Management of Advanced Services in H.323 Internet Protocol Telephony

Bernard Pagurek, Jingrong Tang, Tony White, Roch Glitho*

Systems and Computer Engineering, Carleton University,
1125 Colonel By Dr., Ottawa, Canada K1S5B6
*Ericsson Research Canada, Montreal, Canada

Abstract-- **The Intelligent Network (IN) represents the world wide accepted basis for uniform provisioning of advanced telecom services. On the other hand IP based communication is fast becoming a viable alternative for voice communications. Mobile agents offer unique opportunities for structuring and implementing open distributed service architectures, facilitated by the dynamic downloading and movement of service code to specific network nodes. In this paper, a new service architecture for IP Telephony, based on the ITU-T standard H.323, is proposed. The implementation uses Mobile Agents and Jini as an enabling technologies and existing architectural concepts taken from IN. This IP service architecture enables telecom services deployed through mobile service agents on a per user basis, which results in several advantages when compared to centralized service architectures. The paper demonstrates that the flexible extensible architecture can accommodate not only existing services but is flexible enough to accommodate a wide variety of future services. In addition we show how the architecture addresses the full management life-cycle of advanced services, from open third party creation, to subscription and utilization, and ultimately to maintenance and withdrawal.**

Index terms-- **IP Telephony, Service Architecture, Mobile Agents H.323**

I. INTRODUCTION

Internet Protocol Telephony[1][2] or IP telephony, which delivers voice and data communications over IP networks[3], became a reality for the first time in 1995 when Vocaltec, Inc. introduced its Internet Phone software[2]. Because of its low price and efficient use of bandwidth, it has progressed rapidly in the relatively short period of time since then. As the Internet is an open, distributed and evolving entity, it is expected that there will be many extensions to IP telephony. In the classical telephony world which is based on circuit switched networks, a number of service architectures have been developed in the last decade. There are for example, the Intelligent Networks (IN) framework[4], Telecommunications Management Network (TMN)[5], the Telecommunications Information Networking Architecture (TINA-C)[6], etc. The purpose of these service architectures is to increase the quality and range of services offered in communication networks. In order to compete with classical telephony in today's market, one of the challenges that IP telephony faces is to offer not only the same high quality voice calls, but also a set of advanced services that are at least at a par with what classical telephony offers today. While the high quality of voice calls has not yet been achieved in the IP telephony world, sound architectures are needed for the control and management of services.

The traditional telephone system has very primitive end terminals (telephones) and considerable intelligence inside the network[7]. Advanced service architectures separate call setup and call processing functions. In general, the Internet represents a different balance, with intelligent end-terminals (computers) and a simple set of functions inside the switches of the network. Switches are composed of software and general-purpose hardware. It is reasonable to foresee that long-term evolution of IP Telephony will have much more intelligence implemented in the end terminals rather than inside the network. Advanced services such as call diversion and call transfer, which are implemented inside the telephone network today, can be implemented in users' computers.

In order to realize this view of IP telephony, appropriate protocols and technologies are needed. Currently, there are two protocols that address this issue, one is the International Telecommunication Union's (ITU-T) H.323[8], and the other is the Internet Engineering Task Force (IETF) Session Initiation Protocol (SIP)[9]. The ITU-T's Study Group 16 approved the first version of the H.323 specification in 1996 and the third version of the standard has been recently released. The standard is broad in scope and includes stand-alone devices and embedded personal computer technologies as well as point-to-point and multi-point conferences. SIP is rather lightweight, reusing many of the header fields, encoding rules, error codes, and authentication mechanisms of HTTP. (See, for example,[10][11] for a comparison and [12] for a critique.) The Service life cycle, defined by the TINA-C service architecture recommendation, consists of service construction, deployment, utilization and withdrawal.

Advanced services supported by H.323 are specified in the H.450.x series[13]. Each of the defined advanced services has it's own specification. As is pointed out in [11], "How they may be broken down into reusable building blocks is not clear and this will lead to specification and implementation inefficiency". The H.323 specification does not address service control and management. In addition, there are no third party defined services. Hence there is obviously room left for developers to design more flexible service architectures using enabling technologies; e.g., Mobile Agents (MA). One of the goals of our research is to design a more open architecture in which services are broadly available, and in which lower level services can cooperate to generate higher level services or can be combined into higher level services.

IP telephony is real-time data communications over IP transport. As the Internet is an open, distributed and evolving entity, flexibility, scalability and robustness are very important issues to be considered when designing new service architectures. MA technology has the ability to provide solutions addressing all of these issues. As is pointed out in [14], the key advantage of MA is flexibility. It can enhance service architectures, providing easy service customization and instant service provisioning.

The remainder of this paper is organized as follows. In Section II, the basic enabling technologies are briefly discussed. Concepts of IN and MA, Jini/JavaBeans, and advantages for using them, are explained. In Section III, a new IP telephony service architecture and aspects of the actual implemented solution using Jini, MA, and IN concepts are described. Section IV discusses proposed mechanisms for service creation from low level components and ongoing service management. In addition how mobile agents can contribute to advanced services is explored further. Section V presents three scenarios for application of this new service architecture - Call Forwarding Unconditional without and then with a gatekeeper, and finally aspects of Virtual Private Network (VPN) Services. Section VI presents the conclusions relating to this research.

II. Enabling Technologies

The Java language has a number of advantages that make it particularly appropriate for Mobile Agent (MA) technology. Its main appeal for agents is its portability. This means that any system with sufficient resources can host Java programs. The Java Virtual Machine and Java's class loading model, coupled with several of Java features – most importantly serialization, remote method invocation, multithreading, and reflection – have made building first-pass mobile agent systems a fairly simple task[15].

MA, as one of the enabling technology for IP telephony services, is introduced in Section II.A. JavaBeans and Jini (both from Sun Microsystems) are two good complementary candidate technologies for the implementation of an IP Telephony service architecture; they are described in Section II.B.

A. Mobile Agent Technology

An agent can be described as a software component that performs a specific task autonomously on behalf of a person or an organization [16]. It contains some level of intelligence, ranging from predefined rules to self-learning Artificial Intelligence (AI) mechanisms. Thus agents may operate rather asynchronously to the user and may communicate with the user, system resources and other agents as required to perform their tasks. They are often event or time triggered.

A Mobile Agent clearly is not bound to the host where it begins execution. It has a unique ability to transport itself from one host in a network to another. As it travels, it performs work on behalf of a network user. Flexibility and extensibility are due to the dynamic nature of the underlying network infrastructure and service demand. Other arguments for mobile agents have also been forthcoming[17].

In the past, the main motivations for the application of mobile agents were the lack of capacity to execute programs locally, and the desire to share resources and improve load balancing in a distributed system. In contrast to these concepts, designed for rather specific or closed environments, new agent concepts aim for open environments (e.g., the Internet). Today, flexibility and extensibility are key design issues for emerging network service architectures in order to permit quick adaptation to changing customer service demands. The following are some of the reasons for using MA technologies:

- A MA-based approach may reduce the network load when compared to an RPC (Remote Procedure Call) – based approach.
- Asynchronous and autonomous execution provide the possibility for realization of advanced services by means of using mobile agents.
- Being independent of the underlying network infrastructure makes the service architecture extendable.
- MAs allow new services to be provided dynamically either by customization or (re) configuration of existing services.
- MAs provide an effective way for deployment and utilization of advanced services within a distributed environment.

The mobile agent paradigm and emerging agent technologies are considered key for implementing open, flexible and scalable services. There are many commercial and nearly commercial agent platforms, such as Grasshopper (IKV++), Voyager (ObjectSpace), and Concordia (Mitsubishi. For this work we chose Grasshopper as the platform, mainly because of its adherence to the OMG MASIF standardization effort.

In April 1997, CLIMATE – The Cluster for Intelligent Mobile Agents for Telecommunication Environments, a pool of projects within the European Union collaborative research and development program on Advanced Communications Technologies and Services (ACTS), was launched to explore the usage of agent technologies. Most of these projects are located within the Service Engineering, Security, and Communications Management domains. CLIMATE is taking an active part in contributing to relevant agent standards (e.g., OMG, FIPA) and telecommunication standards (e.g., IN, TMN, UMTS standardization). Notably, the Grasshopper MA framework has been developed under the CLIMATE umbrella.

B. JavaBeans/Jini

The JavaBeans specification is an object-oriented programming interface that is used to build re-useable application or program building blocks called components that can be deployed in a network or any major operating system. Ideally, any Java component conforming to the JavaBeans component model can be reused in any other JavaBean compliant application. Every Bean not only complies with the JavaBean model, it also carries with it all its properties and methods, which can be easily garnered through introspection – a JavaBean property whereby any Bean-aware tool (e.g., a visual programming tool) can analyze and report on how a Bean operates.

Jini technology[18] takes advantage of the Java language. In Jini everything is a service. It brings to the network facilities for distributed computing, network based services, seamless expansion, reliable smart devices, and easy administration. It provides lookup services (LUSs) and a network bulletin board (or blackboard) for all services on the network. Jini LUSs facilitate a search of services connected by the communication infrastructure and store not only pointers to the service on the network, but also service proxy code, interfaces that enable a user to acquire and execute these services. It is important to note that finding a usable service results initially in the downloading of the proxy code which can then be used to configure and deploy actual services. The components of the Jini system can be segmented into three categories: infrastructure, programming model and services. The infrastructure is the set of components that enables building of a federated Jini system, while the services are the entities within the federation. The programming model comprises interfaces that enable the construction of reliable services.

C. Intelligent Networks

IN services [4][19] are based on additional service logic and data on top of different switched telecommunication networks. Centralized service nodes known as Service Control Points (SCPs) control the telecommunications network via a dedicated out of band signaling network; i.e., the International Signaling System No. 7 (SS7) network. The bearer switching nodes, known as Service Switching Points (SSPs), provide only basic call processing capabilities. IN service deployment and management is realized through a Service Management System (SMS), which interacts with IN elements via a data communication network. Since the SSPs and the SCP have to interact for each IN service call (usually multiple times), the signaling network and the central SCP may become serious bottlenecks. Furthermore, SCP failures would result in global service unavailability.

The IN platform provides greater flexibility for service creation in general and also for the tailoring of services to suit the exact requirements of a particular customer. IN-based services rely on service-independent building blocks (SIBs) these being the smallest units in service creation. SIBs are reusable and can be chained together in various combinations to realize services. They are defined to be independent of the specific service and technology for which or on which they will be realized. SIBs were not traditionally implemented with object oriented concepts and this is one of the reasons for the drive towards JavaBeans for low level component implementation.

With more effort to standardize agent platforms, agent platforms are maturing, and with Java as an enabling tool for implementing MAs, Mobile Agents have brought tremendous opportunities for development of new service architectures for IP telephony. It should be noted that MA architectures for traditional IN services in the PSTN[16] and Internet services[24] have already been proposed. The use of CORBA and Java for multimedia services has also been proposed[20] as has the integration of IN and Internet services [21]. Based on the hypothesis that networks based on the IP protocol will not replace the Public Switched Telephone Network (PSTN) in the short term and that migration in this direction will require hybrid services, Gbaguidi et al.[22] propose that services should be outside the scope of H.323/H.450. In their model services are provided by the vendors of network equipment allowing for easier integration. They also suggest JavaBeans for implementation and focus on several low level components. Our approach in contrast is H.323 oriented and emphasizes a more open environment in which third party services may be offered, found, and may dynamically contribute to higher level services.

III. Advanced Service Architecture

A. Overview

As a consequence of the drawbacks of the existing service architecture defined by H.323 described earlier, we propose a MA-based advanced service architecture for implementing H.323 advanced services using the widely accepted service provisioning basis (IN), and enabling technologies (MA, Jini/JavaBeans)

As is illustrated in Fig. 1, H.323 gatekeepers and H.323 terminals (Users) are connected to the Enterprise LAN. User terminals join one gatekeeper's zone through H.323 gatekeeper discovery and an endpoint registration process. Mobile agent platforms, called Agencies, are introduced into the devices, gatekeepers and terminals, that are connected to the enterprise LAN to provide an agent execution environment. Agencies can vary considerably in their capabilities but as a minimum, they allow for code to be asynchronously pushed on to the node and executed, and they provide for code migration to another target.

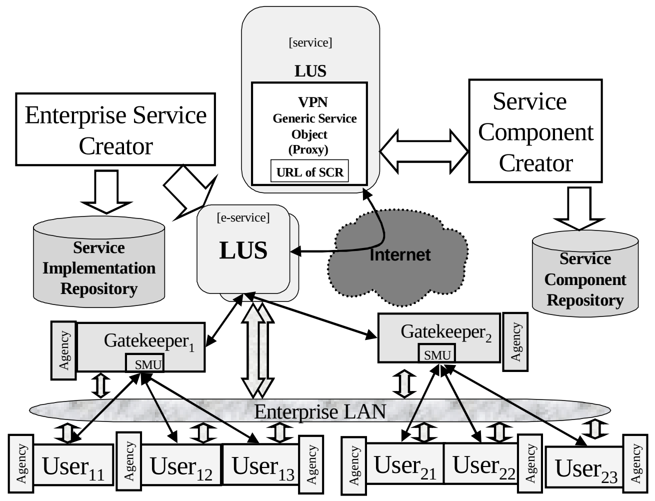


Fig. 1. Mobile Code for IP Telephony

The main idea behind this architecture is to provision H.323 advanced services in a uniform, but very flexible way, supporting dynamic deployment of services. H.323 advanced services are realized by means of mobile service agents. The key to this approach is to deploy service agents to the service users; i.e., the call parties, which makes this service architecture open, distributed and flexible. As we shall see later in more detail, the User Service Agent (USA) is constructed for a user when the user subscribes to one or more advanced services. It is subsequently updated each time the user subscribes to additional services. Its purpose is to record the user's service requirements and to manage services for the user, and it normally resides in the user terminal. Another kind of agent, called the Call Agent (CA) is constructed dynamically at service execution time and is deployed to carry out actual call processing and related activities which may call for mobility. These are discussed more fully in the next two subsections.

Lookup Services (LUSs) are more like blackboards where all the available services' specifications and proxy code (interface of a service not the actual service code) is placed. A Service Component Creator (SCC) is responsible for creating generic service components and advertising its services on a LUS, where all the service components are stored in the Service Component Repository (SCR). An SCC can assemble these service components into new higher level services. End users can not subscribe to services from an SCC directly. Rather, they are made available to an Enterprise Service Creator (ESC) who is responsible for customizing /assembling the service components into services it provides to end users through the code it makes available in a Service Implementation Repository (SIR). The SCC and SCR bring opportunities for third party service creators and providers, making them able to compete in the service market. LUSs can be local or remote. In our implementation they are provided by Jini and are linked by the Internet

This service architecture allows for open service creation. A different Service Component Creator (SCC) using Service Components from a Service Component Repository (SCR) can create advanced services.

A Service Management Unit (SMU) is an entity that can be placed in the gatekeeper, or completely separated from the gatekeeper. It manages service subscription using protocols not defined by H.323; e.g., HTTP. If a service were to be dynamically upgraded, the SMU would be involved. Also, the SMU could be involved in ongoing network management of the services. An SMU can discover an enterprise LUS using a multicast protocol; a unicast protocol is used to discover remote LUSs that are outside of the enterprise LAN. In the latter case, the SMU has to know where the LUS is before it sends out a request to the remote LUS

Service utilization is realized by activating a caller's User Service Agent (USA) and ultimately activating the callee's USA. A Call Agent (CA) will be instantiated by the result of a USA creating a new CA. Using terminology from the Design Patterns community, the USA represents a Factory for Call Agents. The USA has responsibility for service management while the CA embodies the call processing functionality required to set up the call, e.g., it interfaces with an H.323 protocol stack for call setup. In this way we have clear separation of service management and call processing responsibilities.

This architecture supports universal access to the service through the Jini Lookup process. A level of service customization by the end user is also supported by this service architecture. Each user connected to the network can define his or her own services and service data. An example of such data could be the number to which calls might be forwarded in a forwarding service like CFU. Service logic is not embedded in the network nodes but ultimately in the end user's terminal. This makes this service architecture highly distributed.

In summarizing the roles of the participant components introduced above, Jini is used to provide access to service-related code and Service creators generate service components which they advertise by storing them in a Jini lookup service. Enterprise service creators who generate value-added services specific to their own enterprise needs discover these components. These value-added service components in turn are made available to users through the SMU by advertising them in Jini lookup services visible from inside the enterprise network. The SMU also is responsible for maintaining a user profile of subscribed services and USA information for all registered users

In the next two subsections, we will describe in more detail service subscription and utilization with our architecture when only one gatekeeper is involved, and at the same time we will explain the related components' functionality. These scenarios are presented as they represent familiar H.450.x standard services. However, the architecture proposed in this paper is considerably more flexible. Examples of services which take advantage of CA mobility are discussed later in the paper.

There are four phases in the life cycle of a service; they are service creation, subscription, utilization, and ongoing management and withdrawal. We shall begin by discussing the middle two which we implemented using Jini and the Grasshopper mobile agent system.

B. Service Subscription Using Jini

Let us assume for illustrative purposes that the service we are dealing with is a Virtual Private Network or VPN. A VPN is a private enterprise telephone network established by the enterprise administrator. Such a VPN can be used, for example, to control who has access to VPN services such as long-distance calling etc. A VPN can be thought of as a service comprising feature sets that have been customized for a particular enterprise. More specifically the Outgoing Call Allowance (OCA) and Outgoing Call Restriction (OGR) VPN features allow an enterprise administrator to restrict the numbers that may be called from within an enterprise. For example, an easily accessible terminal might be restricted to numbers within the enterprise whereas a terminal on an employee's desk might have unrestricted dialing privileges. For these features access to a Restricted Destination Database is needed. Another VPN feature, an extended hunt group, is discussed later in section IV.

Referring again to Fig. 1 and then Fig. 2, service subscription consists of four major steps. The steps are:

1. The Service Component Creator advertises a generic VPN service proxy object using the Service LUS.
2. The Enterprise Service Creator discovers the Service LUS, downloads the VPN service proxy object to its machine, and uses its graphical User Interface to customize the VPN service for its enterprise users.
3. The Enterprise Service Creator uploads its customized service proxy object to the e-Service LUS. This completes the preparatory steps. The service proxy will now be available for downloading to the Service Management Unit in the next step whenever needed.
4. Finally, any User can send a request to its SMU to subscribe to services as shown by the top arrow in Fig. 2. which expands the actual form filling interaction

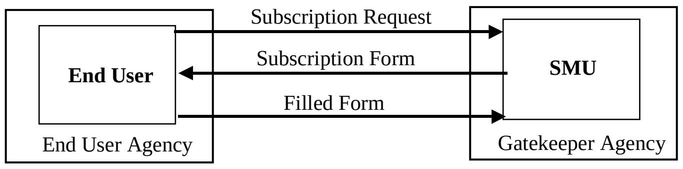


Fig. 2. Service Subscription

There is some degree of activity resulting from a subscription request. After the SMU receives a service subscription request from the end user, it multicasts the request in the network, discovering enterprise LUSs which have advanced services. Following discovery, the SMU gets a response from the LUS listing all the advanced services it has on the network and the addresses/URLs of other LUSs outside of the enterprise which have the same kind of services available. Using this list and service list proxy code, the SMU constructs a service subscription form and sends it to the end user stating that these are the advanced services available. The end user selects the services that it wants, fills in the form and sends it back to the SMU. The form includes facilities for the user to specify service related data. The user may not find all the services it wants. In this case, the user should send a message to the SMU indicating that it wants particular services that are not in the form, the SMU then queries other available LUSs external to the enterprise LAN using the addresses/URLs from the last query. After the SMU receives the completed form from the user, it checks its User Profile, which contains all the services that are already in use by the user. Then it sends a request to the lookup service to download the proxy code (interfaces) of the services to which this user has subscribed. When the SMU checks the user profile, the following things may occur:

- The service / services that the user wants to subscribe to is already there; then the SMU sends a notification to the user indicating that the requested service or services are already provisioned.
- Some of the services the user requested are already available to the user; then the SMU sends a notification to the user, and it also sends the proper request to the LUS.
- If all the services are new to the user, the SMU sends a request to the lookup service to download the proxy code of the services that the user has requested.

The service proxy code will be downloaded to the SMU. The proxy code contains the interfaces that a gatekeeper needs to construct a USA, and the location of a SIR where the actual service code can be found. There may be many URLs for the addresses of multiple SIRs. If it is the first time the user is subscribing to services a customized USA is created and sent to the user agency where it resides in the user's terminal. If it is not, the USA, which is already in the user's terminal, is either replaced by a new one or updated in order to be able to handle the new services to which the user has subscribed. In the rest of this paper we assume that it is the first time the user subscribes to services. Once the user receives the USA, it acknowledges the SMU whereupon the User Profile may be updated. Note that if a user does not subscribe to any advanced services, its terminal will rely on built in basic call functionality and will not have a USA.

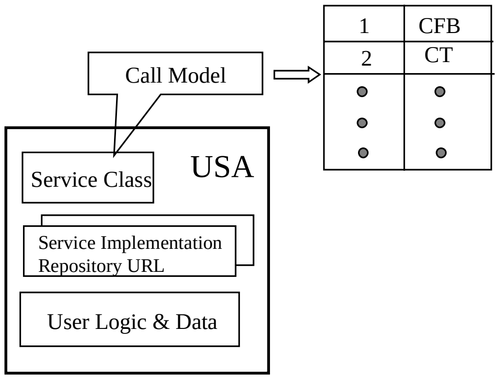


Fig. 3. User Service Agent

As is illustrated in Fig. 3, a USA consists of ServiceClass, Code Repository URL, and User Logic and Data. It defines how a call will be processed; e.g., the management of feature ordering. It handles service management and other aspects of network management; e.g., fault management.

Referring again to Fig. 3, the ServiceClass (Call Model) is a call model specific to an end user. We view the Call Model as a basic service, nothing more. The Call Model component of the USA is an IN call model consisting of two separate sets of call processing logic: Originating and Terminating call models. The Originating call processing logic provides support to the Calling Party, and is modeled by the Originating Basic Call Model (O-BCSM). The Terminating call processing logic provides support to the Called Party, and is modeled by the Terminating Basic Call Model (T-SCSM).

The call model provides support for a finite state machine with points of interaction with advanced service implementations. In the traditional IN view of advanced services, these points of interaction would be implemented using detection points. In the approach proposed here, using component-based technology, we would expect Java Beans to be used with well-known interfaces and the interaction mode would be via method call.

The User Logic in Fig. 3 represents processing that is required for subscribed services. For a specific service, in one call processing state, it specifies how we deal with this specific service and what the next step is in the call processing. For sophisticated services, the Call Model may have provision for the Enterprise Service Constructor or user to write scripts (rules, perhaps) that add a degree of intelligence to the service. Choosing an example from the e-mail domain, we might choose to write scripts that filter out particular callers, or calls from a specific set of network addresses.

The User Data in the above figure is the service-related data. For example, after a user chooses a service (e.g.; CFU), he will also be asked to fill in the phone numbers to which he would like to forward calls.

To reiterate, the User Service Agent (USA) consists of a Call Model, one or more Service Implementation Repository URLs (references to where the service code can be found) and User Logic and User Data.

A USA is constructed when the end user sends a request for service subscription. The call model will be unique to the user according to its subscribed services. For example, $user_{12}$ may subscribe to Call Forwarding Unconditional (CFU) and Call Transfer (CT). In this case, the ServiceClass component of the USA will be a call model that has different detection points at different points in the call. For CT, the detection point would be in the originating call model. For CFU, the detection point would be in the terminating call model. Once constructed, the USA moves to the user local agency from the gatekeeper. When the user has several services, the construction of a USA could be a tricky endeavor due to potential interactions between the services.

C. Service Utilization Using Mobile Agents

The Call Agent is a mobile agent, dynamically constructed, that implements the specific call model for a particular end user and makes use of basic call processing functions to control the call setup. It is generated using the USA as a factory at call time, and may do fairly basic call processing. For more sophisticated features like the VPN hunt group, it could use its mobility to carry out its task in a flexible way. It can execute in the gatekeeper agency, in the end user agency, or in the agency of another required resource like a database.

Call Setup and Service Utilization in a simple case where gatekeeper routed call signaling is used are shown in Fig. 4 Assume for this discussion that call originator $user_{11}$ has not subscribed to any advanced services but that $user_{12}$ has subscribed to Call Forwarding Unconditional (CFU). $User_{12}$ would have a USA containing info about the subscribed services. User$_{11}$ on the other hand would not have a USA and would initiate a call by invoking the Basic Call Processing (BCP) function in the user terminal. As a result of the function call, a setup message

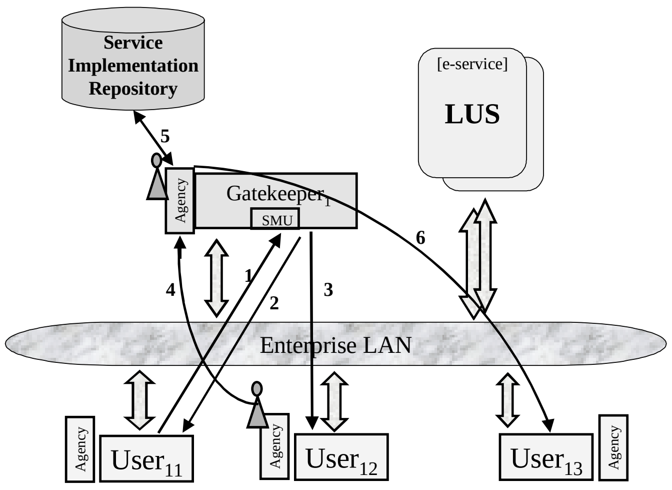


Fig. 4. Call Setup and Service Utilization

is sent to the called party (user$_{12}$) via the gatekeeper (1). The gatekeeper immediately returns a call proceeding message to the originator (2) and routes the setup message to the called endpoint (3).

For the called endpoint, as soon as it receives the first setup message, its USA moves to the gatekeeper's agency (4) if gatekeeper routed call signaling is being used. The SMU checks its user profile - user$_{12}$ has subscribed to CFU - and hence there must be a USA for user$_{12}$. So user$_{12}$'s USA is activated and a Call Agent (CA) which implements the call model is instantiated. The USA instantiates a CA by performing a new CallAgent (USA) call which retrieves necessary code from the SIR (5). At this time, the USA and CA reside in the same agency and the CA is ready to take over signaling for user$_{12}$. The CA sends call signaling to the called endpoint via the gatekeeper. Since CFU is in play, the CA sends a setup message (6) to the diverted-to party user$_{13}$. And so on. The fine details and H.323 signals are shown in section V

Note that for endpoint–to–endpoint call signaling, the USA will remain in the user local agency instead of moving to the gatekeeper's agency. The CA will reside in the user's terminal also.

Call control messages are sent between the CA on behalf of user$_{12}$ and user$_{13}$. Thus, the CA will know that whenever it gets an incoming call, it will forward the call the phone number (IP Address) of user$_{13}$ that user$_{12}$ has specified during the service subscription process.

D. Ongoing Service Management

The intention with the architecture proposed in this paper is to meet more of the needs of the phases of the life cycle of a service. So far we have discussed the phases up to and including utilization. Service Proxy Code downloaded to the SMU contains different service interfaces according to the services the user has registered for. There are four types of interfaces for service to discuss:

- Service Instantiation Interface. The call agent builds the operating service code in three steps. First of all using information from the proxy previously downloaded from an LUS, it now downloads the actual service code. It then loads the code using load methods also supplied by the LUS. Finally the Call Agent has access to an API (constructors) which it uses to create an actual instance of the service class
- Service Operation Interface. Here the service provides methods that are used to run the service- i.e. get it to do something.
- Ongoing Service Management Interface. This interface built into the service code itself provides the classical FCAPS management behavior (Fault, Configuration, Accounting, Provisioning, Security)
- Service Programming Interface. Manipulation and programming of the service logic can be applied through this interface. It also enables the composition of complex services, or stringing together of several features. This the point at which an algorithm like DFC (Dynamic Feature Composition discussed in IV.B) would interact with the service code.

Using the Ongoing Service Management Interface, an administrator issues SNMP-like GET/SET requests to manage the service. The administrator may also withdraw (or set status of) a service for maintenance purposes through this management interface. This management interface itself extends many interfaces, corresponding to the functional areas designated for Network Management. For example:

```
public interface ManagementInterface
                extends ConfigurationManagementInterface,
                        FaultManagementInterface,
                        SecurityManagementInterface,
                        PerformanceManagementInterface,
                        AccountingManagementInterface {
    public Object get( Object attribute, SecurityInterface
                    adminSecurityInfo);
    public boolean set( Object attribute, Object value,
                    SecurityInterface adminSecurityInfo);
    public boolean withdrawService(SecurityInterface
                    adminSecurityInfo);
    …..}
```

When subscribing to specific services, users may ask to be notified of changes to that service. User Service Agent registration with the services in the LUS enables notification of service changes. Jini event services are particularly useful in this regard. For example, should a service need to be taken off line for maintenance purposes or withdrawn, the users subscribing to this service can be notified thereby making possible the identification of alternate service providers. This notification of service changes takes place through the SMU which is notified by the LUS and then in turn deals with the end user. Such notification allows for enhanced user-controlled service management and calls for an appropriate interaction interface in the USA. Examples of the use of such interfaces have been described in the context of plug and play networking[26].

As mentioned earlier, if a service is to be dynamically upgraded, the SMU would be involved. In fact, through software hot-swapping[25] it is often possible to dynamically upgrade a service and change certain classes while it is still in use. This is important because with many users, there may never be a time when a particular service is never in use by some user.

IV. APPLICATIONS AND EXTENSIONS OF THE ADVANCED SERVICE ARCHITECTURE

A. Using XML to Support Advanced IN and Multimedia Services

The eXtensible Markup Language (XML) is used to format data into structured information containing both content and semantic meaning. Most importantly, XML provides a convenient and highly effective way to encode a service specification in such a way that an application can quickly determine the attributes of that service and the operations it can perform. There is an enormous potential that a structured service specification holds for revolutionizing how IN and multimedia services are created and deployed. By storing all of the semantic information for the feature with the feature itself a whole host of new applications are possible. One of the more interesting applications is Open Service Composition (OSC).

This involves constructing new IN and multimedia services from a library of component-based services or building blocks. These new services could be created using modified versions of existing H.323 telephony services or they could be built from a series of highly optimized feature engines. A feature engine is simply a component that performs a commonly used operation such as call setup; call redirection, access list management, billing, or linking multiple calling parties. In the next section, we will explore how the Advanced Service Architecture could be modified to support OSC.

B. Extending the Advanced Service Architecture to Facilitate Open Service Composition

The LUS currently provided with Jini is useful for finding and retrieving simple services. It uses a basic wildcard matching system for service lookup. This means that it either succeeds or fails to locate a service that corresponds to the set of attributes that form the search criteria. This will not be sufficient for Open Service Composition since there are potentially many ways to construct a new service. The LUS will always return no match for the attributes of the service described if it is indeed a new service. In other words, OSC is useful when a service is required that doesn't currently exist in the Service Component Repository. For this reason, a more fuzzy LUS will be required that allows the architecture to make intelligent decisions on which feature engines can be combined together to provide the functionality required. Creating a structured specification in XML of each feature engine and storing that specification with the feature engine itself can realize this fuzzy search. In order to support this, several changes will need to be made to the Advanced Service Architecture.

Currently, Jini services in our architecture are simply Java objects with well-defined interfaces that are stored for lookup and retrieval. However, in order to move to a component-oriented architecture, JavaBeans will be used. JavaBeans is a commonly used software component model for Java where each JavaBean is housed in standard container called a Java ARchive (JAR). The principal components of this JAR file are the properties and methods of that component and a BeanContext API, which allows these properties, and methods to be discovered and extended. Our architecture would require an addition to each JavaBean that would allow these properties and method descriptions to be stored in XML format. Additional information that may be useful for OSC could also be stored in this XML document such as dependencies on other components and information on potential conflicts with other components. The Jini LUS in the Advanced Service Architecture would be modified to allow the JavaBeans for all available services and feature engines to be stored and retrieved. This would involve adding an XML parser to the LUS so it can search through the specifications for each service for matches.

Finally, in order to facilitate open composable services, a distributed collaboration environment such as JavaSpaces or Tspaces could be used. While direct communication between Jini LUSs is possible, a Jini service such as JavaSpaces would allow services stored in multiple environments to communicate with each other. This added communication could facilitate software composition and coordination of services from multiple vendors assuming the services were based on widely used industry standards such as H.323 or SIP. Clearly, many more new services could result if a distributed tuplespace was available in the architecture.

Feature interaction and coordination is a rather difficult problem. In our prototype implementation, with a limited number of features/services available, feature interaction can be analyzed and solved by simple priority sequencing of services. There are no miracle solutions for the general problem but in a fuller developmental system we would anticipate exploring a more systematic architecture such as Distributed Feature Composition or DFC [23] within our advanced service architecture. DFC offers a more comprehensive pipeline filtering mechanism for ordering (rather than constructing) services with interesting possibilities for distribution.

C. Mobile agent based applications

No one has demonstrated a major application that can only be implemented with mobile agents. H.323 however, brings a host of opportunities for investigating applications that depend on mobile agents, maybe not in an essential way but at least in an important way. In the future we will investigate applications, in which, in the context of H.323, call agents visit a series of processors to complete their tasks. We will prototype them and show how they depend in an important (if not essential) way on mobile agents. These applications can be grouped in two categories that we term "advanced telephony services" and "Non telephony services".

Advanced telephony services

As previously mentioned in the paper, telephony services are provided today using the IN framework. The features as perceived by end users are often rudimentary and not very user friendly. The implementation is not time efficient. Mobile agents could help in not only enhancing the features but also in making the implementation much more efficient. This is illustrated below by the hunt group service. The IN based hunt group is described first. A sketch of an advanced mobile agent based hunt group service follows this description. It could be included as a VPN feature and could offer more features than the traditional one. Furthermore it would be more time efficient.

The traditional hunt group consists of a pilot number that corresponds to a group of end users. When the number is called, the end users that make up the group are tried one by one, following a predefined order, until the call is answered. In general, it is the first end user who is not busy who takes the call. In a pure IN implementation, the service is triggered in the switch and the switch consults the SCP that gives back the first number to try. The switch tries the number, and if the end user is busy, it gets back to the SCP to get the next number to try. The process is iterated until an end user answers the call.

The features of the traditional hunt group are rather rudimentary in the sense that it is the first end user that is not busy who takes the call. More advanced features could be offered in the context of Internet Telephony. Mobile agents can play an important role in the advanced hunt group. They can carry, for instance, a succinct description of the reason for the call (e.g. description of the problem to be solved) and display this it to the end user and ask if he/she is willing to take the call. The end user who picks up the call could be the first who thinks he can solve the issue. The traditional implementation is not very efficient in the sense that end users are tried sequentially The hunt group could be divided into sub-groups and several agents dispatched in parallel on the basis of an agent per sub-group. Several end users will then be tried simultaneously and when an end user decides to take the call, appropriate measures will be taken to stop the search in the other sub-groups. This implementation will certainly be more efficient and we will try to demonstrate this in future work.

Non-telephony services

Non-telephony services are commonly offered today by telephony services providers and more and more will be offered in the future. Examples in mobile telephony include access to the Web via the General Packet Radio Services (GPRS) and the Wireless Access Protocol (WAP). Internet Telephony service providers will certainly tap into that wealth of non-telephony services in order to enhance their service portfolio. Potential services to be offered include electronic commerce, information retrieval and office task automation.

It has already been demonstrated in the literature that mobile agents can play an important role in electronic commerce and information retrieval. In the case of electronic commerce, they can visit virtual stores, bargain, and purchase goods on behalf of end users. In the case of information retrieval, they can visit the site where the information resides and filter huge amounts of data close to the source. This can improve performance in a significant manner.

Internet Telephony will probably be first deployed in Intranet environments including PBX environments. This leads us to first focus on the non-telephony services that will help in automating routine office tasks. A first application we have in mind is the travelling administrative assistant. An administrative assistant handles many tasks but sometimes in an inefficient way due to the lack of appropriate tools. Mobile agents could play an important (if not essential) role in some applications that automate office tasks.

Booking a suitable time for a conference call or a face to face meeting involving several "over booked" executives can easily become a nightmare. It usually triggers a flow of emails and phone calls between the executives' administrative assistants and between these assistants and the executives. The process could be automated by the use of mobile agents. Let us imagine a mobile agent, perhaps a call agent, which visits each executive, accesses directly the appointment book, suggests (directly to the executive) alternative time slots and so on …We will explore this in future work.

V. Service Utilization Scenarios

In this section, several advanced service scenarios are presented. To illustrate the USA's and CA's movement on deployed advanced services using H.323 messaging. Several examples are provided, without a gatekeeper and with one gatekeeper involved and gatekeeper routed call signaling; i.e., Call Forwarding Unconditional (CFU) in sections A and B, and VPN in Section C. CFU is defined by ITU-T standard H.450.3; VPN is not currently defined in the standards.

A. Call Forwarding Unconditional Using End-to-end Call Signaling

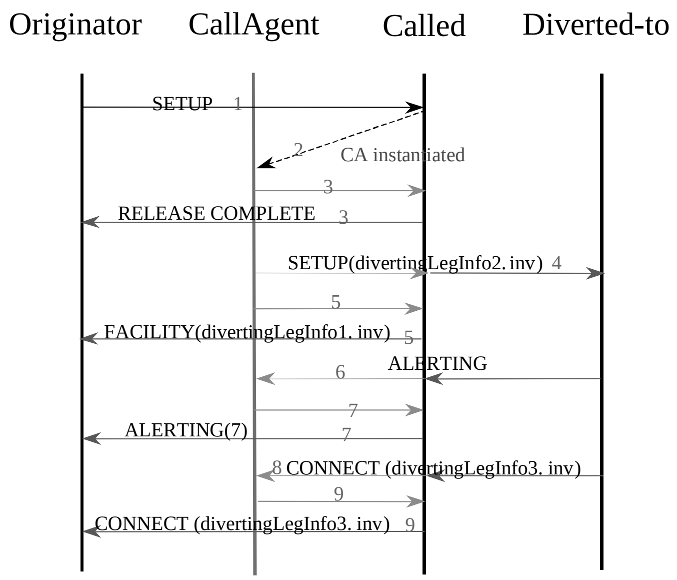

Originator
CallAgent
Called
Diverted-to
SETUP 1
2
CA instantiated
3
RELEASE COMPLETE 3
SETUP(divertingLegInfo2. inv) 4
5
FACILITY(divertingLegInfo1. inv) 5
ALERTING
6
7
ALERTING(7) 7
8 CONNECT (divertingLegInfo3. inv)
9
CONNECT (divertingLegInfo3. inv) 9

Fig. 5. CFU (End-to-end Call Signaling)

Fig. 5 is explained as follows:

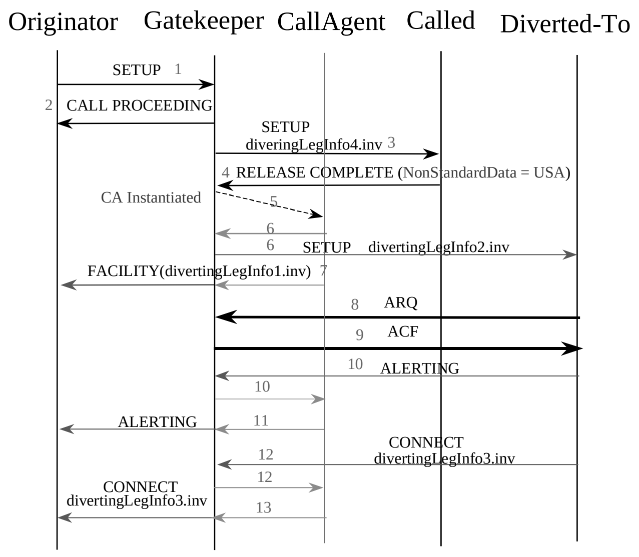
Assumption: The client application will be responsible for sending messages specified by the Call Agent to the other applications that are dealing with the call setup process. The Call Agent can accomplish this by calling a client application's interfaces.

Description:

1. An Originator (Caller) application sends a SETUP message to the Called party (Callee), a flag will be set in SETUP

message's NonStandardControl field to activate the USA, so that a Call Agent is instantiated.
2. After the call agent is instantiated -- it resides in the callee's user agency -- it takes over the call processing from the client application, sending out all the call setup related call signaling messages.
3. The Call Agent sends RELEASE COMPLETE to the caller.
4. The Call Agent sends SETUP with divertingLegInfo2.inv to the diverted-to party.
5. The Call Agent sends FACILITY with divertingLegInfo1.inv to the calling party.
6. The Diverted-to party sends ALERTING to the Call Agent.
7. The Call Agent sends ALERTING to the calling party.
8. The Diverted-to party sends CONNECT to the Call Agent.
9. The Call Agent sends CONNECT to the calling party.

B. Call Forwarding Unconditional Using Gatekeeper Routed Call Signaling



Originator
Gatekeeper
CallAgent
Called
Diverted-To
SETUP 1
2 CALL PROCEEDING
SETUP diveringLegInfo4.inv 3
4 RELEASE COMPLETE (NonStandardData = USA)
CA Instantiated
5
6
6 SETUP divertingLegInfo2.inv
FACILITY(divertingLegInfo1.inv) 7
8 ARQ
9 ACF
10 ALERTING
10
ALERTING 11
12
CONNECT divertingLegInfo3.inv
12
CONNECT divertingLegInfo3.inv
13

Fig. 6. CFU (Gatekeeper Routed Call Signaling)

Fig. 6 is described as follows:

Assumptions: Same as CFU using end-to-end call signaling. NonStandardData can be used to transport a USA.

Description:
1. The originator sends a SETUP message to the Gatekeeper.
2. The Gatekeeper responds with CALL PROCEEDING.
3. The Gatekeeper sends SETUP with diveringLegInfo4.inv to the called endpoint.
4. The called endpoint sends RELEASE COMPLETE with USA in the NonStandardData field.
5. Once the Gatekeeper receives the USA, a Call Agent is instantiated.
6. The Call Agent sends SETUP with divertingLegInfo2.inv to the diverted-to endpoint.
7. The Call Agent sends FACILITY with divertingLegInfo1.inv to the calling endpoint.
8. The Diverted-to endpoint sends ARQ to the Gatekeeper indicating it will accept the call.
9. The Gatekeeper sends ACF to the Diverted-to endpoint with Gatekeeper's call signaling transport address.
10. The Diverted-to endpoint sends ALERTING to the Call Agent via the Gatekeeper.
11. The Call Agent sends ALERTING to the originator.
12. The Diverted-to endpoint sends CONNECT with divertingLegInfo3.inv to the Call Agent via the Gatekeeper.
13. The Call Agent sends CONNECT with divertingLegInfo3.inv to Originator.

C. VPN - Outgoing Call Allowance (OCA) / Outgoing Call Restriction (OGR)

VPN services are illustrated here by OCA/OGR. The OCA and OGR VPN features allow an enterprise administrator to restrict the numbers that may be called from within an enterprise. For example, an easily accessible terminal might be restricted to numbers within the enterprise whereas a terminal on an employee's desk might have unrestricted dialing privileges. It should be noted that steps 3 and 4 below offer opportunities for a wide variety of services like the Hunt Group or other non-telephony services. Fig. 7 is explained in the following:

Assumptions:
Since the VPN service is not defined by H.323, here we are using the H.225.0 call signaling in order to illustrate the call management sequence using a Call Agent, and also to make it easier to understand by using the same style of call sequence diagram. The messages used here need to be identified in the future.

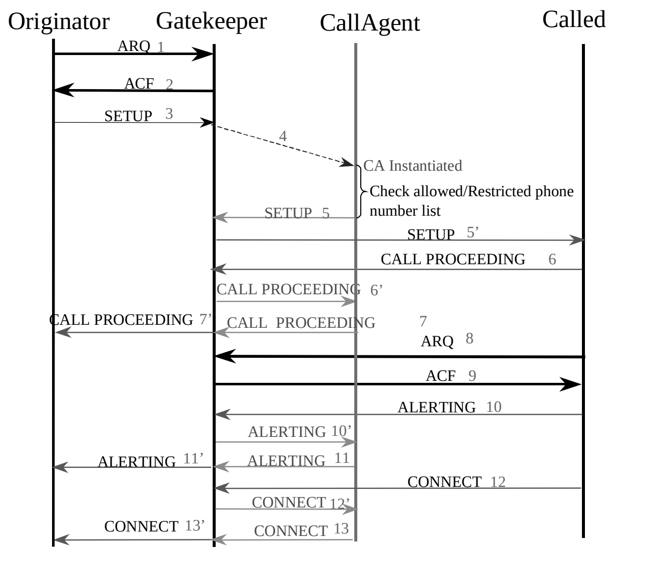


Fig. 7. OGA/OGR (Gatekeeper Routed Call Signaling)

Description:
1,2. The originator and its gatekeeper exchange admission messages.
3,4. The originator (Caller) application sends a SETUP message to the Called party and a flag will be set in the SETUP message's NonStandardControl field to activate the USA residing in the gatekeeper, so that a Call Agent is instantiated. After the call agent is instantiated, it resides in

the gatekeeper's agency. It checks the allowed/restricted phone number list first. If the called number is in the allowed phone number list or not in the restricted phone number list, then the CA takes over the call processing from the client application, sending out all the call setup related call signaling messages. (If the called number is denied, then the CA will send CALL RELEASE to the originator).
5. The Call Agent sends a SETUP message to the called party via the gatekeeper.
6. The Called party sends a CALL PROCEEDING message to the CA via a gatekeeper.
7. The CA sends a CALL PROCEEDING message to the Originator via the gatekeeper.
8, 9. The Called party and its gatekeeper exchange admission messages - ARQ, ACF.
10,11. The Called party sends an ALERTING message to the CA via the gatekeeper, and the CA sends an ALERTING message to the originator via the gatekeeper.
12,13. The Called party sends a CONNECT message to the CA via the gatekeeper, and the CA sends a CONNECT message to the originator via the gatekeeper.

VI. Conclusions

The mobile agent based advanced service architecture solution proposed in this paper provides the following features and benefits. The architecture can:

Enable the provision of flexible software solutions, where H.323 advanced services software is partitioned into mobile service agents realizing dedicated functionalities (e.g., IN service features).

It enables on demand provision of customized advanced services by open construction of a user service agent that uses downloaded service code from the SEC or ESC to the gatekeeper.

It allows for decentralized realization of advanced services, by means of bringing the user service agents directly onto the user terminals.

We have demonstrated that mobile agents may be successfully integrated with H.323 IP telephony protocols for the provision of advanced services. The architecture that this paper proposes seeks to address the entire service lifecycle, an important consideration in opening the IP telephony marketplace to non-traditional telephony service providers.

Our future work consists of the construction of IP telephony services not currently defined by the existing H.450.x specifications in order to further validate the architecture. We are also currently evaluating the traditional IN SIBs with a view to re-factoring the behavior provided by them. Finally, a performance evaluation of the existing architecture using typical hardware and software platforms needs to be performed. Result of these activities will be communicated in future publications.

ACKNOWLEDGEMENT

This work was supported in part by Ericsson Research, Canada and Communications and Information Technology Ontario (CITO). The contribution of David Mennie is also appreciated.

REFERENCES

[1] A. Gary, "H.323: The Multimedia Communications Standard for Local Area Networks", IEEE Communications Magazine, Dec. 1996
[2] "Internet Telephony", http://www.webproforum.com/siemens2.
[3] ITU-T Rec. H.225.0, "Media Stream Packetization and Synchronization for Visual Telephone Systems on Non-Guaranteed Quality of Service LANs", 1997.
[4] T. Magedanz, "Intelligent Networks", International Thomas Computer Press, 1996.
[5] CCITT Recommendation M.3010 "Principles for a Telecommunications Management Network" 1992.
[6] R. Minetti, E. Utsunomiya, "The TINA Service Architecture", http://www.tinac.com/specifications/abstract.htm.
[7] D. Clark, "A Taxonomy of Internet Telephony Applications." http://itel.mit.edu/itel/publicaions.html.
[8] ITU-T Rec. H.323, "Visual Telephone Systems and Terminal Equipment for Local Area Networks which Provide a Non-Guaranteed Quality of Service", 1996.
[9] M. Handley et al., "SIP:Session Initiation Protocol" IETF RFC2543, March 1999
[10] H. Schularinne, J. Rosenberg, "Comparison of H.323 and SIP", http://www.cs.columbia.edu/~hgs/sip/h323.html.
[11] B. Pagurek, and T. White, "A Quick Evaluation of H.323/H.450", Technical Report SCE-99-02, Systems and Computer Engineering, Carleton University, April 1999.
[12] "The Problems and Pitfalls of Getting H.323 Safely Through Firewalls", http://support.intel.com/support/videophone/trial21/H323_WPR.HTM.
[13] ITU-T Recommendation H.450.1 "Generic Function Protocol for the Support of Supplementary Services in H.323" and H.450.x Series
[14] T.Magedanz, K. Rothermel, S.Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?" INFOCOM 96, San Francisco, USA.
[15] J. Kiniry and D. Aimmerman, "A Hands – On Look At Java Mobile Agents", http://computer.org/internet/ic1997/w4021abs.htm.
[16] M. Breugst and T. Magedanz, "Mobile Agent - Enabling Technology for Active Intelligent Network Implementation", IEEE Network, May/June 1998.
[17] D. B. Lange, "Present and Future Trends of Mobile Agent Technology", Second International Workshop on Mobile Agents '98 (MA '98) Stuttgart, Germany, September 1998.
[18] Jini specifications, http://www.sun.com/jini/specs
[19] T. Jan, "Intelligent Networks", Artech House, Mass., 1994.
[20] A. Limongiello, R. Melen, M. Rocuaao, V. Trecordi and J. Wojtowicz, "An Experimental Open Architecture to Support Multimedia Services based on CORBA, Java and WWW Technologies", Proceedings, Fourth International Conference on Intelligence in Services and Networks, IS&N'97 Cernobbio, Italy, May 1997.
[21] O. Miauno, J. Urata, Y. Sueda, and Y. Niitsu, "Advanced Intelligent Network and the Internet Combination Service and Its Customization", IEICE Transactions on Communication, No.8, August 1998.
[22] Gbaguidi, C., Hubaux J-P., Pacifici G., and Tantawi A., "An Architecture for the Integration of Internet and Telecommunication Services", Proceedings of IEEE Openarch '99,
[23] Jackson M., and Zave. P., "Distributed Feature Composition: A Virtual Architecture for Telecommunications Services" IEEE Transactions on Software Engineering, Vol.24 No. 10, October 1998.
[24] A.Park, A. Kupper, S. Leuker, "JAE: A Multi-Agent System with Internet Services Access", Proceedings, Fourth International Conference on Intelligence in Services and Networks, IS&N'97 Cernobbio, Italy, May 1997.
[25] N.Feng, G. Ao, T.White, and B. Pagurek, "Software Hot-swapping Technology Design" , Technical Report SCE-99-04, Systems and Computer Engineering, Carleton University, Ottawa, June 1999.
[26] S. K. Raza, , "A Plug and Play Approach with Distributed Computing Alternatives for Network Configuration Management" M.Eng. Thesis, SCE Dept., Carleton University, Ottawa, Canada, April 1999.