An Intelligent Network Architecture for Internet-PSTN Service Interworking

Menelaos K. Perdikeas, Iakovos S. Venieris
National Technical University of Athens, Athens, Greece.

Key words: Intelligent Network, Service Interworking, Distributed Object Technology

Abstract: This paper examines Intelligent Network (IN) approaches to interworking between the PSTN and Internet. It notes that until now, convergence between Internet and PSTN has taken place mainly at the transport and (less so) signalling layers. Even there however, the integration is not seamless enough to guarantee the ability to use the same IN superstructure over both constituents. This is an important concern given the investments that have been made in IN technology and its ability to introduce new services expeditiously. Further, the Internet world is more service-rich than PSTN and has a completely different service model. Telephony services are but a small fraction of the services that Internet can offer. Can IN retain its validity when examining provision of services other than telephony? How can IN concepts be put to use to allow interworking between, for instance, the web and the phone? In this paper we are describing an architecture that makes use of distributed object technology and IN principles in order to provide a comprehensive framework for service interworking.

1. INTRODUCTION

To date, convergence between the Internet and switched networks has for the most part taken place at the transport layer. This if for instance the case in dial-up connections when the local telephony access network is used to provide the missing last mile to connect the Internet Service Provider with the subscriber homes. This would also be the case when telephony service providers are routing calls through an IP network to reduce costs. In both

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35522-1_37

H. R. van As (ed.), *Telecommunication Network Intelligence*

situations only the lowermost layers are replaced and so convergence can be viewed less portentously as an instance of the encapsulating ability of layered protocol stacks. Where facilities of the one network are used by the other this is always on a rudimentary level. There are also some instances of convergence in the signalling layer for the particular class of telephony-like services. However, with the exception of the PINT protocol [1] for 'click-to' services there is no instance of service-level interworking between these two networks.

That the lower layers are interchangeable and accommodating their usage for the purposes of a different protocol stack should be attributed to the fact that these layers are always more generic and have more in common. Not so in layers above the data link one where differences between IP's connectionless mode and the (virtual) circuit switching Public Switched Telephone Network (PSTN) are irreconcilable. Differences in purposes and orientation are largely to blame for this disparity.

The PSTN (a term which in this paper will be used as encompassing also the technologies that have enhanced it like ISDN or ATM) is oriented towards the provision of the telephony service and other related services (like, more recently, video-telephony or video-on-demand). All these services share the concept of a call after which a (uni- or bi-directional) bit stream can be established between the parties participating in the call. The class of telephony-like services can be defined more narrowly as one comprising those services who have a call state model compatible to that of the Basic Call State Model (BCSM) as defined in the Intelligent Network (IN) specifications. Since IN exerted much effort to make the underlying network architecture service independent we can rationally expect to find in the BCSM a quite parsimonious service model that abstract as much as possible and identifies only what is intrinsic to the services provided by switched networks. So for the remainder of this paper we will use the term 'telephony class of services' to mean all services whose behaviour can be described in terms of a BCSM.

The Internet on the other hand, before the standardisation of popular application level protocols like the Hyper Text Transfer Protocol or the Simple Mail Transfer Protocol, was just a generic packet switching network used to interconnect mainframe computers. These services do not naturally lend themselves to description using a BCSM. Nevertheless once these services became available and commanded a critical mass of market attention, it was inevitable that demand would start to appear for two distinct abilities: (a) to access a network's service using the facilities of another and (b) to having discrete services coming from different networks work together for the needs of some envisaged solution to a real problem.

This paper is structured as follows. In the next section we will examine signalling interworking between PSTN and the Internet. Signalling interworking satisfies demand (a) and is shown to be narrower in scope and a sort of preamble for the support of a more thorough interworking at the service level which satisfies demand (b) and is covered next. We next introduce a layered service architecture that accommodates under its umbrella both signalling and service-level integration and interworking between PSTN and Internet. The architecture draws heavily from IN principles and can be used in a complementary manner with the specifications produced by the Parlay group [2].

2. SIGNALLING INTERWORKING BETWEEN PSTN AND INTERNET

The basic concern when seeking to integrate telephony in PSTN and Internet is that signalling emitted from PSTN terminals should transparently find its way to Internet nodes hosting Voice over IP (VoIP) applications and vice versa

The approach is usually to terminate some of the lower signalling layers and convey the upper layers transparently. Therefore the key point of comparison between different strategies is the level at which they terminate PSTN signalling. The decision of what types of Internet signalling components should be supported and of what the overall signalling architecture would look like largely depend on this subtle point. As an example work done in the IETF SIGTRAN group aims to relay SS7 signalling over the Internet (in effect, using it as a trunk line). There are at least two internet drafts [3] proposing to adapt the SS7 stack onto the Internet. The first provides an adaptation layer at Message Transfer Part 2 while the other at Message Transfer Part 3. Figure 1 depicts in a generalised manner these two approaches.

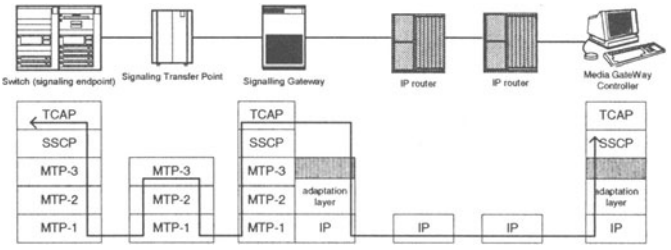

Figure 1. A number of signalling layers terminated, the others being relayed in the Internet side

There is another more clear-cut solution which can be seen as an extreme variation of the interworking approach. Namely, to terminate all signalling layers and completely encapsulate the signalling, conveying it transparently over the Internet. This however can lead to severe deficiencies mainly in terms of quality of service as no signalling information is made available to entities that could provide some sort of control over established connections at a local (e.g. enterprise) level. In general the main advantage of the more balanced approach to let some signalling layers pass on to the Internet side is that as signalling information becomes available at Internet nodes (these nodes hosting Internet Telephony components), it is possible to provide both (a) for the applications specific aspects of call control without needing to establish transactions all the way down to the actual terminals and (b) to impose some heuristic algorithm for QoS (e.g. by restricting the number of allowed connections within a domain). Naturally, the trade-off in this solution is that in addition to the terminals some Internet nodes are require to be augmented and able to play the role of their PSTN counterparts. For instance the H.323 architecture for VoIP foresees entities like Gatekeepers and Media Gateway Controllers which are equivalent in functionality and state information maintained with the PSTN switches.

Another equally important concern is the ability to allow the same IN superstructure to be used over both constituents. This is important in preserving the investments that have been made so far in IN and to take advantage of its efficiency in expeditious service creation and deployment. Signalling interworking as defined in the previous paragraph is an obvious prerequisite to the extension of the IN's realm of control to the Internet. But to support seamless IN operation more than that is required and some – to an extend artificial – constraints need be imposed.

The ability of IN to apply its services rests on the fact that state information on active calls is maintained by the network (at the Service Switching Points - SSPs). By examining this state and operating on it, IN services can be provided requiring only that a protocol is defined to communicate that state and IN-incurred operations that may modify it (a typical listener-controller pattern). Signalling as it is applied in the PSTN can be described as a protocol pattern where the uppermost layer protocol is not transparently conveyed end-to-end but is explicitly relayed and observed by intermediate nodes. We will not get into the discussion of why this particular protocol pattern is so prevalent in telecommunications while nearly non-existent in the Internet. One reason is that the progress of call and call control signalling is inextricably intertwined with the reservation of resources along a network path. One other is that it was necessary to keep the partial image of the state of a call in synch among the switches participating in a call and that therefore protocols that had a bearing on the

call state had to be terminated and re-instated at each switch so that the partial images were kept synchronised. This protocol pattern which is not familiar in the Internet needs to be applied even if only for reasons of allowing IN to be used in the Internet as well. In order to do so it is necessary that i) at least call and call control signalling is not encapsulated but is relayed over to Internet and ii) that that signalling is not conveyed end-to-end but passes through Internet nodes which are designed to terminate it, observe it, operate on it and then re-instate it. Such nodes in H.323 terminology are known as Gatekeepers and this process is known as Gatekeeper-routed signalling.

In all cases and disregarding the Internet Telephony architecture that is used (H.323 or SIP) the key issue is of performing the appropriate mappings between the various protocols that can be used in IP telephony and making sure that a common BCSM can be suitable for all of them. Since the call control of H.323 is strongly influenced by Q.291 both Q.2931 BCSMs and (more easily) Q.931 BCSMs can be mapped to H.323's call model. The SIP model although much simpler can also be mapped to the previous call models. Refer to [4] for concrete examples.

Integrating the two networks at this level allows the same set of services to operate over both types of networks with no need to differentiate between the two. The notion of a service layer is thus given rise to, in which services invariably engineered are executed perceiving only a set of similar SSP-like entities and being completely oblivious as to whether the SSPs they interact with will effect their actions through the issuance of ISUP, H.323 or SIP messages.

3. SERVICE INTERWORKING

The second class of problems as has been mentioned earlier has to do with the ability to define interworking schemes between different kinds of services. The Pstn-Internet iNTerworking (PINT) specifications provide one such example in the click-to services. However the approach in PINT is too narrow in scope and qualifies for little more than an ad-hoc arrangement. In the more general case, service interworking aims for a framework that can enable services to find each other, communicate and share their capabilities. A protocol-based infrastructure is inhibiting the realization of such a framework for a variety of reasons. The lack of a common data model, common wire format and common procedures for directory and lifecycle operations are some of them. In a distributed processing environment the use

of distributed object technologies offers a way out of most of these problems.

In particular, the exposure of the functionality of a service as a set of distributed, stateless APIs offers a neat way to export functionality in a self-explanatory manner bundled in coarse or fine-grained bundles as seen fit. That these APIs are stateless and usually synchronous further contributes to the robustness of the system as it minimises the number of states that the system may found itself into and it makes it harder for even an erroneously or maliciously operating client to throw the system out of synch. Another important advantage is that by using a distributed infrastructure we are replacing a fully-meshed graph of inter-service relationships, protocols and data models with a hub structure. All distributed objects plug into the same bus and share the same wire format, data model and protocol conventions. As it turns out service interworking comes naturally once participant services export their self-explanatory interfaces on a common scope allowing others to find their facilities, interact with them or employ them for their own purposes. A flat plane where remote objects (the service's proxies in the distributed world) interact is all that is needed for architectural support of service interworking to be in place. What flexibility is then allowed or how arbitrary or articulate interworking schemes can be defined is something that is determined solely by the nature of the interfaces that the services export into the distributed plane and by their expressiveness. This is just as it should be as it is impossible to abstract from arbitrary services. It is true that the telephony-like services that bear an underlying resemblance at the BCSM level have indeed be successfully abstracted and have lent themselves uniform treatment under IN but until now no other class of services has emerged with the same ubiquity, degree of penetration and level of coherence that would enable the appropriation of control models similar to the ones used in IN. The Parlay group considers this flat, API-based approach in order to export network-centric functionality to edge-of-network developers and to allow the development of cross-network services.

Once it has been decided that no higher level abstractions can be pursued for services other than telephony and that interworking between these services will be based on distributed APIs the focus shifts to architecture-related, service-independent considerations. These have to do with the ability of the services to find each other at runtime, with the binding semantics that the distributed processing infrastructure supports, with the copy and pass-by-value semantics and with mobile code. For the remainder of this paper we will assume that the distributed infrastructure is based on CORBA and that service code is in Java as the combination of these two technologies provides in our opinion the better solutions to the above problems (CORBA in the

specifications of common object services and facilities and Java in code mobility).

4. A SERVICE INTERWORKING FRAMEWORK

At this section we describe a service architecture aimed at meeting the following requirements:

- Telephony-related services are integrated at the signalling and IN-control levels enabling existing IN architectures to control telephony in the Internet.
- Services expose their functionality to a control layer through open APIs allowing arbitrary interworking schemes between them to be defined
- Cross-network services can be built and operate seamlessly across diverse underlying network technologies
- Services can roam the network for load balancing or personal mobility reasons.

For a unified cross-network service framework there is no reason to reserve different treatment for the telephony services. These services like all the others will expose a set of primitives (a distributed API) in the form of CORBA objects on the distributed service layer. The only provision specific to the telephony services should be the accommodation of signalling interworking between PSTN and Internet and the use of IN to control and provide services in both segments. Even this however is not manifested to the service layer which only observes a set of service primitives exported in an API form (using the Interface Definition Language). Telephony IN seen from the global scope of our architecture is reduced to an implementation strategy for telephony services. The listener-controller pattern however by which IN services operate is adopted at the service layer where services register as listeners to each of the services proxy CORBA objects allowing them to be notified of events, changes in their status and offering them handles through which to effect some behaviour first on the CORBA objects and ultimately to the network resources which they represent.

Figure 2 shows the architecture of such a unified service provisioning environment.

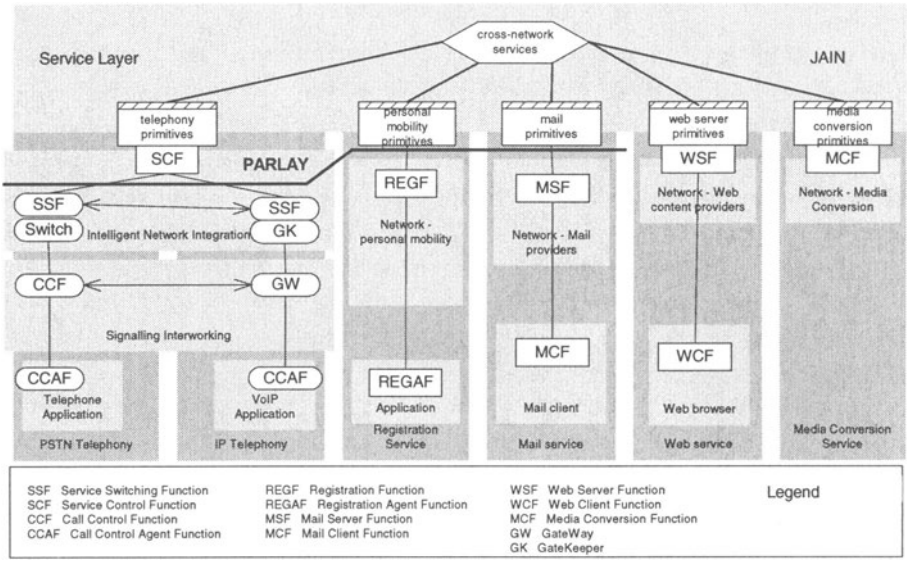

Figure 2. A Unified Service provisioning environment

According to Figure 2, telephony services seen from an architectural point of view are indeed one of the many vertical services that are available and that expose part of their functionality to a horizontal service layer above. It is also noted that the Intelligent Network as it is used for the purposes of the telephony services has little impact on the architecture at large. Provided that the same 'telephony primitives' or APIs could be exposed to the interworking service layer, it would have made no difference if services were switch-based.

Cross-network services exist as distributed objects in the service layer registering as event listeners to the CORBA objects representing the services and using the APIs the latter export to interact with the underlying networks. Figure 2 also depicts the interface line where Parlay standardisation efforts are focused and the area where ideas coming from JAIN [5] (for Java-based services) can be put to use.

Compared with telephony, services on the Internet have the disadvantage that they are ad-hoc and usually destined to spawn a shorter life-cycle. Few Internet services will reach the maturity of the telephone service and certainly none's will the standardisation be so carefully scrutinised. It is therefore necessary that any service interworking architecture takes that matter into account. Appropriate white or yellow pages mechanisms are necessary for service discovery and for allowing services to connect and use other services. The CORBA Naming and Trader services fall into that category. Resilience, fault-tolerance and transactions capabilities are all necessary components for a robust service infrastructure and fortunately

standard interfaces to such services have been made available by the OMG (the consortium that is responsible for CORBA standardisation).

So the vertical services that appear in Figure 2 should be interpreted as pluggable components that do not form an integral part of the architecture. More dynamically yet the CORBA objects that represent these services in the service layer should be assumed to be of indeterminate uptimes and possibly replicated in different localities.

We will explore this issue as we delve more deeply into the structure of the service layer.

Figure 3 is a more detailed version of Figure 2.

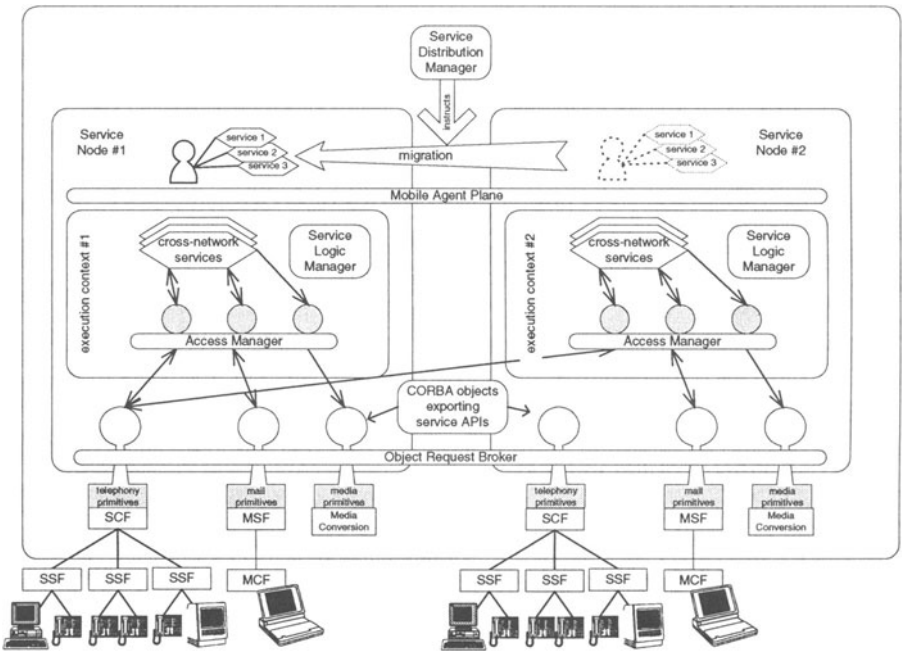

Figure 3. Service Layer for cross-network service provisioning.

It is seen that services are executed inside an execution context that is the same irrespectively of the service node (i.e. physical entity) where they are located. Programmatically the services are Java objects and thus their communication with their execution context is Java-based. Each cross-network service can register with Java objects that simply relay commands and events between the service logic programs and the underlying CORBA objects that represent the actual facilities. This is important as in a given service node a particular service might not be supported. For instance, in Figure 3 the SCF-based facilities are not available in the second execution

context. The second service node's execution context (the access manager entity) transparently to the executing service substitutes the locally unavailable SCF facilities with a remotely located one.

Execution contexts hosts the service logic programs and are responsible for resolving such things as intermittent CORBA objects, local unavailability of a given facility, directory searching etc. So even if a given facility is not available at a particular physical location so long as a proxy CORBA objects exists somewhere in the distributed bus that provides access to the service capabilities, the execution context of the said physical node will be able to contact it transparently and still provide that facility to the service logic program that it hosts.

The alternative would be that services executing in service node 2 directly access the remotely located SCF object through CORBA. We want however to insulate services from the distributed platform we use and so this immediate step is necessary. With it we keep our services Java-based eliminating the need to upgrade them in case the distributed infrastructure we use changes. Services are simple Java objects that do not observer libraries other than the standard libraries that are available with the Java virtual machine. We also get to keep the location transparency offered by CORBA (or any other distributed processing technology used for that matter) as it is up to the access manager to connect the Java proxy objects with the real distributed objects to which they relay their commands or from which the receive event notifications.

Java nicely complements CORBA's implementation transparency due to its write-once-run-everywhere characteristics. Thus the same Java code can be executed in any service node disregarding its operating system or hardware architecture and that Java code will in the process ultimately (i.e. through the access managers and the proxy CORBA objects) interact using distributed APIs with proprietary code running in the SCFs, the mail servers or the media conversion centre without the latter pieces of code being able to distinguish the type of code they interact with. While the implementation transparency offered by CORBA is lost on the one side of the interface as service code will need to be written in Java, the uniform execution environment provided by the Java VM is more essential to any telecommunication architecture than the ability to have service code written in any language. After all, it is the diversity of equipment and operating systems that plagues telecommunication networks and could hinder an attempt of cross-network interworking. Furthermore, we get to keep implementation transparency at the side of the interface where it matters more: below the service layer where we are interfacing with legacy equipment.

By (a) using CORBA at the lower level in order to wrap native components and export their interfaces in a standard manner into the service's execution context and (b) using a Java-based execution context in our architecture's service nodes, we are exploiting the best of both worlds. With (a) comes a common data and object model, an API-based approach (with distinct advantaged over protocol-based ones) and seamless encapsulation of underlying network technologies. With (b) comes the power of Java in terms of mobility of code, safety (a simple language with automated garbage collection and fewer pitfalls for the developer to be aware of) and a standardised object oriented development and runtime environment that has entered the mainstream.

Another benefit accruing from the separation of the service logic programs from the actual distributed processing technology used for conveying event notifications and commands between the Java objects and the CORBA objects is that it is much easier to provide fault tolerance mechanisms at the access manager level than at the service logic programs level. Putting the code handling all possible eventualities into a service logic program would bloat the service code out of proportions and would hamper the mobility procedures described later in this paper. Note that we have already made considerable savings in terms of service logic size by allowing the services to remain 'pure' Java objects and putting stub and skeleton code in the access manager entity (which is responsible for CORBA to Java mapping and vice versa). Technically speaking our services are not CORBA objects (they do not extend a pre-compiled skeleton) nor do they need to be compiled with the stubs of other CORBA objects which they contact. All this is done at the access manager level.

We envisage a situation where the geographical dispersion of service nodes is indeterminate and not known beforehand. Service nodes may be added or removed from the network. It is clear that some of the cross-network services that can be supported by our architecture have topological associations with specific physical elements. For instance a service that has registered with the mail primitives provided by a mail server, expecting to receive notifications when new mails arrive would preferably be located somewhere 'close' to that mail server. Similarly it would be optimal for terminating or originating IN services to be 'close' to the terminating or originating switch. Personal mobility adds a further degree of freedom to this problem as users may out of their own accord move from one location to another rendering a previously optimal situation, sub-optimal. Services operating in such a fluid environment would have to be able to move between service nodes either to follow the user to whichever network he is roaming or to respond to varying network link conditions or even service

node overloading or temporal unavailability. As service nodes should be able to be introduced or removed dynamically with no need of configuration scripts or pre-allocated distribution loads service could use all the mobility capabilities that could be afforded to them. That the service logic is implemented in Java and thus is in a semi-interpreted form ready to be interpreted on the fly helps promote mobility of service logic.

Java offers code mobility. It is however sometimes necessary to transport state as well as code. In these cases, a mobile agent platform should be used. Mobile agent platforms [6] transport state by serialising the objects and transferring their serialised form to their destination along with the code. In our system it is not necessary for service logic programs to be mobile agents themselves. Instead they can be simply transported by means of a migrating agent that holds pointers to them. Indeed the specifications for the Java serialisation describe that when an object is serialised the entire graph of objects to which it holds pointers (and, recursively, the objects to which the pointed objects hold pointers) are serialised. So a pattern can be easily derived by which service logic programs are being freighted by mobile agent objects implementing a simple 'freighter' interface. This has two distinct advantages:

- we are disentangling the properties of being a mobile agent and a service logic program. In narrower terms we are keeping the two class libraries separated and have removed from our service logic code any dependence on a particular mobile agent middleware. More than that, we have clearly delineated the role of mobile agent technology to that of an enabling technology for dynamic distribution of code for load balancing or other purposes – we have not made it an integral part of our approach. Note in this context that the same can also be said of the CORBA technology. From a functional point of view it does not matter if the access managers communicate with the entities they control through CORBA or some message-based protocol.
- Seen from a performance point of view, we are making migration operations less costly. Since services will not extend the – usually bulky – base mobile agent class their code size is reduced accordingly. Secondly, a single mobile agent can carry with it more than one service logic programs yielding a reduced overall bytelength.

For the architecture to be able to load-balance itself in that manner it will be necessary that a 'service distribution manager' takes into account two categories of events: (i) asynchronous events happening in the service node network. For instance, by the addition, removal, temporal suspension of a service node or by signalling overloading across a link or computational overloading on a service node. (ii) personal mobility events. A mobile agent is then instructed to migrate to the location the manager estimates as optimal

and the service logic programs are attached to it so that they may follow it on the migration.

The architecture is in this way self-balancing without imposing any special requirements on the service logic programs.

5. CONCLUSIONS

In this paper we presented a service architecture for the purpose of providing an overall framework for service-level interworking between Internet and PSTN in general and for facilitating the creation and execution of cross-network services. We argued that IN is limited in scope and that in a environment where PSTN has converged with the more service rich Internet, it should be viewed simply as an implementation strategy for telephony services. Its listener-controller pattern can however be put into use by allowing services to expose their functionality to a service layer through open APIs allowing arbitrary interworking schemes between them to be defined. Using distributed object technologies it is possible to provide uniformity in data and object models and wire-formats that is indispensable in interworking between different services. By making use of distributed processing technologies the architecture can expose network-centric functionality to edge-of-network service providers. In that sense it is oriented towards the same objectives that have motivated the Parlay specifications. In doing this and by exploiting mobile code capabilities, it enables nomadic computing and personalised services that roam the network to follow their registered user.

REFERENCES

[1] Internet Draft: "The PINT Service Protocol:Extensions to SIP and SDP for IP Access to Telephone Call Services" (IETF PSTN and Internet Interworking Working Group).

[2] Parlay API Specification 2.0, available from http://www.parlay.org

[3] See the IETF's Signaling Transport working group at www.ietf.org

[4] Ackermann, D. and Chapron, J. "Is the IN Call Model still valid for new Network Technologies?", International Conference on Intelligence in Networks 2000, Bordeaux, France.

[5] JAIN: Integrated Network APIs for the Java platform, available from www.sun.com

[6] M.K. Perdikeas et al., Mobile Agent Standards and Available Platforms, Computer Networks, Elsevier, Volume 31, pp. 1999-2016.