Development and Implementation of a Third Party Call Parlay X API for Application Development in NGN

Rodrigo Pimenta Carvalho & Antônio Marcos Alberti

Abstract— **The search for technologies to develop value-added services in Next Generation Networks created what is being called Information and Communication Technologies convergence or ICT. The success of such convergence will depend on, among other factors, adequately opening NGN capacities to support value-added applications development by IT community. Open APIs such like OSA/Parlay and Parlay X are being considered the "bridges"between both worlds. Despite such APIs, there are other technologies and several possible paths involved in this task. In this scenario, our paper firstly shows how next generation converged networks and its functionalities can be exposed properly to someone else, through Parlay X interfaces, implemented in Web-Services-based application gateways. Therefore, this paper cames to fulfil the lack of literature regarding methodology for value-added applications development in NGN. Secondly, we developed and implemented a Third Party Call Parlay X API inside a Parlay X gateway emulator to demonstrate how useful a gateway emulator could be in order to provide quick and correct design and implementation of NGN applications. Finally, we discuss the results obtained.**

Index Terms— **Service Creation, Value-Added Services, Web Services, Parlay X and NGN**

I. Introduction

The technological convergence toward an *Internet Protocol* (IP) network is being considered a landmark without return. Such convergence allows more flexibility, development speed, money saving, services diversification, interactivity and network access spray, besides a whole world of new opportunities. Looking at this scenario, and concerned with the invoices reduction, telecommunications companies have fomented new technologies development, able to modernize their services staffs and applications. One of the main proposals, according to this, is the exposure of the telecommunications networks services and capabilities through gateways implementing open *Application Programming Interfaces* (APIs), such as: Parlay/OSA [1] and Parlay X [2].

The aim is that, through those APIs, a significant amount of software developers will get together with the Telecommunication industry, bringing new and creative applications. Currently, the experienced professionals in Information Technology, with the software development technologies, such as Java [3], *eXtensible Markup Language* (XML) [4], *JavaServer Pages* (JSP) [3], and Web Services [5] complete the new companies outsourced job's environment, because they, themselves, bring knowledge, that are not in the Telecommunications professionals' domain.

On the other hand, *Information Technology* (IT) professionals seeking entering the Telecommunications market, but without having skills about the technologies from this market, need the common interfaces, which make possible to request telecommunications services. Therefore, the APIs that describe the companies capabilities (and the domain about using them) are the "bridges"between the IT and Telecommunications worlds. It is the IT/Telecom or *Information and Communication Technologies* (ICT) convergence. We will discuss ICT convergence on Subsection I-A, presenting how both worlds could be integrated using application server gateways in a next generation network scenario.

The motivation for this work arose when we became interested in *Next Generation Networks* (NGNs)[6]. More specifically, we were interested in developing *Value-Added Services* (VAS)[1] for standardized NGNs, such like *International Telecommunication Union - Telecommunication Standardization Sector* (ITU-T) *Next Generation Networks Global Standards Initiative* (NGN-GSI) [7][8] and/or *3rd Generation Partnership Project* (3GPP) IMS [9]. At that time, the main difficulty was to find out which path to follow, since there is a great quantity of possible technologies involved in this task, not only in IT world, but also in telecommunication world. Therefore, we studied several candidate technologies for NGN and IMS applications/services development in a 2007 [10]. In Subsection I-B we briefly presents such technologies in the context of this work. This 2007 study pointed that it was necessary a new investigation about how NGN functionalities and resources could be exposed properly to IT domain. In this paper, we present the path we follow to reach convergent networks service stratum, i.e., to use standardized Parlay X APIs, through Parlay X interfaces, implemented in Web-Services-based application gateways. The paper aims to fulfil the lack of literature regarding methodology for NGN value-added services development.

With Parlay X APIs in mind, we started to choose a service to implement. Of course, it must be a service that could allow interesting applications development in IT world. But also, we were interested in invocate such service using NGN functionalities and resources at service stratum. To

Manuscrito recebido em 21 de setembro de 2010; aceito em 29 de dezembro de 2010.

R. P. Carvalho (pimenta@inatel.br) and A. M. Alberti (alberti@inatel.br) are with Instituto Nacional de Telecomunicações - Inatel. Av. João de Camargo, 510 - Santa Rita do Sapucaí - MG - Brasil - 37540-000.

[1]Examples of value-added services are instant messaging, on-line gaming, push-to-talk, video push and pull, click-to-dial, presence, location, etc [26].

complete the requirements, we considered the fact that we didn't have a real NGN environment ready to test Parlay X APIs applications. Gateway emulation appeared as an option and we decided to implement *Third Party Call* (TPC) service because it satisfied our requirements as well as it wasn't yet implemented in available emulators. Thus, we explored Parlay X TPC API with gateway emulation, showing and discussing the necessary technologies to their use, besides contributing for the development of a gateway emulator from Ericsson Company [11]. As we will see, gateway emulation proved to be a very useful tool to merge IT and Telecom worlds, allowing quick prototyping and testing, without the need of using a full operational convergent network test-bed all the time.

The remaining of the paper is divided as follows: Subsection I-A discusses Information and Communication Technologies Convergence, presenting how both worlds could be integrated using application server gateways in a next generation network scenario; Subsection I-B briefly presents the technologies available to implement value-added services in NGN; Subsection I-C presents related works regarding Parlay X application development for NGN and IPTV; Section II presents development and implementation of a Third Party Call Parlay X API on a gateway emulator. We discuss the methodology used and obtained results; Finally, in Section III we draw some conclusions and final remarks about the work.

A. Information and Communication Technologies Convergence

Information and Telecommunications technologies are experiencing significant changes motivated by Internet success, new users behavior, technological convergence and new paradigms, such as context awareness, usability, user-centric design, networked electronic media, among others. In addition, there is the importance of such technological platforms for general companies, government and other institutions. Such technologies had become a strategic part of their business. Also, ICT convergence is playing a fundamental rule on Future Internet design. Telecommunications operators need this technical convergence to quickly create ingenious value-added services no matter what the business model is being used.

The financial question related to the maintenance of operator revenues by means of innovative services, became so excellent that agencies of standardizations, as ITU-T [8], 3GPP [12] and *European Telecommunications Standards Institute* (ETSI) [13], have defined standards for the architecture and services support on NGNs. Evidently, such applications development will depend on the facilities provided by telecommunications network operators and how these facilities are exposed to software developers. To facilitate the sprouting of these applications, a solution is to define standardized APIs to access telecommunications capacities and services. For this, however, a reorganization into networks infrastructure was necessary, creating a separated layer for services invocation, conceptually decoupled from transport network.

The current ITU-T convergent networks standardization effort is called NGN-GSI [7], [8]. It focuses on the development of detailed standards, necessary for NGNs development. The generalized functional architecture proposed for NGN-GSI separates service control and application/service support functions from transport control and transport functions. The functional separation on service and transport stratus allows independent software designing, without concerning to low-level details, like transport technologies, protocols, traffic management, security and configuration. NGN-GSI transport is based on IP protocol and includes sophisticated *Quality of Service* (QoS) support.

NGN-GSI service stratus contains a set of Application/Service Support Functions where several types of gateways could be presented in order to expose NGN architecture to Third Party Applications. More specifically, there is a functional element called A-1: *Application Server Functional Entity* (AS-FE) which supports *Open Service Architecture* (OSA) application servers [14]. This element interacts with S-1: *Serving Call Session Control Functional Element* (S-CSCF) in order to invoke telecommunication services using *Session Initiation Protocol* (SIP) [15] sessions. Therefore, when an application needs some convergent network service it needs to interact with one or more of these Application/Service Support Functions. The frontier between Third Party Applications and Application/Service Support Functions of NGN-GSI is called *Application-to-Network Interface* (ANI). This interface is defined by open APIs such as OSA/Parlay and Parlay X, which will be discussed on the next subsection.

B. Application/Service Development Technologies for NGN

NGN success will depend, among other factors, on the adequately opening of these networks capacities to support value-added applications development. According to Glitho [16], NGN key promise is just the ability to allow developing innovative and lucrative software applications, quickly and efficiently. Therefore, not only IT, but also telecommunication players are interested on formulate the best architecture to expose network capacities. Falcarin and Licciardi presented in [31] a survey of advanced technologies for service creation on NGN. Among the key technologies discussed there were: *Java APIs for Integrated Networks* (JAIN), OSA/Parlay, Web Services, Java and XML.

JAIN [17] is an initiative of the Java programming language community to develop service creation APIs for NGNs. These APIs bring service portability, network independence and open development for telecommunications environment. JAIN interfaces allow to hide communication protocols from value-added services. Thus, NGN services that expose such interfaces could be invoked by virtually any third party application that recognizes such interfaces, independent of available transport network.

Parlay provides IT and Telecommunication technologies integration through the definition of standardized APIs. According to [18], Parlay group, ETSI and 3GPP achieved a consensus about the set of APIs that became known as Parlay/OSA APIs. The set provides a high level view of telecommunication service capabilities to application developers, specifically determining which service can be accessed and how. Therefore, an IT developer could become prepared to work developing NGN application by studying such interface declarations.

However, it is the NGN operator that decides which interfaces will became available to third party applications through Parlay gateways. In fact, every Parlay API should be implemented in gateways [19] before it can be used by applications. Therefore, a Parlay gateway not only hides network stratus from applications, but also avoids illegal access from some malicious application. A Parlay gateway could interact with SIP proxies (e.g. S-CSCF) in order to establish SIP sessions. In this case, the Parlay gateway could interact with a JAIN-SIP-API implementation.

Typically, IT domain applications and Parlay gateways will be localized on different portions of the network. Therefore, both entities should use some technology to exchange requests and responses between their objects. An option to accomplish with this is the *Common Object Request Broker Architecture* (CORBA) technology [20].

A more contemporary possibility is to use Web Services [5]. A Web Service is a software system that allows software applications to interact directly over a network. Web Services are typically published on the Internet, in such way that its characteristics are declared and made accessible to interested clients who can send remote requests to a Web Service. The register of web services descriptions on the Internet is made by means of a private *Universal Description, Discovery and Integration* (UDDI) implementation [21]. A Web Service is described by means of an interface similar to an OSA/Parlay one, where the available methods as well as their return values (or exceptions) are precisely described. As a consequence, a Parlay gateway also can be available to remote clients as a set of Web Services. A mapping between *Unified Modeling Language* (UML) OSA/Parlay interfaces and an appropriated Web Services description language called *Web Services Description Language* (WSDL) is necessary. Web Service is published on web servers, which contain WSDL files. Because each application could use its own programming language and operational system, the interaction between an application and the Web Service is made using XML messages.

As defined in [21], WSDL specifies how to describe Web Service in XML. Therefore, with WSDL, it is possible to map one Web Service to XML. So, there is XML for two objectives: describing interfaces and exchanging messages. The XML messages could obey the *Simple Object Access Protocol* (SOAP) protocol standard [21], [22] for transportation. SOAP messages (requests and responses) could be encapsulated on *Hypertext Transfer Protocol* (HTTP) messages. However, it is important to notice that SOAP messages can be encapsulated in any protocol and not necessarily in HTTP. When encapsulated in HTTP messages, they are transported using *Transmission Control Protocol* (TCP) over *Internet Protocol* (IP), or TCP/IP.

The main obstacle of Web Services from a telecommunication operator point of view is how to integrate them to the telecommunication network. Firstly, the network could be too much exposed to applications, generating hard resource usage. Secondly, telecommunication operators must provide QoS and availability for their clients. To deal with these questions, OSA/Parlay specifications have been extended [23] to provide two technologies: Parlay Web Services and Parlay X Web Services concerned to merge telecommunication networks and Web Services. Also, these technologies could increase the amount of developers that could take access to telecommunication networks.

Parlay Web Services have WSDL files that mapped such interfaces. Parlay X specifies how to expose telecommunication capabilities using Web Services in a simplified way. While OSA/Parlay interfaces have been mapped to WSDL and Java, Parlay X API was mapped only to WSDL, because this API was created just to describe Web Services. According to [24] and [25], Parlay X API is more simple than OSA/Parlay API, virtually increasing the number of IT developers able to create value-added applications. Also, in [2] it is mentioned that Parlay X interfaces are more suited for the degree of web developers familiarization in telecommunications. Yim et. al. [24] also argued that with Parlay X it is possible to develop applications with minimal understanding of the telecommunications domain. Finally, Glitho [16] agrees that OSA/Parlay API isn't easy to be understood by readers without previous knowledge on the details of the traditional telephony networks. These reasons motivated us to focus in Parlay X, instead of OSA/Parlay.

C. Parlay X Related Works

In March 2008, Sedlar et. al. [26] proposed Parlay X and TPC, among other technologies, to build a new application in a *IP Television* (IPTV) environment. The work considered an IPTV user that is watching some advertisement on its terminal (mobile or fixed) and decided to obtain more information. Such application allows to establish a telephony call between this terminal (or other user terminal) and a call center in the enterprise which made the advertisement. Or, it allows the telespector to call to some telephone without the need to dial any number from another different terminal than its television set.

What happens is that the proposed application interacts with a Parlay X gateway and invokes a function on its TPC interface, in such way that the gateway communicates with the telecommunication operator to establish the call. This gateway is implemented in Java and the application could send requests using Web Services. According to Sedlar et. al. [26], Web Services were choosen due to its implementation and capability facilities. Our work also uses Web Services, Parlay X and Java to establish TPCs in a NGN environment. However, we are also concerned with the development methodology. Both works use `ThirdPartyCall` Parlay X API. This demonstrates the versatility of such technologies to create value-added services in converged environments.

Also, *Computer Supported Telecommunications Applications* (CSTA) standards were used in [26] to provide a common interface to access existing telephony systems infrastructure. The authors argued that existent telephony interfaces are more accessible than Parlay X APIs on today networks. Therefore, Parlay X methods were translated by a Parlay X gateway to

Abstract Syntax Notation One (ASN.1) messages, which were sent to CSTA enabled network elements.

In 2008, Nabil Ajam [27] proposed a new Parlay X Web Service concerned to users privacy. To evaluate the solution, an implementation was done in Java using a Sun Application Server. The work uses a Parlay X gateway emulator called *Location Platform Emulator* (LPE), where the new Web Service was implemented. The implementation emulates location services in a Parlay X gateway and contains a set of distributed databases to store emulated terminals information, such as terminal location. Our work also uses the same Sun Application Server to support an Ericsson Parlay X gateway emulator (see Subsection II-A). For our purposes it wasn't necessary to use a database.

II. Third Party Call: Development and Implementation

This section presents the service development methodology we used to develop a TPC NGN application using standardized Parlay X Telecommunications Web Services.

A. Development Environment Selection and Preparation

Since we have decided to use Parlay X APIs, we started defining our development architecture. We considered a Java application capable to request TPCs to a Parlay gateway, where call requests could be invoked by means of SIP sessions [1]. So, we started looking for Parlay X application gateways. However, at the time of the beginning of this work, we didn't find any mature open gateway available. Therefore, our options were: to develop a Parlay X gateway or to use some available Parlay X gateway emulator. Ericsson Telecom Web Services Network Emulator [11] appeared as a good option for our requirements, since we firstly were interest in testing a new service from the application point of view. Thus, we decided to use this gateway emulator instead of developing our own gateway. In a second step (outside the scope of this paper) a test with a real Parlay X gateway interconnected to an NGN will be necessary. Remember that service/transport separation was one of the first guidelines assumed in NGN standardization.

Ericsson Telecom Web Services Network Emulator is an open software system developed to be used in personal computers. It was built using Java *Enterprise Edition* (EE) and it is composed by an Web Services set running in a web server. The emulator belongs to the Ericsson Mobility World program. The 3.0 release was made available to developers on March 31, 2008 and it is the release used on this work.

The emulator contains a *Graphical User Interface* (GUI) that could be used via browser and allows to create emulated mobile terminals. Thus, for example, if an emulated terminal is created with the number 34713309, the application interacting with the emulator could send an *Short Message Service* (SMS) message to this address and the emulator GUI could show that such message was successfully received on the emulated terminal. This allows IT developers to test if an application is interacting successfully with a Parlay X gateway. The new virtual terminals added to the emulator will work like real ones, in a telecommunication network, but reachable through this emulated gateway. Also, if a terminal was turned off and a call is in progress, the emulator will return an exception. Two web servers could be used with the emulator: Apache Tomcat 6 or *Sun Java System Application Server* (SJSAS). In both cases, administration console could be accessed via web browser, showing which applications and Web Services are active or not. SJSAS is a fully compliant implementation of Java EE 5 platform [28] and was the option taken in this work. SJSAS is now called Sun GlassFish Enterprise Server.

[1]The mapping of Parlay X implementation to SIP protocol is outside the scope of this paper.

Figure 1 illustrates Ericsson service development architecture [11]. An IT professional could work developing server and client web software applications. The server web application will receive HTTP requests from the client web application and generate SOAP over HTTP requests sent over a TCP/IP network. The server web application could be developed in any platform [11]. The *Java API for XML Web Services* (JAS-WS) servlet receives SOAP messages, removes XML code and sends it to a `ThirdPartyCallWsEndPoint` class, as it will be described on next section. The emulator changes terminal appearance according to TPC interface implementation. The IT developer interacts with the emulated terminals to emulate user behavior. Observe, that the emulator and the web applications could be installed on different computers or on the same computer. In the latter case, HTTP requests will not pass by a real TCP/IP network. They will be handled by computer's operational system.

Besides the functions to emulate terminals and its status, the emulator allows: 1) to set up the kind of exception must be returned to the application and when they must be returned; 2) to return information about requested services; 3) to set up geographical positioning information to emulated terminals over a map. Release 3.0 supports four use cases: SMS exchange, *Multimedia Messaging Service* (MMS) exchange, terminal status information and terminal location. As we will present on Section II-B, a new capacity was implemented to allow TPCs emulation.

We installed software development tools based on Ericsson document [11]. The only exception was the Eclipse tool, since [11] suggests SUN NetBeans for Java coding.

B. Learning about TPC Parlay X Interface

Third Party Call is a kind of telephony call that allows one entity to invoke a voice communication between two or more participants. The great advantage of TPC support on web based applications is the capability to establish voice calls invoked by web browsers or other web applications. The possibilities of third party call applications together with web based applications are quite interesting. As an example to sustain our affirmation, consider a person who is navigating in an online auction and shopping web site and desires more information about some product. Alternatively, instead of a chat terminal, the web site could offer a button to talk with some attendant. This application is being called click-to-talk.

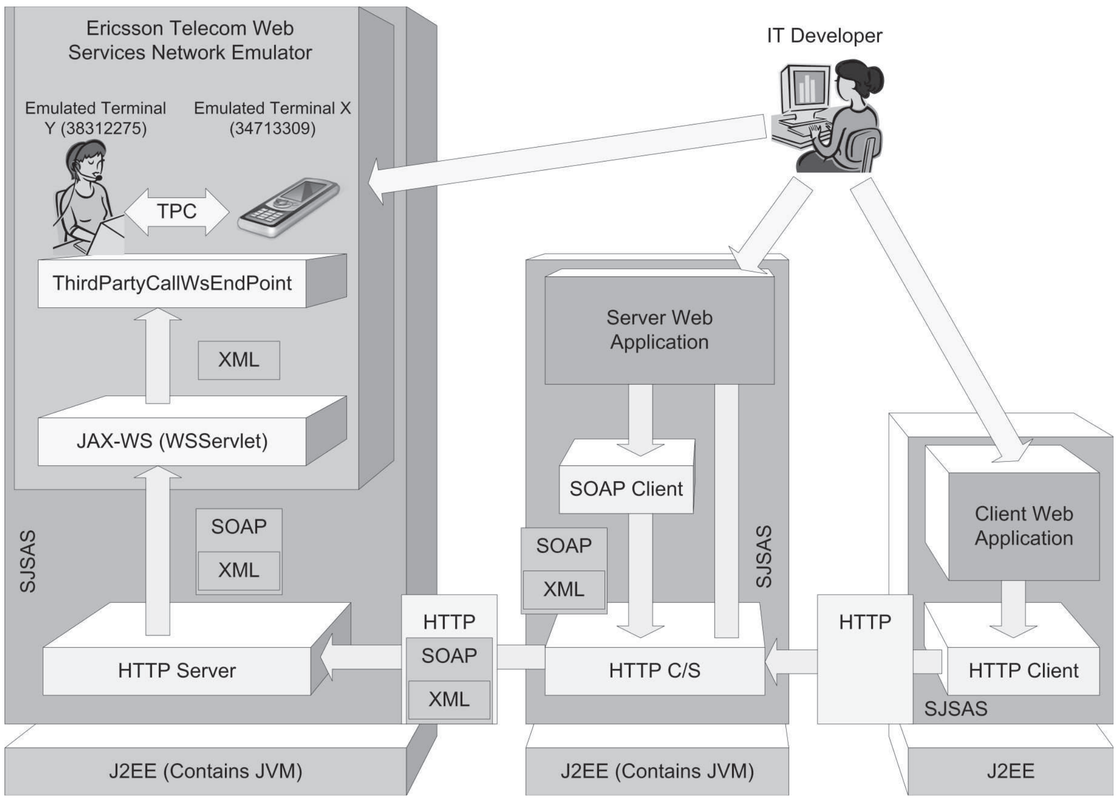

Fig. 1. Service development architecture with gateway emulator.

In this case, some web application could originate a TPC to put in contact the customer and a company attendant, by means of their respective telephone sets.

Obviously, this is just a simple example, but pertinent to arise the understanding of how useful it could be to support third party call requests in a telecommunications operator. We decided to implement this new service on Ericsson emulator since we are mainly focused on developing a methodology for service development on NGNs. TPC interface is quite simple when compared with other Parlay X interfaces, but not less important. Therefore, it was a good starting point for us, since it makes possible to work on Parlay X interface implementation limited to a reasonable degree of complexity.

In this work, we followed the complete TPC specification document referenced in [29]. We also used Ericsson document [11]. The methods defined on [29] for `ThirdParty Call` interface are:

- `makeCall`: Allows an application to request the establishment of a voice call between two terminals (or parties).
- `cancelCall`: Allows an application to cancel a previous third party call request. It is ignored if both terminal are already connected.
- `getCallInformation`: Allows an application to obtain status information regarding a previous third party call request. See II-E for more details about call status.
- `endCall`: Allows to finish an initiated or connected call.

C. Compiling WSDL Files

The WSDL files related to `ThirdPartyCall` service have been obtained on Parlay web site as well as the respective API. There are two files:

- `parlayx_third_party_call_interface_2_3.wsdl`
- `parlayx_third_party_call_service_2_3.wsdl`

The second one defines some abstract data types used by `ThirdPartyCall` service, such like: `CallInformation`, `CallStatus` and `CallTerminationCause`.

To compile WSDL files and *XML Schema* (XSD) in order to automatically generate the desired Java files, we used Java Platform together with JAX-WS API. The tool wsimport was used to map WSDL and XSD files into Java code. The following XML script was necessary:

```
<property name="px_v2_1.tpc.wsdl"
value="parlayx_third_party_call_service_2_3.wsdl" /> <target
name="wsimport_tpc" description="Compiles the Third Part
Call Stubs">
     <mkdir dir="${src.dir}"/>
     <mkdir dir="${build.dir}"/>
     <exec executable="${env.JAVA_HOME}/bin/wsimport">
         <arg value="-s"/>
         <arg value="${src.dir}"/>
         <arg value="-d"/>
         <arg value="${build.dir}"/>
         <arg value="${wsdl.px_v2_1.dir}/${px_v2_1.tpc.wsdl}
         "/>
     </exec>
     <echo message="Done"/>
 </target>
```

This XML code was attributed to the build.xml file of the emulator. Observe that this file makes reference to the second file mentioned earlier. This second file also makes a reference to the first one mentioned. Therefore, the three files are considered by the wsimport tool.

After mapping these files, several Java entities (classes, interfaces and data structures) were automatically created and distributed into three Java packages. The `Third PartyCall.java` file contains the Java interface code for `ThirdPartyCall` service methods. The package `org.csapi.schema.parlayx.third_party_call .v2_3` contains classes that define objects with attributes values for `ThirdPartyCall` methods as well as the values returned by these methods. For example, a `MakeCall` class instance could be used to maintain parameters values to be passed to `makeCall` method of the interface defined on `ThirdPartyCall.java` file.

The files `CallInformation.java`, `CallStatus .java` and `CallTerminationCause.java` from `org.csapi.schema.parlayx.third_party_call.v2_3` package define data structures to maintain information about call status, call ending cause, etc. These structures, for example, are the Java mapping of the XML content from file `parlayx_third_party _call_types_2_3.xsd`. More precisely, as an example, the contents of the file `Call Status.java` is the Java mapping of the following code from `parlayx_third_party_call _types_2_3 .xsd`:

```
<xsd:simpleType name="CallStatus">
   <xsd:restriction base="xsd:string"
      <xsd:enumeration value="CallInitial"/>
      <xsd:enumeration value="CallConnected"/>
      <xsd:enumeration value="CallTerminated"/>
   </xsd:restriction>
</xsd:simpleType>
```

D. Parlay X Interfaces Implementation

The UML diagram of Figure 2 shows some classes and interfaces that were created during the implementation of the `ThirdPartyCall` interface. The `ThirdPartyCallWsEnd Point` class is the final responsible to define the algorithms of this interface methods and it is defined on `Third PartyCall.java` file. Therefore, in the emulator, this class determines how the gateway will work (see Figure 1), according to [29]. Evidently, this class doesn't implement all the code necessary for the new service, it relies on several other classes with useful functions for the `ThirdPartyCall` service. An object (or instance) of `ThirdPartyCallWsEndPoint`, as necessary, uses an object of `ThirdPartyCall ListenerImpl` class.

When the SJSAS web server runs, automatically the emulator is instantiated, becomes available at a certain IP address and starts to accept Web Services requests. The first emulator class that runs is the `InitServlet` class during emulator initialization. Such class is already present in the emulator version 3.0. This class is responsible to initiate other classes, which prepare the execution of the `ThirdPartyCallWsEndPoint` class.

When the emulator receives a request, as for example, "`makeCall`", the `make Call` method from `ThirdPartyCallWsEndPoint` runs. However, this method needs an object that implements the interface `ThirdPartyCallEndPointListener` as shown on Figure 2. In fact, every time `ThirdPartyCallWsEndPoint` object has one of its methods invoked, a new instance of the object `ThirdParty CallListenerImpl` is created. Therefore, a new instance of this object will always be responsible to answer the needs of an `ThirdPartyCall WsEndPoint` object. The consequence is that a certain service request will not interfere on other requests. Though, these objects share some common structures, like a Java vector to maintain information about answered requests. Figure 2 also shows that `ThirdPartyCallEndpointListener` object is always an instance of the `ThirdPartyCallListenerImpl` class.

An object of `WebServiceContext` class is used by `ThirdPartyCallWsEndPoint` object to provide access to the object `HttpServletRequest`. This object is created by the web server and contains address information about the Web Service requester.

E. Including New Features on Emulated Terminals

In order to better design functionalities and states necessary to implement `ThirdParty Call` service on emulated terminals, we drew the state diagram shown on Figure 3. This diagram doesn't exist on Parlay X document[29]. We drew it based on the defined states available on [29]. The states represent all possible behaviors of the emulated terminals, according to `ThirdPartyCall` Web Service requests and use cases of an IT developer interacting with the emulator. In other words, it represents the defined states when a requested call between terminals X and Y, caller and called, is in progress. Information about each of these states can be obtained through the execution of the `getCallInformation()` method from `ThirdPartyCall` Web Service.

With exception of the states `Call_Initial` and `Call_Connected` all the other states change to the final state after a certain time. This means that call status information must be maintained during a certain time and after that be removed as specified on [29]. We adjusted this time to 40 seconds. In addition, `Call_Initial` state automatically goes to `Call_Terminated` state if X or Y doesn't answer the call after 20 seconds. Those parameters are implementation-dependant and [29] haven't to fix such parameters. In this paper, those parameters concern the emulator.

Although this diagram is a direct consequence of the rules presented in [29], some details are not specified on such document. For example, when one of the terminals already connected decides to turn off (or the user hungs up), the other terminal doesn't return to a free state. It remains busy until its button Hung Up receives a click on the terminal GUI. We considered that the terminals are fixed. In this case, the second terminal will be really free only after its user hung up. Therefore, with this decision, telephone status could become dependent of the user status. This consideration is important, since a Parlay X gateway could in fact be used on fixed

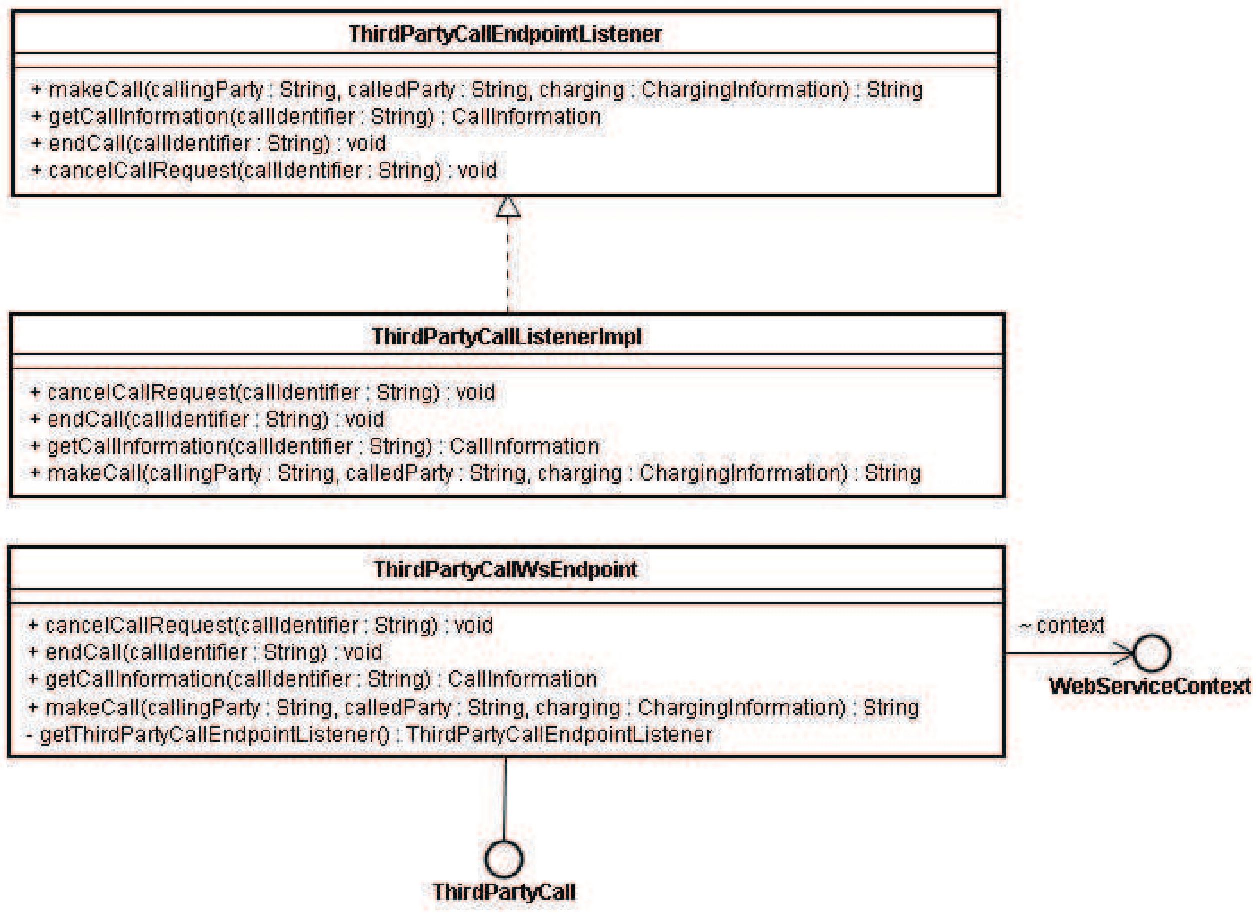
ThirdPartyCallEndpointListener
+ makeCall(callingParty : String, calledParty : String, charging : ChargingInformation) : String
+ getCallInformation(callIdentifier : String) : CallInformation
+ endCall(callIdentifier : String) : void
+ cancelCallRequest(callIdentifier : String) : void
ThirdPartyCallListenerImpl
+ cancelCallRequest(callIdentifier : String) : void
+ endCall(callIdentifier : String) : void
+ getCallInformation(callIdentifier : String) : CallInformation
+ makeCall(callingParty : String, calledParty : String, charging : ChargingInformation) : String
ThirdPartyCallWsEndpoint
+ cancelCallRequest(callIdentifier : String) : void
+ endCall(callIdentifier : String) : void
+ getCallInformation(callIdentifier : String) : CallInformation
+ makeCall(callingParty : String, calledParty : String, charging : ChargingInformation) : String
- getThirdPartyCallEndpointListener() : ThirdPartyCallEndpointListener
~ context
WebServiceContext
ThirdPartyCall

Fig. 2. UML diagram of some classes for TPC service.

terminal networks. Observe that this details are independent from Parlay X and therefore the document [29] doesn't define terminal states.

Another consideration related to the final state is that if a request is done to `getCalInformation()` method, so the application that requested the information will receive the following answer: "`Response Error. Information retention time out`". This occurs because the emulator maintains status information during a predefined time. After that, if an application tries to obtain call status information, such application will receive a warning. In this case, the timeout occurred 40 seconds after the finishing state was achieved.

With the `ThirdPartyCall` interface implemented, the emulated terminals became able to emulate the reception of a telephony call, even ringing the phone bell, being also possible to accept or to deny new calls.

Besides `makeCall()` method of `ThirdPartyCall` interface, it has been added to the emulator the ability to receive requests for other methods of this interface as specified on [29]. For instance, after an accepted call request the application can ask for call information using the `getCallInformation()`.

To include the new terminal behavior on the emulator it was necessary to edit and/or create the files: `terminalscripts.js`, `playNoSound.jsp`, `playoldringsound .jsp`, `playringsound.js` and `terminalstyle.css`. Also, we handled some files of `parlay ws.emulator.web.ajax` package.

The file `terminalscripts.js` contains a JavaScript code which determines emulated terminals behavior. For example, in this file, there is the code that controls the kind of terminal information (SMS, MMS, call, etc) that is shown on graphical interface depending on user interaction. There are also controls to determine button types to be shown (Make a Call, Hung Up or Accept Call). One of the modifications done on this file was the inclusion of the following code:

```
function acceptCall(){
   makeCall();
   var terminal = document.getElementById('terminalUri').
   value;
   checkXmlHttpCALL = getXmlHttp();
   checkXmlHttpCALL.onreadystatechange = doNothing;
   checkXmlHttpCALL.open("GET",
                     "AjaxServlet?action=
                     DoAcceptCurrentCall&terminal="
                     + terminal, true);
   checkXmlHttpCALL.send(null);
   playNoSound();
}
```

The code is run when the user does a click using `Accept Call` button. The *Asynchronous JavaScript and XML* (AJAX) [30] technology was used to allow the asyn-

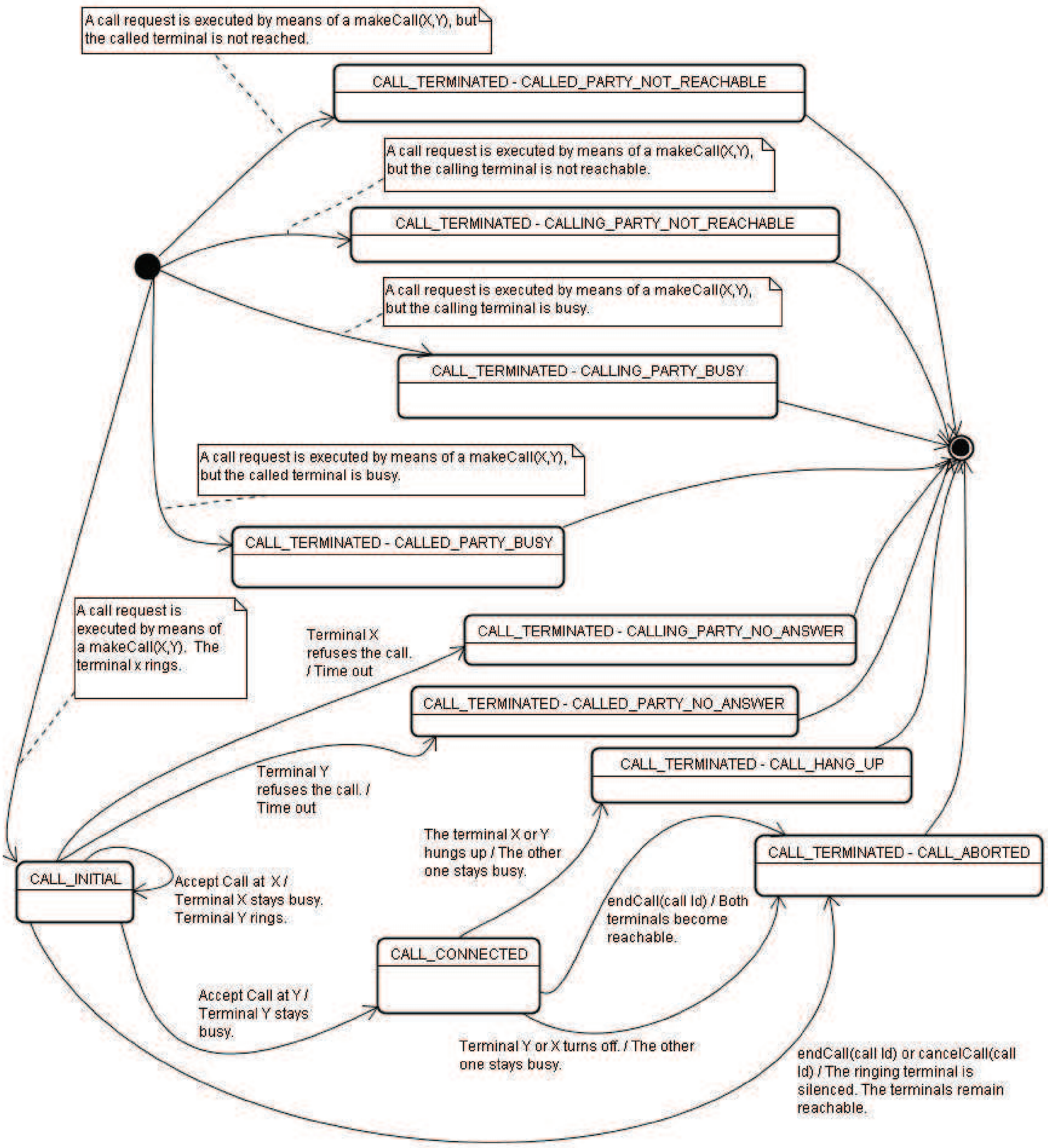
A call request is executed by means of a makeCall(X,Y), but the called terminal is not reached.
CALL_TERMINATED - CALLED_PARTY_NOT_REACHABLE
A call request is executed by means of a makeCall(X,Y), but the calling terminal is not reachable.
CALL_TERMINATED - CALLING_PARTY_NOT_REACHABLE
A call request is executed by means of a makeCall(X,Y), but the calling terminal is busy.
CALL_TERMINATED - CALLING_PARTY_BUSY
A call request is executed by means of a makeCall(X,Y), but the called terminal is busy.
CALL_TERMINATED - CALLED_PARTY_BUSY
A call request is executed by means of a makeCall(X,Y). The terminal x rings.
Terminal X refuses the call. / Time out
CALL_TERMINATED - CALLING_PARTY_NO_ANSWER
CALL_TERMINATED - CALLED_PARTY_NO_ANSWER
Terminal Y refuses the call. / Time out
CALL_TERMINATED - CALL_HANG_UP
The terminal X or Y hungs up / The other one stays busy.
CALL_TERMINATED - CALL_ABORTED
CALL_INITIAL
Accept Call at X / Terminal X stays busy. Terminal Y rings.
endCall(call Id) / Both terminals become reachable.
CALL_CONNECTED
Accept Call at Y / Terminal Y stays busy.
Terminal Y or X turns off. / The other one stays busy.
endCall(call Id) or cancelCall(call Id) / The ringing terminal is silenced. The terminals remain reachable.

Fig. 3. State diagram of a call request between emulated terminals.

chronous use of the `DoAcceptCurrentCall` class. Therefore, the new methods presented on `terminalscripts.js` use `parlayxws.emulator.web.aj ax` package. Figure 4 illustrates Java, AJAX, JSP and Javascript usage on terminal emulation. The class `DoAcceptCurrentCall` for example has the responsibility to set up some terminal information, such as to configure busy status to the terminal, to turn off ring status and maintain Call Connect information together with the call information, if it is the case.

The `parlayxws.emulator.web.ajax` packet classes are action classes designated to take control of terminal behavior, which objects receive requests directly by JavaScript and can interact with other emulator objects. These classes serve AJAX request sent by JavaScript, and even their algorithm are complex, this doesn't affect emulated terminals GUI. The other files of the `parlayxws.emulator.web.ajax` package take care of other emulated terminals behaviors.

Some methods of the `terminalscripts.js` script use the files `playNoSound .jsp`, `playoldringsound.jsp` and `playringsound.jsp`. These files written in JSP are responsible to make the sounds when a terminal is ringing. These files were created during our work.

F. Editing XML Files

After finishing the implementation of the source code needed to reflect the new emulator functionalities regarding TPC, it was necessary to edit the `build.xml`, `sun-jaxws.xml` and `web.xml` files to reflect such changes.

The `sun-jaxws.xml` and `web.xml` files contain the necessary descriptions to run a web application. For example, these files are used to expose each emulator Web Service. To expose means to declare which are the classes that implement service interfaces, in such way that, whether the web server receives a request directed to a service defined on interface Z, a method in the object of the class that implements Z will be invoked. This invocation will be provided by some other object running in the web server, possibly defined by an API such as JAX-WS.

In the file `sun-jaxws.xml` it was added the following code:

```
<endpoint implementation="parlayxws.emulator.
ws.px_spec_v2_1.tpc.ThirdPartyCallWsEndpoint"
name="ThirdPartyCall" url-pattern="/ParlayXTpcAccess/
services/ThirdPartyCall" />
```

This XML code exposes the `ThirdPartyCallWsEndpoint` class, which represents the `ThirdPartyCall` Web Service. The object of such class is the end point for an application request. After that, the request is finally treated by the emulator as shown on Figure 1. This explains the EndPoint string in the class name. We concerned with the coding of this class, but not with other classes responsible to interpret SOAP messages, since this task is a JAX-WS responsibility. This is the case of the `WSServlet` class. However, it must exist a mapping among this class and all the Web Services available on the emulator.

In other words, a mapping must indicate that the `WSServlet` object will use an object to represent the end point where a received request must be delivered. This object could be a `ThirdPartyCallWsEndpoint` in the case of the TPC service or any other EndPoint object for other services. For example, if the emulator receives a request about the status of some terminal, this request will be delivered to the `TerminalStatusWsEndpoint` class already coded by Ericsson.

The following sequence of XML code shows the contents from the `web.xml` file illustrating previous discussion.

1) To indicate which is the JAX-WS class that constitutes a `WSServlet` it was necessary the code:

```
<servlet>
    <display-name>WSServlet</display-name>
    <servlet-name>WSServlet</servlet-name>
    <servlet-class>
        com.sun.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
</servlet>
```

2) If the web server received a HTTP request which *Uniform Resource Locator* (URL) contains the `ThirdPartyCallService` name, then the `WSServlet` class will process such request. This was indicated using the following code:

```
<servlet>
    <servlet-name>ThirdPartyCallService</servlet-name>
    <servlet-class>
        com.sun.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

3) To indicate that a `WSServlet` class object is mapped to a `ThirdPartyCallWsEndpoint` class object the following code was necessary:

```
<servlet-mapping>
    <servlet-name>WSServlet</servlet-name>
    <url-pattern>/ParlayXTpcAccess/services/*
    </url-pattern>
</servlet-mapping>
```

4) The following code was needed to indicate that a URL containing "`/ThirdPartyCall Service`"was mapped to a servlet with name `ThirdPartyCall Service`, therefore validating the mapping defined into (2).

```
<servlet-mapping>
    <servlet-name>ThirdPartyCallService</servlet-name>
    <url-pattern>/ThirdPartyCallService</url-pattern>
</servlet-mapping>
```

In summary, when a request, whose URL contains "`/ThirdPartyCallService`", arrives at the web server, by the definition done on (4) the server must use the servlet named `ThirdPartyCallService`. By definition (2), this implies on the usage of the `com.sun.xml.ws.transport.http.servlet.WSServlet` class. This class is referred by the `WSServlet` according to (1). Also, by definition (3), this last servlet is mapped to any service defined on web server. Finally, by the definition done on `sun-jaxws.xml` file, the right class to run the demanded service is found.

Regarding `build.xml` file, just one more XML instruction was needed to allow that the files created on this work could be compiled together with the automatically Java generated files, JSP and .js files. The following line code shows this modification:

```
<property name="thdPtCall.package.name" value=
"parlayx.tpc"/>
```

G. Testing with Gateway Emulation

For testing purposes, we used a desktop computer to install the gateway emulator. Using SJSAS interface, an IT developer could see which are the Parlay X interfaces running on the web server. Also, SJSAS web server builds a web page with a form where it is possible to directly send service requests to

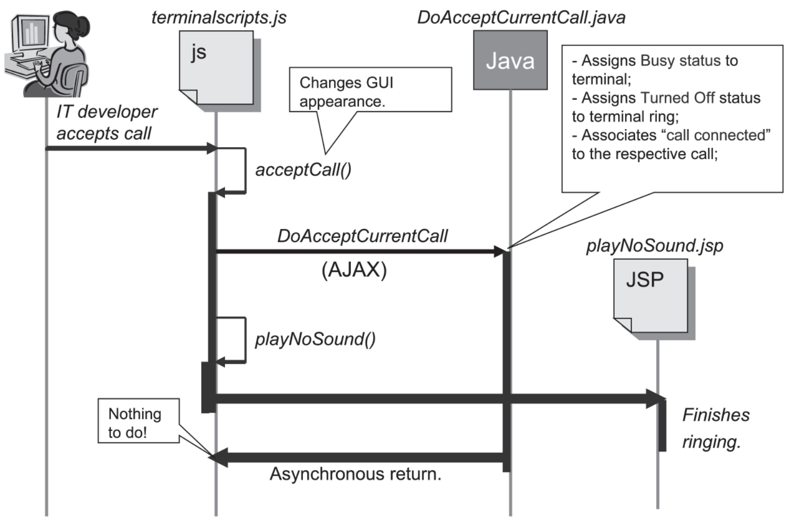

Fig. 4. Java, Javascript, JSP and AJAX technologies usage on terminal emulation.

the gateway emulator. Therefore, we used this web page to test TPC interface, instead of developing the server and client web application shown on Figure 1. The main reason why is that the gateway emulator is unable to determine if a TPC request is sent by an external server web application or by a SJSAS web form. The emulator reacts exactly on the same way.

Also, SJSAS sends SOAP messages as a server web application would do. SJSAS uses WSDL files to build the aforementioned test web pages. Tough, it was possible to use different computers in a more realistic scenario, we chose this approach because we were interested on start the tests as quickly as possible, as well as maintaining the development environment portable and realistic enough to validate our TPC implementation and methodology.

Figure 5 illustrates our test procedure during a third party call between two emulated terminals (X and Y). Also, it reproduces obtained emulated terminals GUIs. The sequence was:

1) An IT developer using the emulator GUI (web pages) creates two emulated terminals.
2) Also using the emulator GUI, he/she requests the establishment of a call between emulated terminals. This request is handled by SJSAS and delivered to gateway emulator. If a client web application was used, the TPC request "click"could be sent by HTTP to the server web application, which would generate an appropriated SOAP message for the SJSAS server to reach gateway emulator.
3) The emulator receives the request and using its objects, it allocates a *Request ID* and stores call request information, such as terminal addresses. The emulator sends a response containing the allocated *Request ID*.
4) Emulated terminals query periodically data on emulator and discover about a call request. Terminal X will ring, while terminal Y will remain waiting. Remember that in a TPC both telephones will ring.
5) When a terminal rings, its GUI looks a little bit different, now with a new yellow button called *Accept Call*. Figure 5 shows the emulated terminal X GUI at this moment. Such yellow button was a new feature added to the emulator.
6) The IT developer effectuates a click, which emulates that the call was accepted by the final user interested in more information about a certain product. Again, the terminal X GUI reflects the change, showing that a call is in progress.
7) In the sequence, the emulator registers the information that it is inquiring terminal Y to ring. As happened to terminal X, terminal Y discovers this fact and rings. See Figure 5.
8) The user on terminal Y decides to accept or decline the call. If the call is accepted, both terminals change their graphical interfaces to reflect that a call is up.

When `Accept Call` button receives a click, the terminal stops ringing immediately and its status is changed to busy. Here is an example of a SOAP message sent by SJSAS web page form to the emulator, where the caller has the address 34713309 and the called 38312275:

```
<?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.csapi.org/schema/parlayx/
third_party_call/v2_3/local"
xmlns:ns2="http://www.csapi.org/schema/parlayx/common/v2_1">
    <soapenv:Body>
        <ns1:makeCall>
            <ns1:callingParty>tel:34713309</ns1:callingParty>
            <ns1:calledParty>tel:38312275</ns1:calledParty>
        </ns1:makeCall>
    </soapenv:Body>
</soapenv:Envelope>
```

After an accepted call request the developer could ask for call information using the `getCallInformation()` method as follows:

1) First, the IT developer using SJSAS could send a SOAP

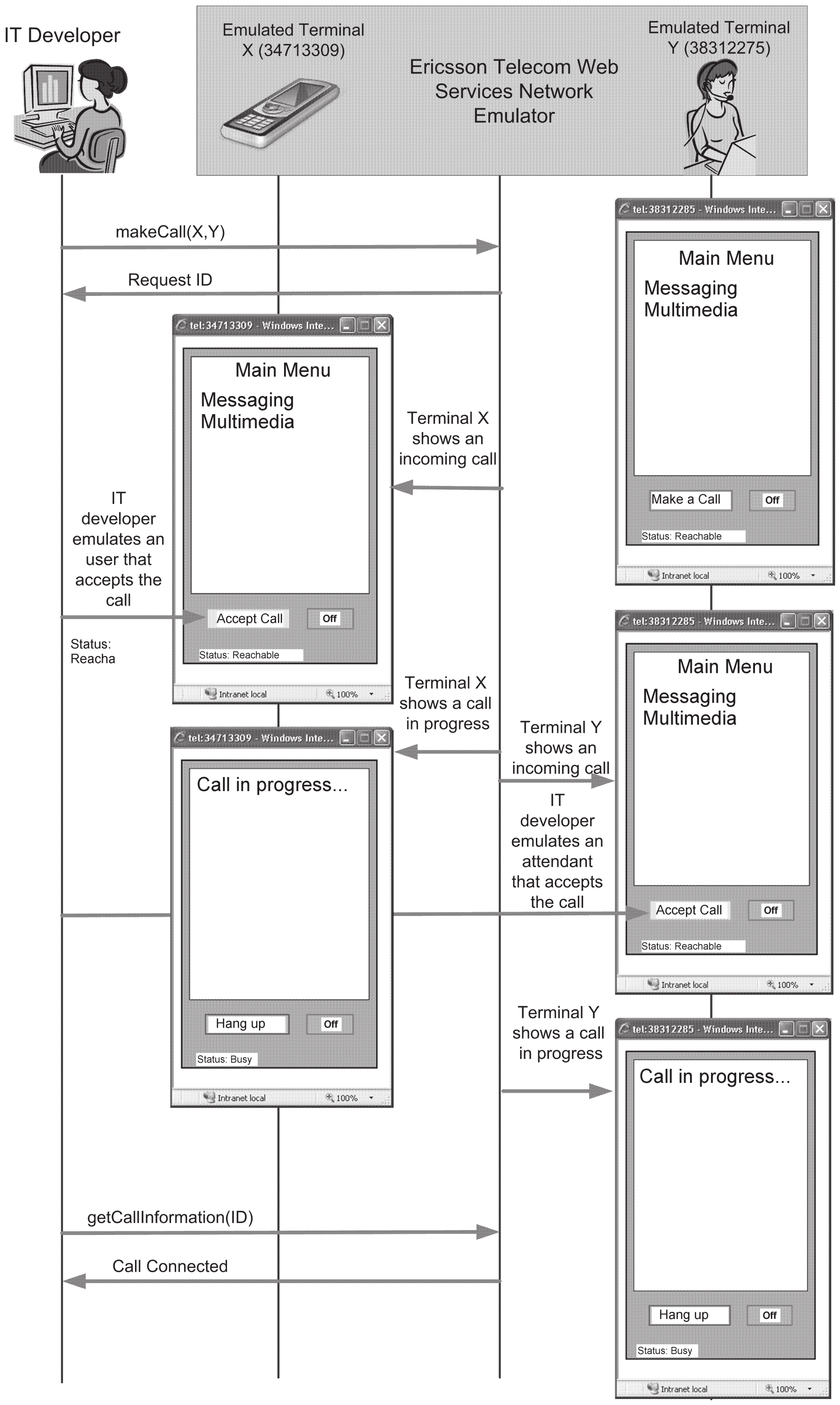
IT Developer
Emulated Terminal X (34713309)
Ericsson Telecom Web Services Network Emulator
Emulated Terminal Y (38312275)
makeCall(X,Y)
Request ID
Terminal X shows an incoming call
IT developer emulates an user that accepts the call
Status: Reacha
Terminal X shows a call in progress
Terminal Y shows an incoming call
IT developer emulates an attendant that accepts the call
Terminal Y shows a call in progress
getCallInformation(ID)
Call Connected
Main Menu
Messaging
Multimedia
Make a Call
Accept Call
Call in progress...
Hang up
Off
Status: Reachable
Status: Busy

Fig. 5. Testing the service emulation architecture.

message requesting call information:

```
<?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.csapi.org/schema/parlayx/third_
party_call/v2_3/local"
xmlns:ns2="http://www.csapi.org/schema/parlayx/common/
v2_1">
    <soapenv:Body>
        <ns1:getCallInformation>
            <ns1:callIdentifier>0</ns1:callIdentifier>
        </ns1:getCallInformation>
    </soapenv:Body>
</soapenv:Envelope>
```

In this example the call identifier 0 is passed as a `getCallInformation ()` method parameter to identify the desired call. The IT developer knows this identifier since it was received as a return of `makeCall()` method.

2) This SOAP message is interpreted by JAX-WS (or other equivalent) tool available on the web server. Thus, the `getCallInformation()` method of the `ThirdPartyCall WsEndPoint` object is invoked.
3) The emulator searches the desired information related to the `makeCall(X, Y)` and returns the current status inside of a `CallInformation` object instance. This object contains a `CallStatus` object instance with the status information in the format of a string.
4) The returned `CallInformation` object is interpreted by other Java objects at web server, available on JAX-WS and capable to formulate a SOAP answer. Thus the emulator answers with the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?> <soapenv:
Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.csapi.org/schema/parlayx/
third_party_call/v2_3/local"
xmlns:ns2="http://www.csapi.org/schema/parlayx/
common/v2_1">
    <soapenv:Body>
        <ns1:getCallInformationResponse>
            <ns1:result>
                <callStatus>CallConnected</callStatus>
            </ns1:result>
        </ns1:getCallInformationResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

5) The requester interprets SOAP message and obtains the current call status, which is `CallConnected` in this case.

III. Conclusion

This paper explored how open APIs could be used to expose telecommunications network resources for IT developers in order to accelerate value-added services development. Specifically, it presented the role of OSA/Parlay or Parlay X gateways to invoke telecommunications services by means of SIP proxies, such as S-CSCF in NGN-GSI or IMS networks. Possible key technologies for this task were presented considering simplicity and IT developers familiarization. It looks to be a consensus that Parlay X API is more simple than OSA/Parlay API, virtually increasing the number of IT developers able to create value-added applications. Because Parlay X uses Web Services to implement service interfaces, it was necessary to deal with WSDL, SOAP, XML and other web services related technologies and tools. The work contextualized where each of these technologies were employed and in which order. Also, it contributed presenting the right moment to use each one as well as on how to fit the resulting deliverables of each technology in order to compose a final service software. Obtained task sequence was organized in a methodology that could be used in other similar works.

The methodology used starts with development environment selection and preparation. It was based on a gateway emulation architecture, keeping development environment simple, portable and isolated from telecommunications world. We contend that this strategy is interesting in a first moment, since it maintains telecommunications complexity hidden from IT developers. The second step in methodology is to learn about desired Parlay X interface and methods. As a use case, we focused on TPC interface. The compilation of needed Java entities from WSDL files was the third step. The fourth step was to implement desired Parlay X interface methods in the emulator using Java and the fifth step was to include the new features in emulated terminals. In the sequence, it was necessary to edit XML files to reflect the changes on emulated terminals. Finally, we successfully tested TPC service.

We concluded that Parlay X Web Services promoted a simplified, organized and exciting way to use exposed telecommunications capabilities, very different from a direct SIP approach, and that isn't necessary to have a real gateway to test applications. Gateway emulation proved to be an interesting approach, in such way that, from a web service client application point of view, the needed work to access some NGN resource is purely computational. The work is still more reduced if a desired Parlay X interface is already implemented on gateway emulator. In this case, the IT developer must concern only about its own application.

We intend to expand this work to other Parlay X interfaces (e.g. Payment and Presence), contributing to increase used gateway emulation capabilities. The proposed methodology proved to be general enough for this. Another future work is to compare standalone Parlay X performance with standalone or combined OSA/Parlay, SIP or CSTA technologies. The objective is to determine the effect of functional overlapping in service performance. If a certain value-added service needs more than one of these technologies, what is the impact on performance?

Acknowledgments

We would like to thank Erik Eriksson and Ericsson, by the help regarding Parlay X Gateway Emulator. Also, we would like to thanks Inatel by the support.

Referências

[1] The Parlay Group. http://www.parlay.org.

[2] The Parlay Group, Parlay Web Services Working Group, Parlay Web Services: Application Deployment Infrastructure, Oct. 2002.

[3] SUN, Java Programming Language. http://java.sun.com.

[4] WWW Consortion, XML. http://www.w3.org/XML/.

[5] WWW Consortion, Web Services Architecture. http://www.w3.org/TR/ws-arch.

[6] ITU-T, General Overview of NGN, Recommendation Y.2001, 2004.

[7] Carugi, M., Hirschman, B., & Narita, A., Introduction to the ITU-T NGN Focus Group Release 1: Target Environment, Services, and Capabilities, IEEE Commun. Mag., vol. 43, no. 10, Oct. 2005.

[8] ITU-T, Next Generation Networks Global Standards Initiative. http://www.itu.int/ITU-T/ngn/.

[9] 3GPP, Technical Specification Group Services and System Aspects, TS 23.228: IP Multimedia Subsystem (IMS).

[10] Carvalho, R. P., Alberti, A. M.. Java Technologies for NGN Service Creation: Discussion and an Architecture to Improve SIP Addresses Discovery, The IASTED European Conference on Internet and Multimedia Systems and Applications, Chamonix, 2007.

[11] Ericsson, Telecom Web Services Network Emulator: Developer's Guide, Ericsson, Mar. 2008.
[12] 3GPP, IP-Multimedia Subsystem. http://www.3gpp.org/article/ims.
[13] ETSI, Next Generation Network. http://www.etsi.org/website/Technologies/NextGenerationNetworks.aspx.
[14] ITU-T, Functional requirements and architecture of the NGN release 1, Recommendation Y.2012, 2006.
[15] IETF, SIP: Session Initiation Protocol, RFC 2543. http://tools.ietf.org/html/rfc2543.
[16] Glitho, R. H., Developing Applications for Internet Telephony: A Case Study on the Use of Parlay Call Control APIs in SIP Networks, IEEE Network, Montreal, Canada, vol. 18, June 2004, 48-55. ISSN: 0890-8044.
[17] SUN Microsystems, JAIN and Java in Communications, White Paper, Santa Clara, California, March 2004.
[18] Gupta, M., Parlay/OSA Mature for the Telecoms Market, Eurescom Workshop 'OSA and Parlay @ Work', Heidelberg, Nov. 2002.
[19] Venters, T., Open Standard Initiatives For Service Delivery Platforms, TMCnet, Mar. 2004.
[20] Leclerc, M., Network Resource Gateway: Benefits and Business Opportunities of Building Wireless Applications Using Parlay/OSA for Developers, Ericsson, Montreal, Canada, Dec. 2003, Part 3: Developing Parlay/OSA Applications.
[21] Cerami, E., Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI and WSDL, O'Reilly, Fev. 2002. ISBN: 0-596-00224-6.
[22] WWW Consortion, SOAP. http://www.w3.org/TR/soap.
[23] The Parlay Group, Parlay and Next-Generation Networks, White Paper, May 2005.
[24] Yim, J., Choi, Y., & Lee, B., Third Party Call Control in IMS using Parlay Web Service Gateway, IEEE Conference in Advanced Communication Technology, Fev. 2006, 221-224, ISBN 89-5519-129-4.
[25] Wegscheider, F., Bessler, S., & Gruber, G., Interworking of Presence Protocols and Service Interfaces, IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, p. 45-52, Aug. 2005.
[26] Sedlar, U., Zebec, L., Bester, J., & Kos, A., Bringing Click-to-Dial Functionality to IPTV Users, IEEE Communications Magazine, Toronto, Canada, vol. 46, p. 118-125, Mar. 2008. ISSN: 0163-6804.
[27] Ajam, N., Privacy Based Access to Parlay X Location Services, Fourth International Conference on Networking and Services, Mar. 2008. Gosier, p. 204-210, ISBN: 978-0-7695-3094-9.
[28] Jendrock E., Ball J., Carson D., Evans, I., Fordin S., Haase, K., "Java EE 5 Tutorial", Prentice Hall, 3rd edition, Nov. 2006, ISBN 03-2149-029-0.
[29] ETSI, The Parlay Group, Open Service Access (OSA): Parlay X Web Services, ETSI standard: ETSI ES 202 391-2, v. 1.2.1, Part 2: Third Party Call, France, Dec. 2006.
[30] Paulson, L. D., Building Rich Web Applications with Ajax, IEEE Computer Magazine, Oct. 2005. Volume: 38, Issue: 10. 14-17. ISSN: 0018-9162.
[31] Falcarin, P., & Licciardi, C. A., Technologies and Guidelines for Service Creation in NGN, exp magazine, vol. 3, n. 4, p. 46-53, Dec. 2003.

Rodrigo Pimenta Carvalho received the degree in Computer Science from Minas Gerais Federal University (UFMG), Belo Horizonte, MG, Brazil, in 1997, and the M.Sc. degree in Electrical Engineering from Instituto Nacional de Telecomunicações (INATEL), Santa Rita do Sapucaí, MG, Brazil, in 2008. In July 1997 he joined the INATEL, as a Software Developer. He has experienced software development across 13 years, including projects for companies like IBM, NEC, LG, Nokia, Ericsson and Benchmark. He is a member of the INATEL Competence Center, daily working with technologies like programming languages, Object Oriented Programming and Telecommunication protocols. In 2002 he received the Best Business Plan Award from Inatel Entrepreneurship Core (NEMP). In 2004 he became a Sun Certified Programmer for the Java 2 Platform 1.4. Nowadays his main working area is Telecommunication software design and development, where he has expertise in project, modeling, codifying, and tests. His current interests include application development in Next Generation Networks, adopting industry standards tools such as Java, stack NIST-SIP, Web Services and Parlay X. He is also a co-founder of the enterprise Biosoftware Sistemas Didáticos Ltd.

Antonio Marcos Alberti received the degree in Electrical Engineering from Santa Maria Federal University (UFSM), Santa Maria, RS, Brazil, in 1986, and the M.Sc. and Ph.D. degrees in Electrical Engineering from Campinas State University (Unicamp), Campinas, SP, Brazil, in 1998 and 2003, respectively. In February 2004 he joined the Instituto Nacional de Telecomunicações (INATEL), as an Adjunct Professor. He has experience in teaching more than 8 post-graduation disciplines, including Analysis and Performance Evaluation of Communication Networks, Optimization Methods Applied to Telecommunications and Convergent Networks. He is a member of the editorial board of the INATEL telecommunications magazine. He was member of the technical committee of Globecom, TEMU, ICDT and ANSS conferences. In 2010, wrote a book chapter entitled "Future Network Architectures: Technological Challenges and Trends"that discusses technological requirements, challenges and trends towards future network architectures. His main working area is communication networks, where he has expertise in project, modeling, simulation, performance evaluation and optimization of such networks. His current interests include future networks design, cognitive and autonomic networks, indirection resolution, entities identification, virtualization and future enabling technologies.