Validation of the Parlay API through prototyping

J.W. Hellenthal
Lucent Technologies
P.O. Box 18
1270 AA Huizen NL
Whellenthal@lucent.com

F.J.M. Panken
Lucent Technologies
P.O. Box 18
1270 AA Huizen NL
Frans@lucent.com

M. Wegdam
Lucent Technologies
Capitool 5
7521 PL Enschede NL
Wegdam@lucent.com

***Abstract*- The desire within the telecommunications world for new and faster business growth has been a major drive towards the development of open network APIs. Over the past 7 years several (semi) standardization groups have announced to work on network APIs, including TINA-C, JAIN, IEEE P1520, INforum, 3GPP, JAIN, Parlay. The Parlay group seems most successful in attracting industry awareness with their API, called the Parlay API.**

The rational behind the Parlay API is that it attracts innovation from third parties that are outside the network operator's domain to build and deploy new network hosted applications. This also means that the public telecom network is opened for niche and short-lived applications as well as for applications that possibly integrate telephones with other terminals such as PCs.

The Parlay group has successfully passed the first two phases of success, namely publishing their API on the right moment in time and attracting a critical mass within the telecom industry with their results. Prototyping the API on a real network execution platform is the only manner to show its technical feasibility. Such an exercise was executed internally within Lucent Technologies and raised a number of questions as well as recommendations on both the technical and the semantical behavior for systems that will be interconnected via the Parlay API. In this paper we share these results, showing draw-backs, advantages of as well as challenges for this API.

I. INTRODUCTION

Recent industry trends and initiatives for open Application Programmable Interfaces (API) indicate that applications may increasingly be operated by companies who operate from outside the traditional network boundary. These applications will interact with network resources to provide novel services to customers. Although the network operator still owns and operates the network resources, the actions requested and the level of demand on the network's capabilities will be driven by applications written and maintained by independent software vendors, and will consequently run outside the domain of the network operator. The network operator will not have access to the detail of the 3rd party applications nor can the operator test or simulate these applications in the same manner as traditional applications that run and are deployed inside the network boundary.

Service providers, at their turn, have access to a restricted set of network resources that allow them to offer their services, but the way these services are realized (i.e., the protocols used or the specific equipment applied) will be completely concealed by the network operator.

The service providers experience the network as a programmable telecom environment, whereas the network operator look upon services as applications that are executed on their network.

Despite the lack of knowledge about the behavior of the application and the shortage of testing, the network operator still needs to ensure that services offered via the services interface will not endanger the network's integrity. Service providers have the responsibility to meet the service functionality they have agreed to their customers and can only do this with the capabilities offered by the network operator. The capabilities provided by the network operator - and the conditions when these capabilities can be used by the service provider - are captured in a contract, known as a Service Level Agreement (SLA). In order to prevent that either deliberately or inadvertently applications controlled and managed by a service provider put the network in an unsafe region, the network operator must verify whether invoked methods are compliant to the SLA. Based on the exceptions raised by the network operator, the service provider can check whether the resources agreed to the service provider in the SLA are met.

Comparing the above discussion to the widely applied network interfaces such as the User Network Interface (UNI) and the Network Network Interface (NNI), we observe that these APIs realize a new concept of interface, namely an interface that realizes the interworking between services and network. The first standardized interface that realized an almost similar goal is the Advanced Intelligent Network (AIN). AIN separates service development from switching, allowing service logic to be developed more quickly and executed in specialized network elements attached to data bases, optimizing PSTN switches for speed and efficiency. The Parlay API (see [1]) goes beyond this approach and specifies an API that offers application developers for the telecom network similar methodologies as used in software and information technology in general, offering customers the benefits of increased competition, reduced time to market and rapid leveraging of new technologies.

To guarantee interworking, the UNI and the NNI are standardized for various transport networks. However, all network vendors have proposed different service interfaces on their products, resulting in a wide range of (open, but) proprietary service interfaces and the lack of interoperability. Applications written for the networks of a specific operator cannot seamlessly be offered to another operator's network, who has bought their equipment from a different vendor. Previous efforts to standardize a service interfaces have failed.

New techniques such as e.g. the Interface Definition Language (IDL) make it possible to specify service interfaces in a programming language independent manner, without loosing

0-7803-7047-3/01/$10.00 (C) 2001 IEEE

the ability to exploit advances from the object oriented programming languages. A standardized services interface allows software vendors to write their applications according to one common information model, allowing a true telecom services synergy where the application only needs to be written once, and can be applied to various transport networks with a minimum of effort. To reach the widest possible range of market players that may eventually develop applications on top of the Parlay API, the Parlay group has defined their API for two middleware platforms: COM and CORBA. For our prototype we used CORBA. Since the project started on January 2000 and at that time the IDL specifications of the Parlay 2.0 API were not available, we made our own IDL, using the Parlay v2.0 documentation as input. In this paper we reveal the experience lessons learned from prototyping the Parlay API on top of the Lucent softswitch, version 1.19. All results gained during this forward looking project were transferred to the development organization where they were used as input for the development of the Parlay API on top of one of Lucent's next generation products: the Lucent softswitch v3.x.

This paper is structured as follows. Section 2 summarizes the experience lessons learned from our effort to prototyping the Parlay call control API both from the application and the server sides. Section 3 reports on the remaining challenges we foresee for the Parlay API, whereas in Section 4 we discuss what is needed for the Parlay API and we enumerate suggestions for improvement, based on our implementation results.

II. PROTOTYPING EXPERIENCES

In this section we first explain the essential parts of the Parlay API and subsequently reveal the implementation details discovered during a relatively small project, called Parlay integration on the Lucent SoftSwitch (PLUSS). The PLUSS project was carried out from January 2000 until April 2000, consuming two technical headcount.

A. On Parlays' framework and call control

The Parlay API consists of two parts: a framework and service capabilities.

The Parlay framework describes the information flows necessary between the domain of the Independent Software Vendor (ISV) and the domain of the network operator. These information flows describe the procedures that need to take place before an ISV can offer its services to end-users, connected to the domain of the network operator. From an ISV's perspective, the Parlay framework can be considered the bridge to gain access to the Parlay service capabilities offered by the network operator. From the network operator's point of view, the framework is the phase that *should offer* a secure context to outsource network services safely.

When the client has gained access through the Parlay framework, the interface to particular Parlay services can be obtained. Parlay identifies 5 distinct services: call control, user interaction, messaging, connectivity manager and mobility. In the PLUSS project we have implemented the Parlay framework and the generic call control service. Parlay's call control service lets the application set criteria that must be met before the applications are informed. These criteria mainly exists of ranges of network addresses for which the application desires to monitor network behavior. For each different service capability, there is a Parlay call control interface that allows the programmer to set simple and complex behavior that is related to that particular service capability.

The implementation of the Parlay framework was kept to a minimum, without making use of e.g., encryption. A quick glance at the Parlay framework is enough to see that security is not solved to a satisfactory level by the Parlay API. Since in other projects we already gained experiences with the enterprise domain where we could register for services etc, we limited ourselves in PLUSS to the interaction of the Parlay client and server sides. All methods that are part of Parlay's generic call control service were implemented, resulting in a situation where the application side can register for network events, can set up connections and receives and subsequently reacts on events coming from the network via the Parlay API.

B. The application side

Two client applications were implemented to validate the Parlay API, namely third party call setup and number translation. The reason for choosing these applications is the ability to validate both calls originated in the network and calls originated in the third party application domain. To apply the two applications, a common framework was implemented, following the Parlay framework interactions: authentication, selecting a service, and signing of the service agreement. Having done so successfully, the client application gets a reference to the IpCallControlManager, and the two applications can use the call control service. For demonstration purposes, the client application is implemented as a GUI.

Figure 1 Screenshot of the 3rd party call setup service

For the third party setup all that is required is to type two numbers, see Figure 1. The client application uses the IpCallControlManager.createCall() method to ask the call control service to initiate the call.

Figure 2 depicts the screen of the number translation application. This application asks the call control service to be notified about calls with address ranges. The address ranges can be set dynamically. This is realized by invoking the method IpCallControlManager.enableCallNotification() as soon as the *send* button is pressed. Every time a call within these ranges is initiated the call control service of the server side sends a notification to the client application by means of invoking the method IpAppCallControlManger.callEventNotify().

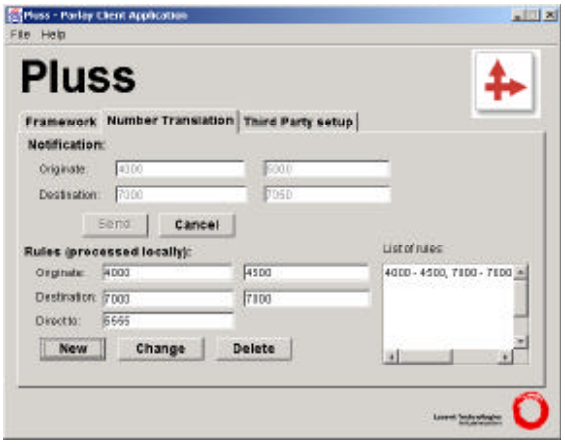

Figure 2 Screenshot of number translation service

The client application can subsequently decide what to do with this call, implemented by the creation of simple decision rules. Several rules can be created but they all use the same notification settings. Figure 2 shows the situation that every call to number 7000 that is originated from an address in the range 4000-4500 should be connected to number 5555. Both the number translation and the third party call setup applications receive notifications about the success or failure of the requested network resources (e.g. the destination address is busy, party answered, no answer) such that appropriate actions can follow. When the call is successful, Parlay progress reports inform the applications about the status of the call (e.g., call ended,..).

C. The server side

The network execution platform used to gain experiences with the call control part of the Parlay API is the Lucent softswitch, version 1.19 [see 2]. This call engine offers a number of network capabilities, including call coordinator, resource servers and device servers.

The **call coordinator** (CC) maintains the state of all calls and coordinates the communications between device servers and resource servers. The CC does therefore not only represent the call model of the call engine, but also provides connection control and is also involved in the establishment of the media paths between resource servers and end points. The CC also offers an API for communication with the internal services API. Finally, the CC maintains a hierarchical namespace for each call routing calls by making use of a service dial plan strategy which allows to associate phone numbers with IP addresses.

Device servers are linked with the CC and offer the Lucent softswitch with the traditionally functionality of signaling gateways. There are multiple device servers, making translations between signaling protocols such as H.323, Q.931 and MGCP to an internal call layer, called Mantra. For the PLUSS project we used the H.323 device server and as H.323 end points we used Microsoft's netmeeting as and an IP Web phone that is based on a modified IS2630 web phone. The latter was provided to us by the SPEED and FLW department in Naperville, Illinois (see[3]).

Resource servers offer media resources (tone detection, playing or recording announcements) to end points. Lucent offers a wide range of resource servers; for our experiment we used a software resource server implementation that is a version of the Elemedia programmable media server.

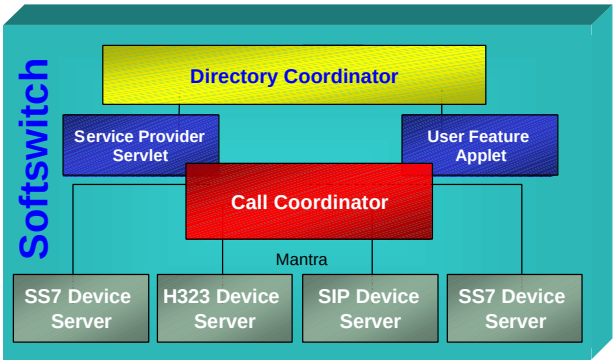

Figure 3 Architecture of the call engine used

On top of the call coordinator, the Lucent softswitch v1.19 offers a programming interface, called Service Provider Servlet (SPS), that provides the capabilities to address standard call processing functionality and allow this functionality to be overwritten at the application level. The network side of the Parlay API is programmed on the SPS, intervene triggers in the call model and forwarding Parlay events via CORBA to the applications. The SPS can interfere at various points in the call flow. Examples points where the SPS can decide to either interfere or to continue normal processing include:

1. onInitCaller
2. onInitCallee
3. onCalleeRouteSelect
4. onCallProceeding
5. onCallAlert
6. onCallBusy
7. onCallConnect

These capabilities are broad enough to support all the requirements needed to offer the Parlay call control capabilities. The combination of resource servers with the call allows for the support of another Parlay service capability, namely the user interaction.

In the number translation application, the CC starts processing upon reception of a *CallRequest* originating from the H.323 device server or the web phone. It will generate an onInitCaller event used by the SPS to create a Parlay IpCall object, representing the new call and a call leg object for the calling party. The next event generated by the call coordinator is onInitCallee. This event is used by the SPS to create a call leg object for the called party and to determine whether or not the caller and callee address match the criteria sent by the Parlay application by means of the enableCallNotification method. If the criteria match, the SPS will then invoke the Parlay callEventNotify method that triggers the application to perform some specific tasks. In our case, number translation. Note that in the case the criteria match the call processing will be suspended and the Parlay application is responsible for resuming call processing. What follows is the application translating the number of the called party and sending a routeReq on the IpCall object. This will result in a new connection setup to the party with the translated address. The call coordinator will now generate events indicating the callsetup state for the called party, e.g. onCallProceeding, onCallAlert etc… These states are intercepted by theIpCall object and translated into routeRes invocations to the application. Note that the application has registered for specific route results like sendRefusedBusyReport and sendCallEndedReport when invoking the routeReq method on the IpCall object. If the SPS does not find a criteria match, it will continue with standard call processing without informing the application side.

III. REMAINING CHALLENGES FOR PARLAY

The success of the Parlay API - as it was intended - depends on the close interworking of three parties, namely

1. Software developers whose applications need to interface with the network according to the information flow described by the Parlay API.
2. Network equipment vendors, who have to implement the parlay server side on top of their products
3. Network operators, who have to realize the services that are offered by the software developers, and offer these services to their customers.

We expect a smooth evolution path in the usage of the Parlay API and identify three milestones on this path. First, the API will be offered by the vendors on their products and vendors offer services that exploit the capabilities of this API. This can lead to a common API for various services, making it possible to offer services seamlessly to different transport networks. In order to realize this, telecom vendors need to be convinced that Parlay is the API that targets the desired functionality and can indeed abstract from all the network details where the service creation platforms for the PSTN are so familiar with. Parlay can play an important role in integrating the Web with the traditional phone network. Examples of services for this era include click-to-dial applications where the browsing customer can request to speak a consultant about a certain product or where customers are a single click away to conference-in experts on a (mobile) phone conference.

The second milestone is reached when network operators become familiar with the API and the product that supports it and subsequently take the initiative to develop service by themselves. By that time, network operators will develop new services that will add value to their network and raise revenue, or ask ISVs to develop these services. During this era, the specification and description of the services originate from the network operator and will also be deployed and executed from resources owned by the network operator. To realize this milestone, network operators need to be convinced that the Parlay API does provide added value to the existing set of Computer Telephone Integration (CTI) APIs and that their network will not be put – either accidentally or on purpose - in an unsafe position. The latter becomes important since during this era it is not the vendor but network operators who will become responsible for malfunctioning of the network as a result of service execution. Examples of services can be small services where operators can distinguish themselves from others by offering new applications. The possibility to offer short living and niche telecom applications has begun.

The final milestone is reached when ISVs combine their various services with network resources and use the telecom network as bandwagon to make different devices and /or applications on these devices to interwork with one another by making use of the Parlay API. By the time this milestone is reached, both the vendors and also network operators need to become involved in solving specific services requirements. Vendors will need to offer the network requirements defined by the ISVs, in order to realize that ISVs can sell their applications. In the era towards the third milestone it becomes essential that network operators are convinced that the applications cannot and will not put the network in an undesired position. In addition, the network operator will ask for guarantees that existing application are not damaged or do not interfere with those offered by the ISV (rogue applications must be banned, but also miss-use of resources must be prevented). Essential for the occurrence as well as the success of this third milestone are the consistence of measurable and billable service level agreements between network operators and ISVs. After all, the business model to realize the third milestone simply depends on making money in a controlled manner that does not risk the continuation of the existing money generating products or services.

Lucent Technologies has launched its Full Circle program (see [4] and [5]) to support development communities. This third party partner program also includes the Parlay API and can as such function as the glue that is necessary to bridge ISVs and network operators to realize the third milestone. The APIs of the full circle program will be offered on various Lucent products, including the Lucent softswitch and the Packet IN application server.

The necessary condition to realize the first milestones is that the Parlay group attracts the critical mass within the telecom industry with their API. This has happened, resulting from the fact that various vendors have announced to imple-

ment the Parlay API on their products. From this we conclude that Parlay has successfully passed the first milestone. Attracting the critical mass, however, is only a necessary condition for success. At the technical level, the API needs to be specified clearly and unambiguously, needs to be under control by a standardization organization that has clear rules with respect to changes and update and as well as the guarantee for backwards compatibility, is semantically correct and addresses the correct middleware platforms. If Parlay fails in realizing one of these technical criteria, each vendor will make its own flavor of the Parlay API to meet their needs. This prevents interoperability and consequently prevents the attraction of ISVs since it limits the re-use of software and, as a result, lowers the level programmability of the controlled telecom network. As a consequence, we expect that if this occurs, the evolution path for the Parlay API as described above will not reach the third and final milestone.

The time necessary to pass each of the inter-milestone eras strongly depends on the acceptance of the Parlay API by the various players and whether the involved parties recognize the potential business model of the Parlay API. The outcome of various trials on the validation of this API that currently take place between network operators and telecom vendors will influence the realization of the second mile stone.

IV. SUGGESTIONS FOR IMPROVEMENT

During the prototyping experience we have discovered certain shortcomings and inconsistencies of the Parlay API. In this section we summarize these findings, limiting ourselves to those that were not fixed in Parlay v2.1.

D. General improvements

Architecture

In every document the general Parlay architecture is described, making use of the figure as depicted inFigure 4.

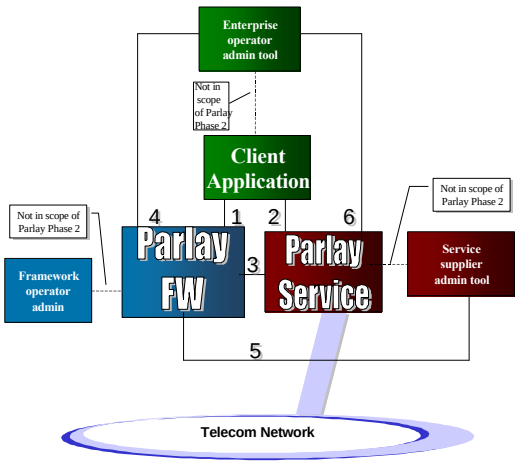

Figure 4 Architecture as depicted in Parlay documents

The explanation of this architecture needs improvements. Each of the 6 reference points should clearly describe the minimum set of interfaces to be supported in order to be compliant to the Parlay API. A suggestion for this reasoning is depicted in Figure 5.

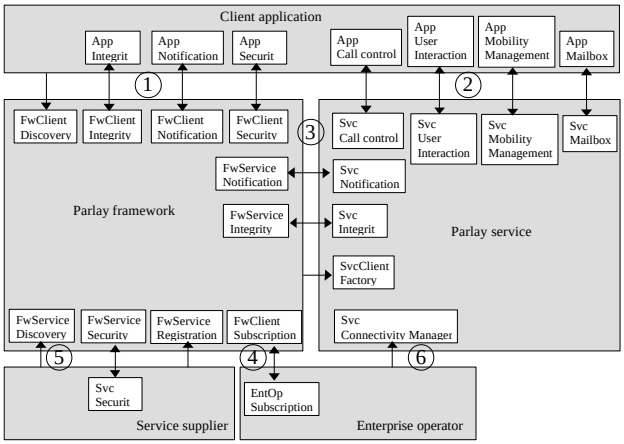

Figure 5 Suggestion for structuring of the Parlay architecture, revealing component level. The numbers correspond with those used in Figure 4.

Poor support for middleware services

The Parlay group has constructed a UML model and has used this model to construct the IDL and MIDL specifications. This saves time but also leads to a situation where the specifications do not exploit all the services offered by the target middleware platform. Examples include the way Parlay deals with exceptions and the void return value for each and every method. Exceptions is a powerful and broadly excepted way to indicate what has gone wrong, or which parameter was for instance missing and should be better exploited in the Parlay specification.

Inconsistency between documents and specification

Various inconsistencies appear when comparing the Parlay v2.1 IDL files with the description given in the Parlay documents. Examples range from missing functions in the IDL file (e.g. recordMessageReq()) via different types definitions (e.g., TpAddressplan) to different names or typos (e.g., TpStringRef). These inconsistencies probably stem from the pressure and time limits the engineers need to work, but raise the undesired discussion what is correct.

Finite state machine

Currently, the Parlay API consists of a set of IDL files that describe all the different methods and their parameters that are supported by the application and server sides. Just as the traditional (signaling) protocols, there is a need for a finite state machine that describes in detail the sequence in which methods can (or should) be invoked, and what to do if e.g., one of the sides receives an asynchronous method correctly, but does not respond? The existence of a description of the Parlay API in terms by means of a finite state machine also contributes to solving interoperability problems.

E. Improvements for the Parlay framework

Discovery and security

The improvements for the Parlay framework mainly stem from the poor support for the services offered by the targeted middleware platforms. CORBA's trader service makes the Parlay discovery interface and services obsolete, whereas CORBA security could have been used to solve security as well as Parlay's attempt to solve non-repudiation in the method SignServiceAgreement(). From a technical point of view it would have been better if the Parlay API describes in detail how the CORBA security service should be used in order to be compliant and how to map e.g., binary strings to their ASCII representations.

Heartbeat

The Parlay heart beat should either be positioned as part of the IparlayInterface or should be replaced by heart beat functionality offered by the middleware platform.Parlay's current heart beat consist of an object that is brought to life for the sole aim of sending *I'm alive* messages.

F. Improvements for Parlays' call control service

Missing call leg ID in enable call notification

The call notification informs the application that previously set conditions have been met. The application is expected to inform the network how to continue and needs to base its conclusions on the information passed by the IpAppCall.enableCallNotification(). In contrast to other call related issues, this method does not contain an ID of the establish call leg, preventing the application to refer to this call leg in a latter stage.

Ability to relate call objects with one another

Parlay prescribes that when previously set criteria are met, the server side informs the application by means of the method IpAppCall.enableCallNotification(). This could for instance happen if one of the end points calls a specific number and this number meets the above criteria. Currently it is not possible to instruct the network that the freshly called endpoint should join an existing call.

Bridge between call control and user interaction

One may think of applications that need both the call control and user interaction service capability. The Parlay v2.0 and v2.1 specifications have introduced the IpUICall to realize this. This is a simple object that inherits from IpUI and support a limited number of methods, such as recordMessageReq(). The Parlay documents show that by a passing the call-leg ID to the IpUICall object the application can indicate to which call leg the announcement should e.g. be played. However, this can only be realized if the call leg was initiated via the call control service. Just as the call control service capability, the user interaction service also supports a method to set criteria. If these criteria are met, and the application is notified about them via the invocation of the method IpAppUiCall.enableUINotification(). However, the application can never continue by requesting connections because the call control capability does not allow to bridge the UI.

Ability to see that terminals have registered

By studying the network events supported ('busy', 'on-hook', 'off-hook'), one may conclude that the Parlay API has a strong AIN flavor. Requirements such as 'terminal has registered' or 'user has registered' that stem from the multimedia protocols such as H.323 or SIP need to be supported.

V. CONCLUSIONS

The Parlay API is a promising API that has the potential to become the standard services API. Lucent's Full Circle development support program has targeted the Parlay API to be supported for application programmability. As a result, various products from Lucent Technologies will offer this API. In this paper we have shown that the rich set of events generated by the network can be abstracted by the Parlay API by implementing two services. The services are constructed such that one originates from the network (number translation) whereas the other service originates from the application domain (third party call set-up). Nevertheless, we have enumerated potential dangers that may prevent the Parlay API from reaching the level of international acceptance as services API as it was constructed for. Finally, the shortcomings found as well as suggestions to solve them were described. Hopefully these recommendations are reflected in Parlay 3.0.

ACKNOWLEDGMENT

We thank all contributors who made the PLUSS project to a success, namely Suan Lie and the members of the SPEED and forward looking work department of Lucent's Switching Solutions Group in Naperville, Illinois, USA: Olivier Clarisse, Pascal Collet, Bruce Westergen who were so kind to provide us with a web phone. We kindly thank Harold Batteram for his technical support and for his help in the construction of Figure 5 and Chinh Pham for reviewing this paper and providing us with suggestions for improvements.

REFERENCES

[1] Parlay Group, "Parlay white paper", <http://www.parlay.org/docs/Whatis.pdf>

[2] R.A. Laksmi-Ratan, "The Lucent Technologies softswitch – realizing the promise of convergence", Bell Labs Tech. J.,Vol.4, No.2, April-June 1999, pp.174-195

[3] O.B. Clarisse, P. Collet, J.P. Dunn, B.A. Westergren and L. Valezquez, "Portal services: an evolution of voice features", Bell Labs Tech. J., Vol.5, No.3, July-September 2000, pp.203-215.

[4] C.L. Bear, W.A. Montgomery, D.B. Nolte, S.K. Rajchel and M.C.Silva, "Open programmable networks", Bell Labs Tech. J., Vol.5, No.3, July-September 2000, pp. 30-42.

[5] Full circle vision 360, <http://www.lucent.com/full_circle>