*IN*EXPENSIVE OPEN DISTRIBUTED SERVICE PLATFORM

Tor Martin Didriksen, Lars Sivert Sørumgård, Tor Øystein Molnes
Telenor R&D, Kongens gt. 8, 7004 Trondheim, Norway

Key words: Intelligent Networks, Distributed Platforms, CORBA, Parlay

Abstract: This paper presents a platform to interconnect an IN node (SCP) to other communication networks (Internet) in order to introduce new IN services and extend capabilities of existing ones. The experimental platform presented is based on standard industrial hardware and software components. A hypothetical case study illustrates the versatility of the various interfaces provided by the platform.

1. INTRODUCTION

The telecom service providers tend to constantly offer more advanced services. Service uniqueness is a means of success in a competitive telecom market. The intention of intelligent networks (IN) is to provide means to meet these challenges by offering a platform for service development and execution. This goal has to some extent been met, and ongoing standardisation [1][2][3] indicates that IN tries to keep pace with the growing demands. Still, between one and three years are probably required for successful introduction of a quite new tailored service [4]. The main reason for this is that the SCP manufacturers have failed in providing platforms supporting *standard tools*, *multi-protocols,* and *standard databases.*

A serious problem caused by the lack of tools, and thus specialised skills required to developing IN services, is *poor programmer availability.* We believe that an open platform is the only means to decreasing *cost* and *time-to-market* for new IN services.

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: 10.1007/978-0-387-35581-8_35

T. Yongchareon et al. (eds.), *Intelligence in Networks*

This paper describes the architecture of a distributed and reliable experimental service platform that is not intended for IN services exclusively. The platform is based on standard hardware and software components.

Section 2 describes a typical telecom service that will be referenced throughout the paper. Section 3 identifies a number of requirements to a service platform. Based on these requirements, Section 4 presents an experimental platform suitable for future services. Section 5 shows how the experimental platform can be opened up against additional access networks as well as external service providers.

2. CASE STUDY: TRAVEL AGENCY (TA)

This section describes a service for handling customers calling a travel agency (TA). We assume that the service was originally developed and available for PSTN only. While introducing the experimental platform later on, we will show how the service may be extended to include interfaces to additional access networks like *mobile networks* and the *Internet*.

We will also show how to outsource the execution and development of the service to an external service provider interfacing to the experimental platform using a standardised interface like *Parlay* [10].

The reader should note that the service described here is just a hypothetical example to illustrate the discussion in this paper.

2.1 Functional Properties of the Service

The TA has one toll free phone number, and a number of offices handling calls to this number. Each office has a number of travel agents. The caller is first exposed to a menu listing a number of options, e.g. *domestic travels*, *travels abroad*, *holiday travels*, etc. Each option is associated with a digit to be entered by the caller to indicate the topic of the request.

Each option in the menu is associated with a dedicated queue and a list of offices able to handle calls concerning this topic. Thus, an office is homogeneous in the sense that all agents in the office can handle the same type of requests. After selecting the most suitable option, the call is first attempted transferred to the offices in the list associated with the queue. If all agents in all relevant offices are busy, the call is inserted into the queue. Regularly, the first call in the queue is attempted transferred to the offices.

When the caller is eventually connected to an agent, the two parties agree on a product and a price. The caller may provide his credit card number to pay in advance, or he may choose to be invoiced. Information about regular

customers may be registered in advance and stored as customer profiles at the TA in a database.

2.2 Service Architecture

A possible overall architecture of the travel agency service is depicted in *Figure 1*. There are basically three independent parts in this architecture:

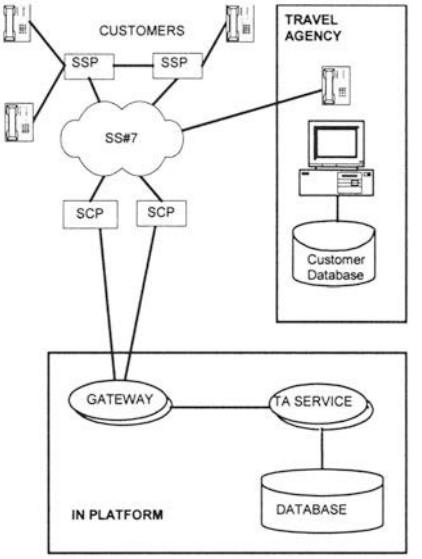

Figure 1. Hypothetical travel agency service

1. The customers, PSTN and the SCP.
2. The IN Platform with the Gateway to PSTN, the service for queue handling and a database to store the queue.
3. The travel agency, possibly with its own customer database.

The service described here is very basic, akin to what many companies have. Towards the end of this paper, we will explain how the service may be extended in various ways, with respect to functionality as well as how to outsource the service to an external service provider, and finally how to open up the platform for additional access networks.

3. EXPERIMENTAL PLATFORM REQUIREMENTS

An experimental platform for IN services should be:

- *Open*: In this context, this is related to the concept of using standard hardware and software as platform building blocks.
- *Reliable*: This is a very broad term [5] and includes both *availability* and *security.* Telecommunication services traditionally have a high level of reliability.
- *Distributed*: This means that the platform should facilitate the decomposition of a service into autonomous entities communicating through specific interfaces.
- *Accessible*: This means that the platform should be available in two respects: To end users, by means of various access networks, and to service providers (other than the network provider), by means of a standardised interface to the service platform.

In the following sections, we will elaborate on these requirements.

3.1 Openness

An open service platform means a platform implemented using widely applied, standard, commercially available hardware and software components. There are a number of reasons why we consider this a requirement to the platform:

- Applying widespread technology means that the number of alternative solutions to choose from is high, and that a suitable product in terms of cost and quality can be purchased.
- Hiring qualified and skilled people to develop and maintain the service platform and the services is easier since more people can be assumed to have experience in the technologies applied.
- Interfacing the platform to other entities such as additional access networks and services provided by external providers is easier.
- Keeping up to date with the technological evolution is easier because new technology is likely to be based on the most promising and widespread existing technology.

3.2 Reliability

The platform should be *fault tolerant,* i.e. be able to recover from failures without performing incorrect actions. *Recoverability* means that failed components are able to restart themselves and rejoin the system after the cause of failure has been repaired. There are two related aspects of reliability:

- **Availability**: The platform should provide *high* to *continuous availability*. This means that the platform should be able to resume providing services *during recovery* from failures. By minimising

recovery time, the platform is capable of providing virtually uninterrupted service to its users.

- **Security:** Security protects an information system from unauthorised attempts to access information or interfere with its operation [6]. Since the platform should be accessible by both *subscribers* and *external service providers*, a multitude of security issues should be investigated:
  - Identification and authentication of both humans and objects.
  - Authorisation and access control.
  - Confidentiality.
  - Security auditing.
  - Security of communication.
  - Accountability and Non-repudiation.

Accountability and *Non-repudiation* are related terms. Users are accountable for their security-relevant actions. Non-repudiation provides irrefutable evidence of actions such as proof of origin of data to the recipient, or proof of receipt of data to the sender to protect against subsequent attempts to falsely deny the receiving or sending of the data.

3.3 Distribution

Distributed object technology is well-suited for creating flexible systems because the data and business logic are encapsulated within objects, allowing them to be located anywhere within the distributed system. This is beneficial in terms of providing a logical decomposition of the system into independent units with clear interfaces, but also because the logical separation of entities easily can be extended to a physical distribution as well, thereby facilitating replication and increased reliability.

Another benefit of distribution is that scalability is supported since the system easily can be augmented with a number of additional components of a certain type. This of course assumes that there is a mechanism for distributing system load among numerous components of the same type.

Distributed objects separate their interfaces from the implementation. They are able to describe their interfaces, events and properties "on the fly". The implementation language of server objects is transparent to clients, which enables the use of distributed objects as wrappers for existing applications regardless of implementation language. Existing IN services should therefore smoothly migrate to the new platform.

The basic idea of the distributed object middle-ware architecture is the object bus, which e.g. in CORBA is the *Object Request Broker* (ORB) that lets objects interoperate across address spaces, languages, operating systems and networks. The bus provides mechanisms that let objects exchange meta-

data and discover each other. At the next level, the infrastructure augments the bus with system-level services including licensing, security, version control, persistence and transactions. Thus, many important low-level services, previously implemented by means of proprietary mechanisms, are now available as parts of off-the-shelf commercial middle-ware systems.

3.4 Accessibility

The experimental platform should provide a standard interface for access by *External Service Providers* (ESP). The *Parlay Group* (AT&T, BT, Cegetel, Cisco, Ericsson, IBM, Lucent, Microsoft, Nortel Networks, Siemens and Ulticom, Inc) focus on the production of an API specification that enables enterprises outside of the network domain to access network information and control a range of network capabilities. The published Parlay API has quickly gained popularity for this purpose, and is briefly described below. It should be noted that there are other alternatives to such an interface, however, we choose to base our discussion on Parlay. *Figure 2* shows where the Parlay API fits into the architecture. The figure and the description below are extracted from [10].

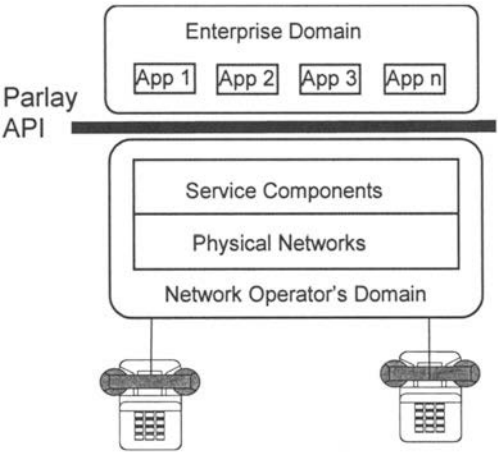

Figure 2. Parlay API

The Parlay API consists of two categories of interfaces:

1. **Service Interfaces**. These offer applications access to a range of network capabilities and information.
2. **Framework Interfaces**. These provide the supporting capabilities necessary for the Service Interfaces to be secure, resilient, located and managed.

Examples of the supporting functionality provided via the *framework interfaces* are: Authentication, discovery, event notification, integrity management, and finally operation, administration and maintenance.

Examples of *service* interfaces are: Generic call control service, INAP [1] call control service, generic messaging service, generic user interaction service, and call user interaction service (voice prompt to user, DTMF input from user).

A key to Parlay's success may be that the API is described using Microsoft and *CORBA IDL* [9]. This means that the API fits nicely into the overall requirements to the architecture of the experimental platform since standard open technologies are applied.

4. PLATFORM ARCHITECTURE

Based on the requirements identified in the previous section, we have developed an experimental platform for development of telecommunication services. *Figure 3* shows the experimental platform architecture. It interconnects through several interfaces to different kinds of access networks (currently SS#7, X.25, TCIP/IP and CORBA). The platform is designed to be both *scalable* and *fault tolerant*. This chapter describes the platform architecture and the middle-ware software used to interconnect the components of the platform.

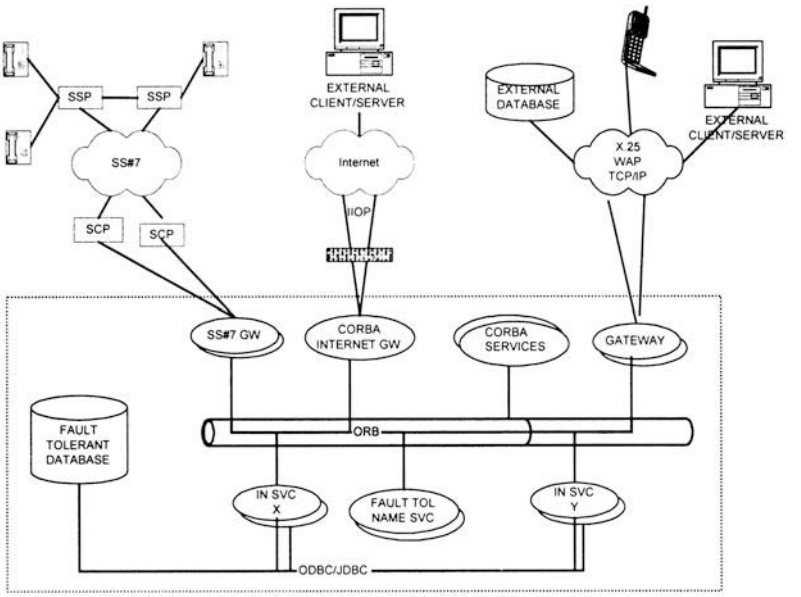

Figure 3. Experimental IN service execution platform

All items within the dashed line are part of the platform. Circles denote functional CORBA-aware units. Stacked symbols indicate replication, and replicas always run on separate physical hosts for fault tolerance and load sharing.

4.1 Overview

In *Figure 3*, gateways (GW) are generic (i.e. service independent) protocol converters, converting requests and responses to/from CORBA. Some of the gateways also have routing and firewall functionality, e.g. to overcome the Java "sandbox" limitations. The database is a standard SQL Server, but it is made fault tolerant by running it on an NT cluster. It is used by the services to store state between requests belonging to the same dialog. A fault tolerant name service uses "hot" replication based on group communication. Based on this platform we have implemented a variety of services, including:

- Traditional IN services (miscellaneous forms of call routing) with web interfaces for customisation.
- A service creation environment consisting of a generic set of building blocks for creating call centre solutions.
- An electronic commerce service, with external interfaces for customers, merchants and banks.

The functional units of the architecture of the platform are:

- **SCP:** The SCP implements a generic script which simply passes INAP requests to the SS#7 gateway. This design allows new service deployment without SCP MIB programming.
- **SS#7 gateway**: The CORBA-aware gateway converts SS#7 INAP requests to CORBA, analyses the message and hands it over to an instance of the appropriate replicated service objects. Upon reception of an answer from the servicing object, the reverse action is performed. Missing response from the requested object causes retransmission to another instance of the desired process, residing on another application server.
- **Middle-ware:** *CORBA* represents the platform middle-ware, i.e. the runtime environment or *Object Transaction Manager (OTM)* for our service implementation objects or *components*.
- **Application server**: A service object implements service specific logic. This facilitates the introduction of new services by deploying new service objects on the application servers.
- **Database server**: The internal database hosts data that needs to be stored during service execution, e.g. data about the state of service

progress. It can also be used to host data for a service that runs solely on the experimental platform, and therefore has no external database. The current implementation relies upon a Microsoft SQL Server 7.0 running on a fault tolerant NT Cluster.

- **Gateways**: Gateways are developed on demand. Of particular interest are gateways to:
  - The Internet.
  - Mobile terminals through WAP/WML.
  - External service providers.
- **Fault tolerant naming service**: The connections between Clients and Servers within the platform are permanent in order to avoid object binding for each service request. In the case of failure, some rebinding is required. The fault tolerant naming service enables rebinding of *all connections* upon failure detection instead of rebinding within a service request. This service is implemented by means of a replicated *supervisor* process. The supervisors implement the *Replica Control Protocol* [8] for group communication.
  - *Servant Objects* register to the replicated name service by providing *kind* (type) to identify redundant objects.
  - *Clients* request object *kind* in order to retrieve an IOR to any object of requested type, followed by an optional *register interest* message to the supervisor. Clients commit to reporting bad references to the name service, which in turn notifies all interested clients about the bad reference and provides a new IOR.

4.2 Satisfaction of Platform Requirements

Since we have already identified several requirements to a new platform for telecommunication services, it is of course relevant to discuss the proposed experimental platform according to these requirements.

- **Openness**: The platform is based on CORBA, which is a commercially available system for communication in a distributed environment. A standard language is used for defining object interfaces. CORBA is language independent.
- **Reliability**: CORBA provides mechanisms to handle application errors through e.g. distributed exception handling. Additional services to restart failing components may be implemented based on CORBA.
  - **Availability**: The fault tolerant naming service ensures that object references are valid, even in the case that the server fails. All servers are stateless. If a servant object needs to store state between invocations, it will use the (fault tolerant) database. Fault

tolerance will be easier to implement in the future as extended versions of CORBA will provide built-in support for this.
  - **Security**: CORBA provides basic mechanisms for secure communication by means of Secure Sockets Layer (SSL). Additional CORBA security has been specified, but is not widely available yet.
- **Distribution**: CORBA provides the necessary means to define clear interfaces in a programming language independent manner. Thus, components of various implementation languages may communicate. Components on different ORBs may communicate through the Internet Inter-Orb Protocol (IIOP). Thus, physical distribution is well supported.
- **Accessibility**: Gateways may be implemented to provide interfaces to additional access networks as well as external service providers. The versatility of CORBA makes it possible to implement the gateways in a number of languages.

We feel that the last item concerning interfaces to the service platform is one of the most important aspects of a modern service platform. Thus, we discuss this topic further in the next section providing examples of how the travel agency service may be provided as an external service interfacing to the new service platform, and how the service may easily be extended to support additional access networks.

5. EXTERNAL INTERFACES

This section describes how adding various interfaces to the platform may extend the capabilities of the travel agency service introduced in Section 2.

5.1 Functional Extensions of the TA Service

A possible extension to the service is to add capabilities for online payment processing. Thus, a customer can choose to register himself with the travel agency by providing information about credit card numbers, bank accounts etc. This therefore becomes an extension to the previously existing customer profile.

After the conversation with the agent is finished, the call can be transferred back to the TA service, and a payment dialogue can be initiated. This assumes that there is a connection between the travel agency computer and the service, to let the TA service be notified about the payment details. An architecture supporting this functionality is depicted in *Figure 4*. Here, the TA service provides more functionality than the service introduced in

Section 2 since it also has to support processing of payment transactions and communication to external banking services etc.

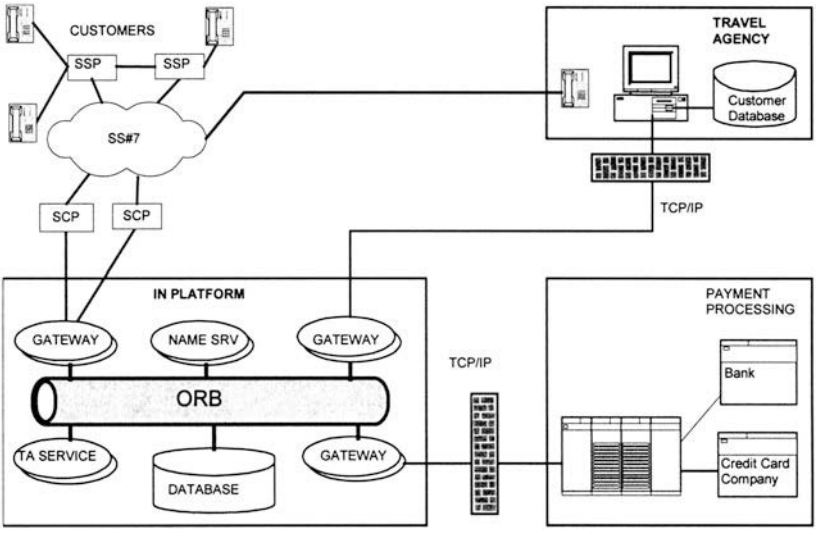

Figure 4. Extended functionality of the TA service

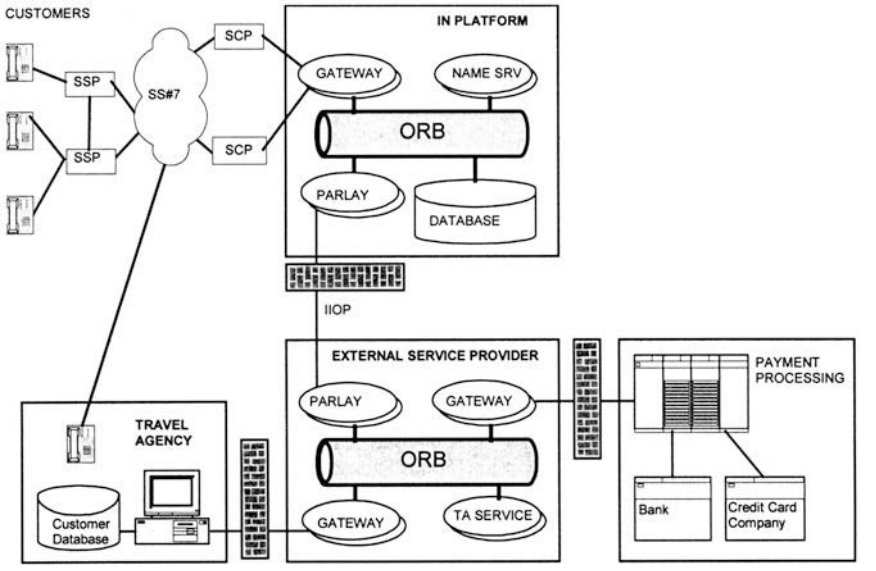

Figure 5. Parlay/IIOP Interface to external service providers

5.2 External Service Provider

The overall architecture for a service platform providing access to external service providers (ESP) is shown in *Figure 5*. The interface is built according to *Parlay* on top of CORBA, thus requiring a CORBA-enabled external platform.

The Parlay interface allows service requests to be passed to the external service provider. Similarly, the ESP is able and allowed to perform PSTN switching. The entire TA service may in this way be developed and maintained by the ESP. In the architecture above, we assume that the payment processing is also an external service with a clear and secure interface to client services. The payment processing service, however, does not need to interact directly with the service platform, and may therefore communicate with the caller through the TA service.

5.3 Gateways to Additional Access Networks

If we concentrate on the upper half of the architecture in *Figure 5*, we will see how the service platform can be extended with gateways to additional access networks. The important point here is that the presentation layer is separated out and made part of the service platform, while the service itself, which is still provided externally, remains unchanged. This separation of the service into different layers is similar to the approach often used in traditional software development projects, where an application is divided into three tiers: *User interface*, *business logic* and *database*.

To illustrate the addition of gateways to access networks, we can consider providing the service for Wireless Application Protocol (WAP) enabled mobile telephones. Adding a gateway to other networks such as the Internet would be quite similar.

The functionality of the service remains basically the same. When using a WAP phone, the customer places a call to the toll free number, and is exposed to a menu presented textually on the telephone. An option is chosen, and the call is placed in a queue in the same way as for PSTN. Then, the connection can be terminated, and the customer can wait offline. When the call is ready to be serviced, the customer is notified through WAP notification, and may choose to answer or abandon the call. If answering, the conversation with the agent takes place, and finally the payment transaction can be carried out using textual menus displayed on the telephone rather than voice messages. An architectural design supporting this solution is depicted in *Figure 6*.

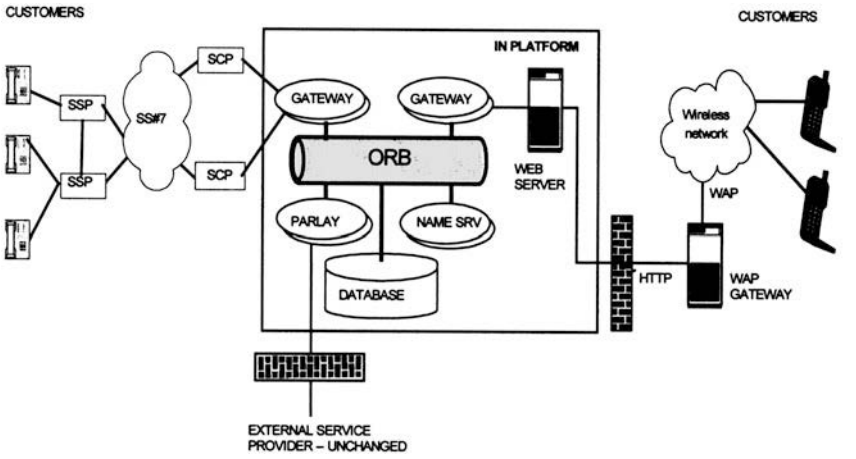

Figure 6. WAP Interface to mobile users

A web server is added to the service platform to run servlets providing the link between the WAP gateway and the CORBA objects of the service platform. The servlet generates output written in Wireless Markup Language (WML) which is eventually presented on the telephone. The WAP gateway is a protocol converter that converts WAP to HTTP. A firewall protects the service platform against unauthorised intrusion.

6. CONCLUSION

We have identified and argued that a new, experimental platform for the development of telecommunication services should be: *Open*, *reliable*, *distributed* and *accessible*. An experimental platform has been established according to these requirements, and initial experiences indicate that this is a step in the right direction. However, more experience is needed, particularly concerning security, before the platform can be used for supporting commercial services.

We have been experimenting with various services to identify common functionality to be considered part of the platform, and have thus implemented a fault tolerant naming service to handle failure of CORBA objects. A related service could be implemented to support error detection and –handling to provide a more flexible, generic and component-independent handling of failures. Report generation and logging could also be supported by this service.

Current efforts are being made to investigate an extension of the platform to include additional access networks, in our case a mobile network carrying

WAP. Our experiences so far are that the solution indicated using a servlet connection is flexible, simple and reliable. The hard part is to make the presentation layer generic to avoid tailor-made user interface components for each service.

In parallel, we are experimenting with external services interfacing to the service platform using Parlay. For traditional PSTN-based services, Parlay appears to provide the necessary functionality for call control, switching etc. The weakness is the lack of support for including service-specific interfaces, which is quite cumbersome at the moment.

Combining the two efforts, thus implementing an external service interfacing to a service platform providing access to multiple networks, revealed interesting problems. Parlay appears to be a rather cumbersome solution for services accessed by a network based on data traffic (i.e., not speech). In particular, online communication, which would be done through conversation using PSTN, is hard to achieve due to the missing support for service-specific interfaces in Parlay.

REFERENCES

[1] European Telecommunication Standards Institute, Network Aspects 6: Intelligent Networks, www.etsi.fr, 1992

[2] International Telecommunication Union – Standardization Section, Q.1200-series, Intelligent Network Recommendation, www.itu.ch, 1993

[3] Telcordia (Bellcore), Advanced Intelligent Network Requirements, www.telcordia.com, 1992

[4] H. Zuidweg, P.Quentin, G. Reyniers, E. Devleeschouwer, B. Quirynen: A Distributed CORBA-Based IN Architecture, 4th International Conference in Networks, Bordeaux, November 25-28, 1996

[5] Kenneth P. Birman: "Building Secure and Reliable Network Applications". Manning Publications Co. Greenwich, CT, 1996. ISBN 1-884777-29-5

[6] Object Management Group, 1996, "CORBA Security Service Specification".

[7] Gray J. and Reuter A.: "Transaction Processing: Concepts and Techniques". Morgan Kaufmann, 1992

[8] AMR EL ABBADI, DALE SKEEN & FLAVIU CHRISTIAN, *An Efficient, fault-tolerant protocol for replicated data management*, ACM 0-89791-153-9/85/003/0215, 1985

[9] PARLAY Industry Working Group: CORBA IDL. http://parlay.msftlabs.com/

[10] PARLAY Industry Working Group: Parlay White Paper. http://parlay.msftlabs.com/