97, he is working as a

Senior Researcher at the ICA (Swiss Federal Institute of Technology EPFL, Lausanne, Switzerland). He is currently working on New Approaches for Service Engineering in Hybrid IP/Telecom Networks.

Asser N. Tantawi received the B.S. and M.S. degrees from Alexandria University, Egypt, and the Ph.D. degree from Rutgers University in 1975, 1978, and 1982, respectively, all in computer science. He joined the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, in 1982, as a Research Staff Member in the Computer Sciences Department. Dr. Tantawi has published numerous articles in books, journals, and international conferences. His fields of interest include telecommunications and information-network applications, multimedia systems, mobile computing, high-speed networking, and performance modeling and analysis. He received an IBM Outstanding Technical Innovation Award and three IBM Plateau Invention Achievement Awards. Dr. Tantawi is a senior member of the IEEE, a member of the ACM, INFORMS, and IFIP WG 7.3 on Computer Systems Modeling.

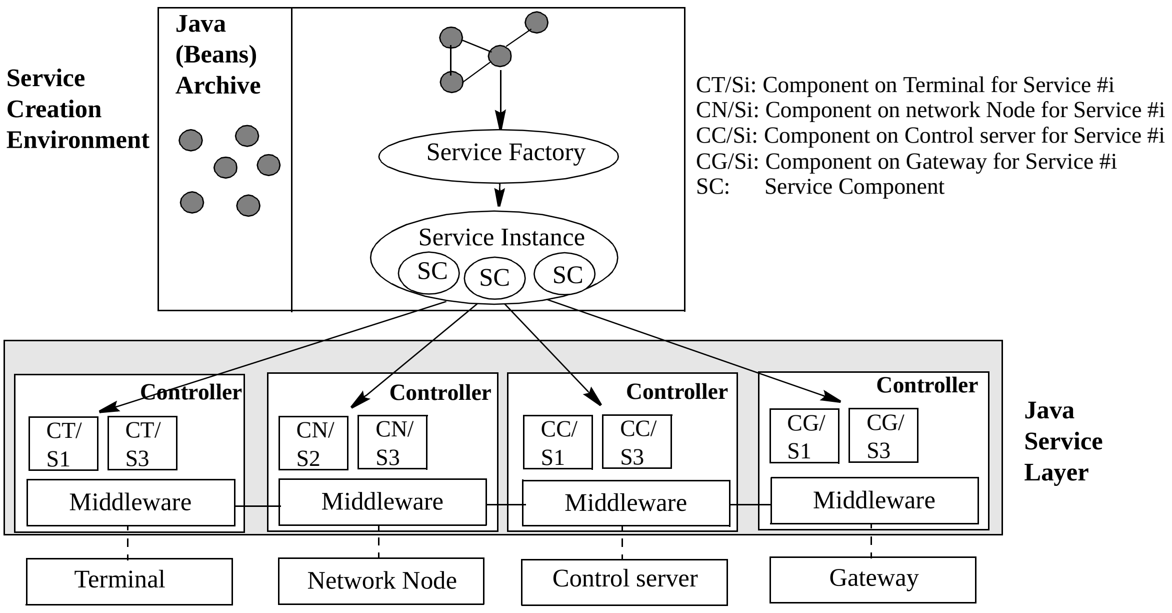

Fig. 1. A programmable architecture for hybrid services.

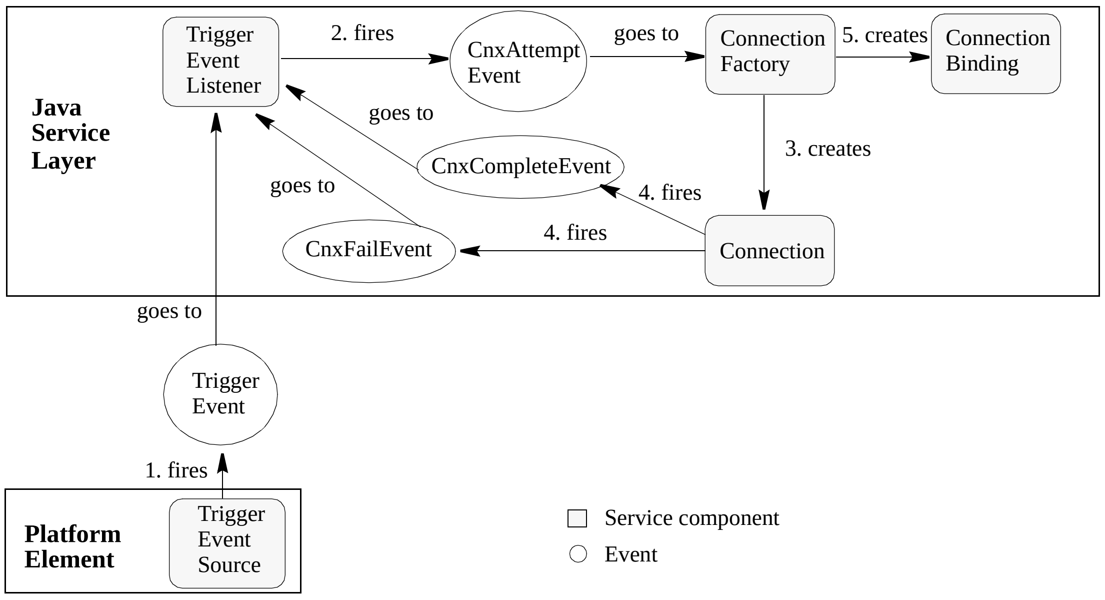

Fig. 2. Trigger-Event-Source and Trigger-Event-Listener model: An information flow diagram.

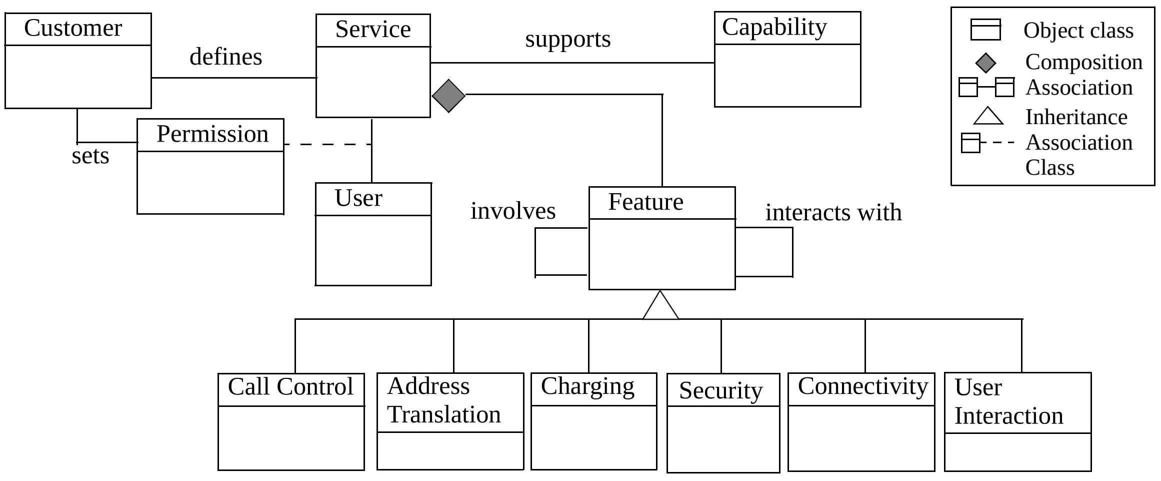

Fig. 3. High-level service modeling.

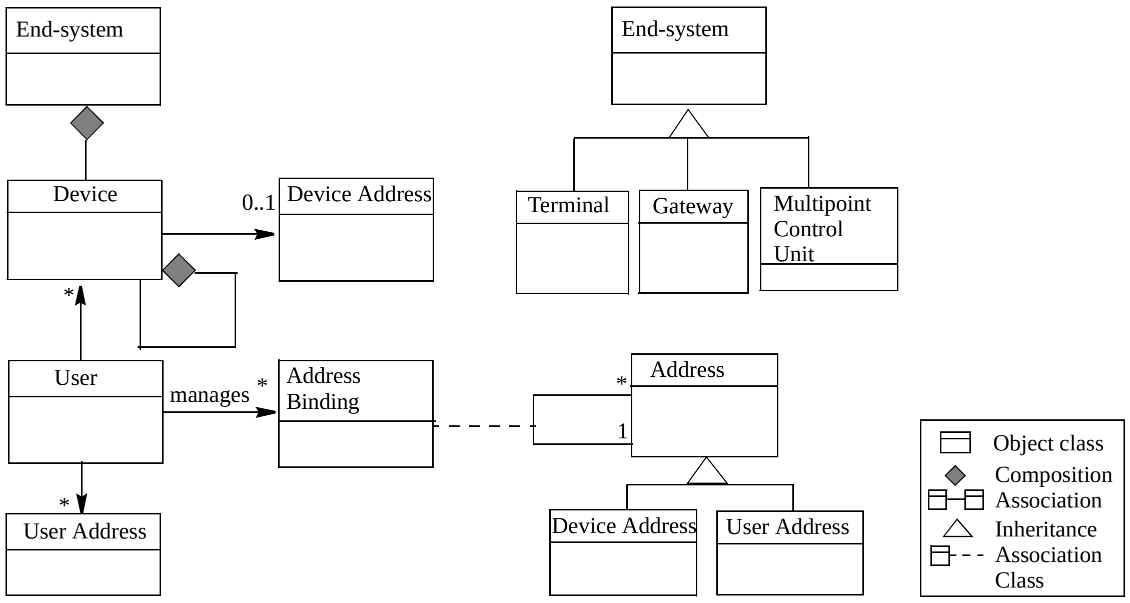

Fig. 4. Modeling of the address translation feature.

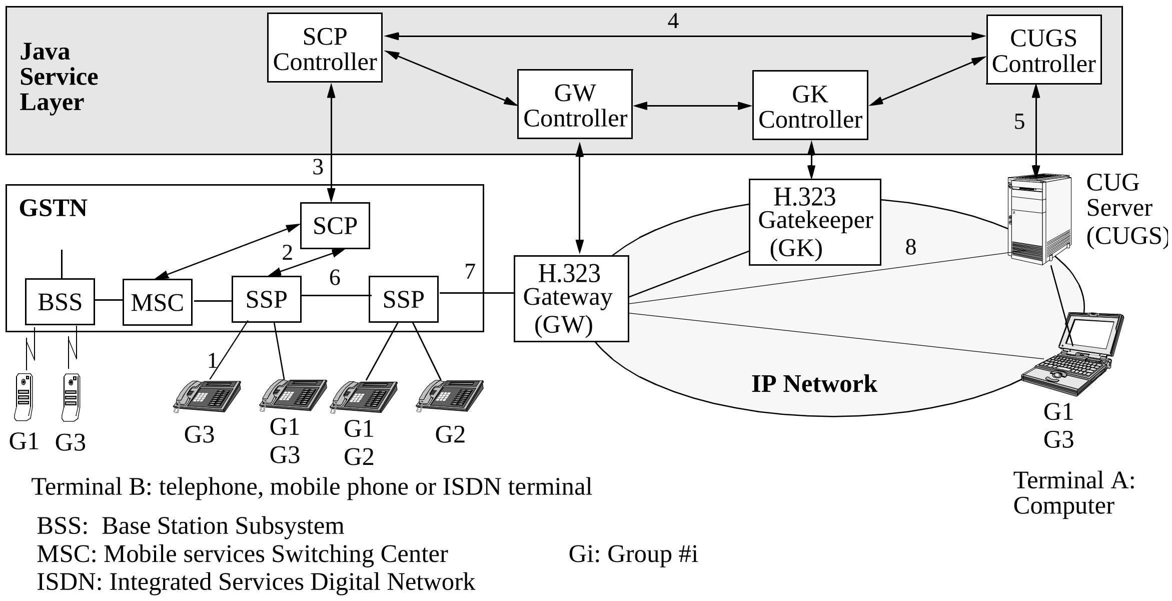

Fig. 5. System layout for a CUG service.

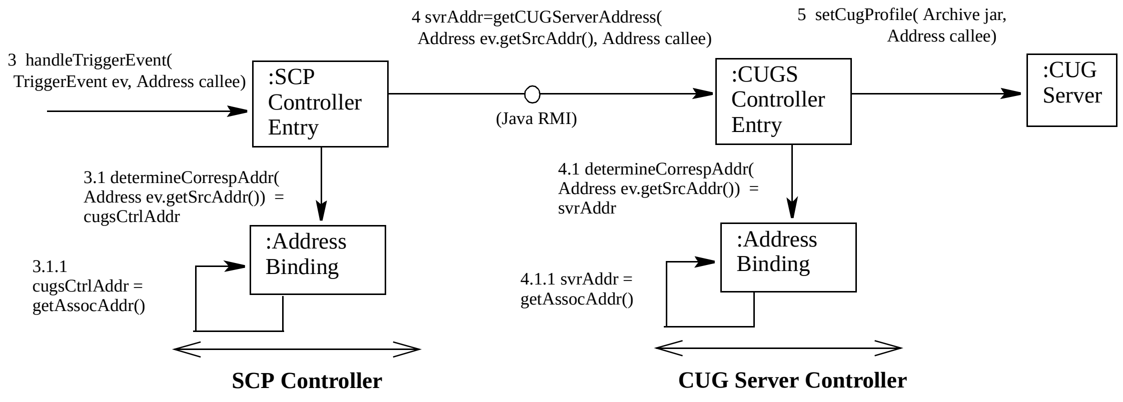
3 handleTriggerEvent(TriggerEvent ev, Address callee)
:SCP Controller Entry
4 svrAddr=getCUGServerAddress(Address ev.getSrcAddr(), Address callee)
(Java RMI)
:CUGS Controller Entry
5 setCugProfile(Archive jar, Address callee)
:CUG Server
3.1 determineCorrespAddr(Address ev.getSrcAddr()) = cugsCtrlAddr
:Address Binding
3.1.1 cugsCtrlAddr = getAssocAddr()
4.1 determineCorrespAddr(Address ev.getSrcAddr()) = svrAddr
:Address Binding
4.1.1 svrAddr = getAssocAddr()
SCP Controller
CUG Server Controller

Fig. 6. Creation of a CUG.