Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

ESTUDO DAS INTERFACES PARLAY E CONTRIBUIÇÃO PARA UM GATEWAY PARLAY X APLICÁVEL NAS NGNS

RODRIGO PIMENTA CARVALHO

NOVEMBRO/ 2008

ESTUDO DAS INTERFACES PARLAY E CONTRIBUIÇÃO PARA UM GATEWAY PARLAY X APLICÁVEL NAS NGNS.

RODRIGO PIMENTA CARVALHO

Dissertação apresentada ao Instituto Nacional de Telecomunicações, como parte dos requisitos para obtenção do Título de Mestre em Engenharia Elétrica.

ORIENTADOR: Prof. Dr. Antônio Marcos Alberti.

Santa Rita do Sapucaí

2008

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em 07 / 11 / 2008,
pela comissão julgadora:

_____
Prof. Dr.Antônio Alfredo Ferreira Loureiro / DCC - UFMG

_____
Prof. Dr. José Marcos Camara Brito / DTE - INATEL

_____
Prof. Dr. Antônio Marcos Alberti / DTE- INATEL

_____
Coordenador do Curso de Mestrado

Ao INATEL e às pessoas estudiosas da área de Telecomunicações.

AGRADECIMENTOS

Primeiramente, agradeço a **Deus** por toda a Sua ajuda poderosa durante o curso de mestrado.

Agradeço também à minha esposa incansável, por vários apoios valiosos durante a realização deste trabalho e também durante os anos do curso.

Agradeço pelo incentivo e apoio recebido dos meus pais e tios, por sempre me ajudarem a evoluir na vida acadêmica e profissional, da melhor forma.

Agradeço ao Professor Antônio Marcos Alberti, pela excelente orientação fornecida durante a elaboração deste trabalho.

Agradeço ao Professor Anilton S. Garcia, por sua amizade ao me recomendar duas vezes como aluno para o curso de mestrado do Inatel e agradeço também à profa. Celeste Vono, pelo apoio dado para resolver dúvidas de questões relacionadas com o idioma Inglês, principalmente ao redigir o documento (CARVALHO & ALBERTI, 2007).

Agradeço também a Julian Richards (*Parlay Technical Secretariat*), por ter esclarecido dúvidas sobre Parlay, a Sebastião Boanerges Ribeiro Junior (Brasil Telecom), Melissa Taboada do Lago e Rodrigo Luglio (Programa Ericsson Mobility World) e a Norberto Alves Ferreira (CPqD) por terem me indicado as ferramentas da Ericsson, para eu complementar as minhas pesquisas.

E agradeço a todos os colegas e funcionários do *Inatel Competence Center* e do CICT, ambos do Instituto Nacional de Telecomunicações, pela amizade.

Finalmente, agradeço à Erik Eriksson, da empresa Ericsson (Suécia), pela ajuda prestada com questões de computação, durante a implementação prática desta dissertação.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3GPP	*3rd Generation Partnership Project*
AIM	*AOL Instant Messenger*
AJAX	*Asynchronous JavaScript and* XML
API	*Application Programming Interface*
ARPA	*Advanced Research Projects Agency*, renomeada para *Defense Advanced Research Projects Agency* (DARPA) em 1972
B2B	*Business-to-business*
BBN	*Bolt, Beranek and Newman*
BT	*British Telecom*
CC	*Call Control*
CDMA	*Code division multiple access*
CEP	Código postal ou Código de Endereçamento Postal
CORBA	*Common Object Request Broker Architecture*
CPqD	Centro de Pesquisa e Desenvolvimento em Telecomunicações
CSCF	*Call Session Control Function*
DSL	*Digital Subscriber Loop* ou *Digital Subscriber Line*
ETSI	*European Telecommunications Standards Institute*
FTP	*File Transfer Protocol*
FW	*Framework*
GPRS	*General Packet Radio Service*
GPS	*Global Positioning System*
GSM	*Global System for Mobile communications*
GW	*Gateway*
HTTP	*Hypertext Transfer Protocol*
ICT	*Information and Communication Technologies*
IETF	*Internet Engineering Task Force*
IIT	*International Institute of Technology* (Em Montreal)
IM	*Instant Messaging*
IMS	*IP Multimedia Subsystem*
INAP	*Intelligent Network Application Part*
IP	*Internet Protocol*

ISDN	*Integrated Services Digital Network*
ITU	*International Telecommunication Union*
ITU-T	Setor do ITU que coordena padrões para as telecomunicações
J2EE	*Java Platform, Enterprise Edition*
JAIN	*Java APIs for Integrated Networks*
JAIN-SIP-API	API JAIN para SIP.
Java EE	*Java Platform, Enterprise Edition*
JAX-WS	*Java API for* XML *Web Services*
JCC	*Java Call Control*
JDK	*Java Development Kit*
JSP	*JavaServer Pages*
LAN	*Local-area Network*
MMS	*Multimedia Messaging Service*
MPLS	*Multi Protocol Label Switching*
MSC	*Mobile Switching Center*
NACF	*Network Attachment Control Functions*
NGN	*Next Generation Network*
NGNs	*Next Generation Networks*
NGN-FG	*NGN Focus Group*
NGN-GSI	*Next Generation Network Global Standards Initiative*
NIST	*National Institute of Standards and Technology*
NIST-SIP	Projeto do NIST para implementação da pilha SIP
OCS	FOKUS *Open Communication Server*
OMA	*Open Mobile Alliance*
OSA	*Open Service Access*
PAM	*Presence and Availability Management*
PDA	*Personal Digital Assistant*
PDF	*Portable Document Format*
PSTN	*Public Switched Telephone Network*
QoS	*Quality of service*
RACF	*Resource and Admission Control Functions*
RFC	*Request for Comments*

RPC	*Remote Procedure Call*
SCS-CC	*Service Control Server – CC*
SCS-SMS	*Service Control Server - SMS*
SIMPLE	*Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions*
SIP	*Session Initiation Protocol*
SJSAS	*Sun Java System Application Server*
SLA	*Service Level Agreement*
SMS	*Short Message Service*
SMS-C	*Short Message Service Center*
SMTP	*Simple Mail Transfer Protocol*
SOAP	*Simple Object Access Protocol*
TI	*Tecnologia da Informação*
TISPAN	*Telecoms & Internet converged Services & Protocols for Advanced Networks*
TPC	*Third Party Call service* ou *ThirdPartyCall interface*
UDDI	*Universal Description, Discovery and Integration*
UI	*User Interaction*
UML	*Unified Modeling Language*
UMTS	*Universal Mobile Telecommunications System*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
VoIP	*Voice-over-Internet Protocol*
W3C	*World Wide Web Consortium*
Wi-Fi	Nome comercial para uma tecnologia popular de comunicação *wireless*. Lembra ou sugere "*Wireless Fidelity*"
WSDL	*Web Services Description Language*
WCDMA	*Wideband Code Division Multiple Access*
xDSL	Tecnologias DSL
XML	*Extensible Markup Language*
XMPP	*Extensible Messaging and Presence Protocol*
XSD	XML *Schema*

RESUMO

A convergência tecnológica sobre IP está se firmando como um marco sem volta, que permite maior flexibilidade, velocidade de desenvolvimento, economia de escala, diversificação de serviços, maior interatividade, pulverização do acesso à rede, além de um mundo inteiro de novas oportunidades. De olho neste cenário, e preocupadas com a redução de seus faturamentos, as operadoras de telecomunicações têm fomentado o desenvolvimento de novas tecnologias capazes de renovar seus plantéis de serviços e aplicações. Uma das principais propostas neste sentido é a exposição das capacidades e serviços das redes de telecomunicações através de *gateways* que implementam APIs abertas. Um exemplo de serviço é o SMS. A possibilidade de envio de mensagem de texto curto, para um ou mais dispositivos móveis, simultaneamente, é um exemplo de capacidade da rede de uma operadora, provida pelo serviço SMS. A aposta é que, através destas APIs, uma quantidade significativa de desenvolvedores de *software* se juntará à indústria de telecomunicações, trazendo novas e criativas aplicações. De fato, os profissionais experientes em Tecnologia da Informação, com o uso de tecnologias de desenvolvimento de *software*, tais como Java, XML, JSP e *Web Services*, completam o ambiente de prestação de serviços inovadores das operadoras, porque trazem consigo os conhecimentos que não são de domínio dos profissionais de Telecomunicações. Por outro lado, os profissionais de TI, ávidos por entrar no mercado de telecomunicações, mas sem destreza com as tecnologias deste mercado, necessitam de *interfaces* padrões as quais possibilitem requisições aos serviços das redes das operadoras. Portanto, APIs que descrevem as capacidades das operadoras (e o domínio sobre como usá-las) são as 'pontes' entre o mundo de TI e de Telecomunicações. É a convergência TI/Telecom ou ICT. Para que esta idéia se concretize, deve haver trabalhos focados na implementação destas APIs e em como torná-las disponíveis. Este é o primeiro foco deste trabalho: mostrar como os serviços e funcionalidadades destas redes podem ser expostos apropriadamente, a terceiros, através de *interfaces* Parlay implementadas em *gateways* junto às redes convergentes de próxima geração. Outro foco importante foi a definição de uma metodologia para alcaçar este objetivo. Finalmente, explorou-se a emulação de um *gateway* com *interfaces* Parlay, demonstrando e discutindo as tecnologias necessárias a sua

utilização, além de contribuir efetivamente para a construção de um emulador de *gateway* deste tipo, que está sendo desenvolvido pela empresa Ericsson da Suécia.

Palavras-chave: NGN, Gateway Parlay, Web Service, Ericsson.

ABSTRACT

The technological convergence toward an IP network has been a landmark without a return that allows more flexibility, development speed, money saved, services diversification, interactivity, network access spray, besides a whole world of new opportunities. Looking at this scenario, and concerned with the invoices reduction, the telecommunications companies have fomentted the new techologies development, able to modernaize 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, that implement open APIs, such as: Parlay/OSA and Parlay X. 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, XML, JSP, and Web Services complete the new companies outsourced job's environment, because they, themselves, bring knowledges, that are not in the Telecommunications professionals' domain. On the other hand, the IT professionals, seeking entering the Telecommunications market, but without having skills about the technologies from this market, need the common interfaces, which make possible the companies networks requirements' skills. Therefore, the APIs that describe the companies capabilities (and the domain about using them) are the "bridges" between the IT and Telecommunication worlds. It is the IT/Telecom or ICT convergence. To this idea becomes true, there must be works focused on those APIs implementation, and how to become them available. That is the first focus of this work: to show how these networks and fucntionalities can be exposed properly to someone else, through Parlay Interfaces, implemented in gateways, together with the next generation converged networks. Another important focus was a methodology definition to reach this purpose. Eventually, it was explored Parlay interfaces with gateway emulation, showing and discussing the necessary technologies to their use, besides contributing effectively for the development of a gateway emulator of this kind, that has been developed by Ericsson Company from Sweeden.

Key words: NGN, Gateway Parlay, Web Service, Ericsson.

1. Introdução

Atualmente, pensar na utilidade de um telefone móvel, na utilidade da Internet ou na utilidade de um aparelho de televisão, por exemplo, não é pensar somente na transmissão da informação de uma ponta a outra, de quem fornece a informação a quem necessita dela. De fato, a utilidade atual dos dispositivos pessoais de telecomunicações pode abranger valores maiores que aqueles advindos da comunicação existente outrora entre as pessoas, por meios eletrônicos. O significado disso é que tanto as pessoas, usuárias comuns das telecomunicações, quanto as empresas fornecedoras de infra-estrutura nesse campo, bem como desenvolvedores de software, já perceberam que há demanda por serviços inovadores que utilizem meios de comunicações e novas capacidades das redes de telecomunicações. Por exemplo, os usuários de telefonia móvel já não querem mais apenas estabelecer contatos telefônicos entre si, mas também querem utilizar os seus dispositivos móveis para usufruírem de serviços que simplifiquem as suas vidas ou que possibilitem entretenimento. Considerando, por exemplo, que o envio de mensagens curtas de texto entre dispositivos móveis é uma nova capacidade das operadoras de telecomunicações, então um serviço capaz de enviar mensagens aos usuários assinantes do mesmo, com dados sobre a situação do trânsito nas principais avenidas, na hora do almoço, torna-se algo de valor e pode simplificar a tarefa de chegar em casa ou no local de trabalho, em certas localidades. Portanto, valores adicionais podem ser criados sobre as estruturas de telecomunicações, possivelmente com o apoio das tecnologias de *software*, se novos serviços úteis tornarem-se disponíveis nesse ambiente. Isso implica que novas aplicações nas telecomunicações, resultantes de novas capacidades das operadoras nessa área, poderão ser criadas com valores agregados, i.e., com funções inteligentes construídas com a criatividade de seus

criadores, por meio de *software* e tecnologias de redes de transmissão de dados. Neste cenário, pode-se ter uma grande oferta de novos serviços, por parte das operadoras de telecomunicações, com o mesmo objetivo de sempre: conquistar mais usuários ou clientes, garantindo, se não aumentando, o retorno do investimento neste mercado. Neste caso, será de grande valia, para as operadoras, o surgimento de várias aplicações inovadoras de *software*, suportadas por capacidades dessas operadoras, aplicações estas desenvolvidas por profissionais de Tecnologia da Informação. Para isso, porém, faz-se necessária uma organização na infra-estrutura das redes, criando uma camada de fornecimento de serviços desvinculada, conceitualmente, da camada física de transporte de dados. Ou seja, é necessário que os desenvolvedores de aplicações com valor agregado tenham as condições adequadas para o planejamento, implementação, teste e aplicação de suas invenções na área de *software,* atreladas à área de telecomunicações, sem que seja necessário conhecer detalhes técnicos desta segunda área, como protocolos de rede. Uma conseqüência desta necessidade é a padronização das *interface*s de acesso às capacidades das operadoras de telecomunicações, algo necessário às aplicações com valor agregado a serem concebidas por terceiros, os quais estarão trabalhando fora do domínio de tais empresas, mas deverão saber como o seu *software* poderá se comunicar com os recursos das redes dessas operadoras.

Uma *interface* de acesso a um grupo de capacidades de uma rede de telecomunicações é, inicialmente, uma documentação que lista quais são os nomes das funções acessíveis, via *software*, em tal rede. Estas funções, quando implementadas e executadas numa dada ordem, desenvolvem algum trabalho conforme as capacidades da rede. Por exemplo, se o envio de mensagens curtas de textos é uma capacidade, então deve haver funções, acessíveis a uma aplicação de *software*, para o envio e recebimento das mensagens, bem como para a coleta de dados de *status* sobre o envio de uma mensagem, etc. O conjunto destas funções forma uma *interface* e deve estar documentado para o público interessado em construir as aplicações que demandarão este tipo de serviço da rede. Além disso, a *interface* explica como invocar a execução destas funções, por exemplo, passando a

elas os parâmetros necessários, etc. Posteriormente, se uma rede de telecomunicações deve expor (tornar acessível) uma capacidade a terceiros interessados, então a forma padronizada para essa exposição será manter funções executáveis, acessíveis em algum ponto do seu próprio domínio, que se comportarão como descrito na especificação da *interface* respectiva. Desta forma, seguindo a mesma documentação, o desenvolvedor de uma aplicação poderá construir o seu *software,* que será capaz de interagir com as funções da rede, para requerer o trabalho de algumas capacidades. Para cada capacidade de uma rede, pode haver uma *interface* documentada. O conjunto das *interfaces* pode ser chamado de *Interfaces* de Programação de Aplicação ou *Application Programming Interfaces* (APIs) (DEITEL & DEITEL, 2003). Pelo visto, uma *interface* estabelecida entre uma rede de telecomunicações e uma aplicação de terceiro está para estas partes assim como uma *interface* gráfica de um *software* está para ele próprio e o seu usuário.

Resumidamente, um serviço numa rede de telecomunicações depende de um conjunto de capacidades. O conjunto dessas capacidades pode ser agrupado, conceitualmente, numa ou mais *interfaces*. Uma *interface* descreve uma capacidade da rede e suas utilidades (funções da *interface*).

Este documento discute sobre as *interfaces* sendo padronizadas atualmente, as vantagens desta padronização, como ela será fornecida ao mercado e como será empregada nos sistemas reais das operadoras de telecomunicações. Além disso, há também uma explanação sobre uma proposta de contribuição em *software* ao esforço da empresa Ericsson para a concretização, divulgação e liberação desse tipo de padronização, promovendo uma ponte entre o "mundo de Tecnologia da Informação" e o "mundo de Telecomunicações". As *interfaces* discutidas aqui declaram quais são as capacidades de uma rede acessíveis em sua camada de serviços.

A divisão das redes em uma camada de serviços e mais uma camada de transporte, com o intuito de organizar e separar o domínio de prestação de serviços do domínio de tecnologias de transporte de dados, pelo menos conceitualmente, é um dos objetivos das redes de próxima geração, ou *Next Generation Networks* (NGNs) (INTERNATIONAL TELECOMMUNICATION UNION; ITU-T, 2004). As redes de próxima geração abrangerão a convergência de tecnologias de telecomunicações, onde várias técnicas de transporte de informação serão suportadas pelo *Internet Protocol* (IP) (KUROSE, et al., 2006), (O Horizonte das Redes de Próxima Geração, 2002), tanto que estas redes futuras são chamadas de redes *ALL-IP*. Neste ambiente, poderão surgir novas aplicações embasadas por protocolos que trabalham sobre o *Internet Protocol*, como o *Hyper Text Transfer Protocol* (HTTP). Mais precisamente, aplicações com valores agregados poderão usar serviços de redes de telecomunicações e estes poderão usar o *Internet Protocol* de alguma forma. O importante é que a formulação das aplicações com valores agregados seja agnóstica aos detalhes de telecomunicações e aos protocolos de mais baixo nível.

Por parte das operadoras de telecomunicações existe um desejo de, realmente, facilitar e agilizar o surgimento destas aplicações, visando mais tráfego multimídia em suas redes, gerando mais lucro. Portanto, operadoras de várias partes do mundo, empresas de produção de elementos de rede, órgãos formuladores de padrões em telecomunicações e outras empresas de desenvolvimento de *software* unem-se para a formulação, em comum, da padronização das *interfaces* de acesso às redes. Um resultado dessa união é um consórcio chamado Parlay (THE PARLAY GROUP, 200-?). As *interfaces* especificadas por este consórcio, chamadas de *Interfaces* Parlay, demonstram quais serão os serviços disponíveis nas redes para as aplicações de terceiros. Ou seja, quais capacidades uma rede pode ter e tornar disponível àquelas aplicações, como envio de mensagem curta de texto, envio de mensagem com multimídia, localização de terminais, etc.

A implementação real destas *interfaces* constituirá um novo elemento nas redes, chamado de *gateway*, totalmente provido por *software*. Ou seja, um novo sistema de *software,* reunindo em si as implementações das *interfaces*, responderá às requisições de aplicações de terceiros, as quais invocarão métodos neste sistema, descritos nessas *interfaces*. O *gateway* poderá, então, se comunicar com elementos de rede e requisitar serviços de telecomunicações, como o estabelecimento de chamadas entre terminais e, ao mesmo tempo, retornar informações de *status* de serviços em execução, às aplicações. Um *gateway* será propriedade de uma operadora de telecomunicações, o que constituirá uma espécie de abertura controlada das redes desse tipo de empresa.

A empresa Ericsson, na Suécia, com o intuito de facilitar o acesso a um *gateway* com *interfaces* Parlay, para a comunidade de desenvolvedores de *software*, está construindo um emulador deste tipo de *gateway*. A *interface* do emulador será idêntica àquela que estaria presente num *gateway* real e as formas de comunicações com o *gateway* emulado serão idênticas àquelas considerando o real. Neste caso, as aplicações de terceiros, demandando o uso de capacidades de redes de telecomunicações, poderão ser testadas, diretamente, interagindo com o emulador. A diferença entre o *gateway* real e o emulado é que o primeiro interage com sistemas e/ou elementos reais numa rede real, enquanto que o segundo emula tais ações. Como o emulador ainda está em fase de construção, esta dissertação mostra como foi realizado um trabalho de contribuição para o mesmo.

Novas funcionalidades foram acrescidas ao emulador de *gateway* da Ericsson, sendo que o código fonte deste trabalho já foi enviado ao time de desenvolvedores daquela empresa, na Suécia, para ser analisado e testado, recebendo o comentário de que, provavelmente, fará parte do código fonte oficial daquele emulador, em sua próxima *release*.

Assim sendo, o objetivo deste trabalho de mestrado é mostrar como expor, apropriadamente a terceiros, as funções das redes de telecomunicações, através de *interfaces* Parlay implementadas por *gateways* adequados. Tem-se, também, como objetivo, emular um exemplo de *gateway* com *interfaces* Parlay, demonstrando e explicando as tecnologias para a utilização do mesmo, além de contribuir, efetivamente, para a construção de um emulador de *gateway* deste tipo, da empresa Ericsson, para facilitar a implantação de aplicações com valores agregados nas redes de próxima geração.

1.1 Organização do trabalho

O restante desta dissertação é organizado como segue: O capítulo 2 dessa dissertação comenta sobre as características da convergência tecnológica em telecomunicações, o que abrange conceitos das *Next Generation Network*s. Aponta também uma visão breve de mercado, relacionada com a importância do surgimento das novas aplicações com valor agregado para as operadoras de telecomunicações. Além disso, este capítulo explica sobre as capacidades das redes de próxima geração e comenta sobre os serviços a serem criados. O Capítulo 3 comenta sobre os esforços da indústria de telecomunicações para a criação das *interfaces* padronizadas e mostra tecnologias relacionadas com a implementação real das mesmas. Nesse capítulo, fica perceptível o grande potencial de contribuição da linguagem de programação Java à indústria de telecomunicações (SUN MICROSYSTEMS, 2004). Além disso, comenta sobre um grupo de *interfaces* do Parlay, mais simples, para desenvolvedores de *software* sem *expertise* em *networks*, que são chamadas de **Parlay X**. O Capítulo 4 apresenta o emulador de *gateway* Parlay da Ericsson, mostrando algumas de suas funcionalidades. O Capítulo 5 explica sobre o trabalho de *software* realizado para a adição das novas funcionalidades ao emulador da Ericsson, comentando detalhes no nível de código fonte implementado. Esse capítulo termina o estudo feito nesta dissertação, mostrando a arquitetura final que envolve o *gateway* da Ericsson e eventos de uso da nova funcionalidade junto a este *gateway*. Esta nova

funcionalidade, agora presente no emulador, consiste na capacidade de emular terminais móveis ou fixos, participando de chamadas estabelecidas entre eles, requisitadas por aplicações de terceiros, o que é chamado de *Third Party Call,* nas especificações do Parlay. A conclusão deste trabalho faz um resumo do que foi pesquisado, estudado, implementado e aprendido em relação ao assunto de *interfaces* de acesso às capacidades das redes de telecomunicações, em relação ao Parlay e em relação às ferramentas que ajudam na criação de *gateways*. A conclusão termina sugerindo novos trabalhos para ajudar no progresso do esforço da indústria em criar aplicações inovadoras atraentes às pessoas e geradoras de lucros para as empresas.

1.2 Artigo Publicado

Durante o período de execução deste trabalho foi realizada a publicação do artigo (CARVALHO & ALBERTI, 2007): JAVA TECHNOLOGIES FOR NGN SERVICE CREATION: DISCUSSION AND ARCHITECTURE TO IMPROVE SIP ADDRESSES DISCOVERY. Este artigo foi publicado em Chamonix, França, em março 2007, no evento *International Conference on Internet and Multimedia Systems and Applications* (EuroIMSA).

Este artigo tratou de vários assuntos relacionados com as questões discutidas nesta dissertação, como as interfaces abertas para as NGNs, além de tecnologias como JAIN, SIP, NIST-SIP e JINI. O propósito do artigo foi demonstrar uma arquitetura de *software*, baseada em JINI, capaz de possibilitar a busca por identificadores de usuários de sistemas SIP, registrados em qualquer *registrar* de qualquer domínio e identificáveis com poucos atributos de seus perfis.

2. Convergência Tecnológica, Visão de Mercado e NGN

A rede de telefonia fixa, comutada por circuitos, transmitiu somente voz, de telefone a telefone, durante um grande período de sua existência. Depois daquele período, na década de 60 (século XX), segundo (TANENBAUM, 1997), esta mesma rede passou a transmitir também dados provenientes de minicomputadores. Tal transmissão de dados se deu a partir de uma necessidade de projeto da empresa *Bolt, Beranek and Newman* (BBN), a pedido da divisão científica do Pentágono, a *Advanced Research Projects Agency* (ARPA), quando foi necessário criar uma rede de minicomputadores para esta agência. À estrutura da rede telefônica, que havia sido construída para exercer uma função (transmissão de voz), foi acrescida uma nova capacidade (transmissão de dados de computadores). O que ocorreu, então, foi a utilização de uma única infra-estrutura de tecnologia para fornecer serviços que, anteriormente, requeriam equipamentos e canais de comunicação independentes. Até então, dados digitais de computadores eram transmitidos em redes de computadores, criadas para este fim. Mas, com a nova utilidade da rede de telefonia, os computadores também puderam trocar informações através dessa estrutura de rede que, inicialmente, não havia sido criada especificamente para isso. Assim, uma estrutura física de rede, já instalada, pôde ser explorada mais uma vez, porque passou a fornecer um novo serviço. Tal novo serviço foi mais uma transmissão de sinais elétricos, como já ocorria, mas encarados como dados e, portanto, com valor agregado diferente dos sinais de voz. Naquele cenário, dois serviços de valores diferentes e tecnologias diferentes convergiram num mesmo canal de atendimento às suas diferentes necessidades.

Quando dois ou mais serviços independentes em tecnologia e/ou duas ou mais tecnologias independentes entre si passam a ser suportadas sobre alguma tecnologia específica, tem-se os chamados serviços convergentes e/ou a convergência tecnológica. E, quando ocorre a convergência tecnológica, por exemplo, sobre um mesmo canal de comunicações, há a possibilidade de se diversificar o uso do mesmo, otimizando este uso para atender a diversas necessidades do usuário, como explicado em (CONVERGÊNCIA tecnológica, 2008).

A tendência para a convergência de tecnologias, nos meios de comunicações, tornou-se mais difundida com a popularização da Internet nos últimos anos. Anteriormente, com o acesso doméstico à Internet, através de conexões discadas, não se podia ter uma qualidade de serviço suficiente para certas necessidades, como o uso do *Voice-over-Internet Protocol* (VoIP), que é um exemplo de convergência de tecnologias. Aquele foi um período em que os usuários domésticos de Internet pouco ouviam falar sobre a convergência de serviços e tecnologias, numa mesma estrutura de comunicações já disponível. Posteriormente, com a adoção de tecnologias como a *Digital Subscriber Line* (DSL) (xDSL), por exemplo, o acesso à Internet tornou-se mais viável para o uso de serviços convergentes, porque esta tecnologia possibilitou, a um custo acessível, uma capacidade de transmissão suficiente para a utilização destes serviços, de acordo com (CONVERGÊNCIA tecnológica, 2008).

Atualmente, as convergências tecnológicas no setor de telecomunicações vêm despertando a atenção das pessoas para novos serviços. Por exemplo, a convergência de transmissão de dados, vídeo e voz através de um mesmo canal de comunicação, que existe no mercado, suportada por algumas operadoras de TV a cabo, como a empresa NET com seu serviço chamado *Virtua*, surgiu justamente para suprir uma demanda reprimida por vantagens oferecidas com tal convergência. Quando se pode oferecer um único serviço de transmissão de dados, vídeo e voz, ou no mínimo, dois destes, através de um mesmo canal, pode-se também ter apenas uma base de assinantes controlada pelas empresas operadoras destas transmissões. Ou seja, uma

operadora de telefonia e uma de televisão a cabo que se unem para prestarem serviços de vídeo e conversação sobre uma mesma estrutura de rede, podem unir suas bases de assinantes, aumentando, então, o número de clientes de ambas as empresas. Se dois serviços diferentes podem ser oferecidos aos mesmos clientes, ao mesmo tempo, isso possibilita a criação de promoções, a fim de que tais clientes realmente se tornem usuários dos dois serviços. Além disso, cada usuário, mesmo usufruindo de serviços diferentes, receberá somente uma única conta para pagamento no final do mês. Como um exemplo do mercado, pode ser citada a parceria da NET com a Embratel, colocada em prática em 2006 e publicada no *web site* FOLHAONLINE em (FolhaOnline, 2006), para vender serviços de telefonia fixa – principalmente para assinantes da TV paga e Internet rápida (banda larga). Neste novo serviço oferecido por estas empresas, a voz trafega pela rede de cabos da Net. Então, a convergência tecnológica pode vir em paralelo com a união de empresas detentoras de tecnologias diferentes que, quando unificadas, por exemplo, numa infra-estrutura comum, criam capacidades para o surgimento de novos serviços, que podem trazer retorno econômico adicional a estas empresas, além da diversificação de serviços ofertados a seus clientes.

2.1 O Mercado

De fato, a convergência tecnológica ou convergência de serviços, que são citadas atualmente na Internet, em revistas de tecnologias ou sobre negócios e em campanhas de *marketing*, estão, geralmente, relacionadas com a convergência de, no mínimo, dois de três tipos de serviços de comunicações: de dados, de vídeo e de voz. Na verdade, os usuários de serviços de telecomunicações já perceberam que podem e, realmente, querem ter mais do que a possibilidade de se comunicarem usando apenas conversação. Está claro, então, que a influência da indústria de telecomunicações, na vida dos usuários, está se expandindo para além de serviços que permitem às pessoas falarem entre si apenas com voz. Ou seja, esta indústria está expandindo a sua influência na vida dos usuários com serviços que utilizam som,

texto e vídeo, como citado em (THE PARLAY GROUP, 2005). Estes usuários estão ávidos por aplicações inovadoras, dito em (INTERNATIONAL ENGINEERING CONSORTIUM, 2005), como aquelas que utilizam *Short Message Service* (SMS), *Multimedia Messaging Service* (MMS), transmissão de vídeo, localização de usuários e tudo mais que possa ser imaginado e criado, e que nos levará às aplicações inovadoras, futuramente. SMS é um serviço pelo qual um usuário pode enviar mensagens de texto curto para o dispositivo de outro usuário, tal como enviar uma pequena mensagem para o celular de uma pessoa. MMS é um serviço semelhante ao SMS, mas com a possibilidade de envio de arquivos anexados às mensagens de texto curto. Segundo (ROSA, 2006), "as necessidades do usuário por novos serviços e aplicações criam uma grande oportunidade de expansão de negócios". Conseqüentemente, as operadoras, como as de telefonia fixa, que continuarem provendo apenas serviços de conversação estarão fadadas a serem fornecedoras de mais uma *commodity* no mercado, como é o caso de energia elétrica e água. Além disso, com as possibilidades oferecidas pela convergência tecnológica em telecomunicações, como o caso de transmissão de voz pela Internet, através de aplicativos que utilizam VoIP, por exemplo o *software VoIPDiscount* (VoipDiscount, 2008), os usuários agora têm alternativas de serviços que os libertam da dependência exclusiva das grandes operadoras de telefonia, as quais outrora detinham uma espécie de monopólio no mercado de telecomunicações. Esta liberdade conferida aos usuários implica em redução de faturamento para as operadoras.

É verdade que o uso das linhas fixas vem caindo constantemente por causa do aumento na utilização de *e-mails* e serviços de telefone via Internet, tanto que a British Telecom teve queda de 11% do lucro, no quarto trimestre de 2005, comparado ao mesmo período no ano anterior, conforme a notícia (OSSE, 2006). A telefonia fixa registrou queda de 0,5% de faturamento em 2006, no Brasil. A Brasil Telecom, por exemplo, fez uma limpeza na sua base de assinantes, dando baixa em 667 mil acessos fixos no terceiro trimestre, segundo (MOREIRA, 2006). As concessionárias registraram uma redução de 1,5 milhão na quantidade de telefones fixos em serviço nos dez primeiros meses de 2006, no Brasil, de acordo com

(BRASIL TELECOM, 2007). Paralelamente às perdas de faturamento na telefonia fixa, existe também a concorrência entre as operadoras deste tipo de serviço e delas com as operadoras de telefonia móvel. Ou seja, as operadoras competem entre si no aumento de suas bases de assinantes, visando mais lucro.

Devido ao atual contexto de mercado desfavorável às companhias telefônicas que não inovarem em seus serviços, como descrito acima, as bases de assinantes destas empresas poderão reduzir, causando a queda no faturamento, caso tais operadoras não criem estratégias eficientes, para manterem e conquistarem mais assinantes. Este contexto desfavorável é uma realidade mundial, como mostra a análise setorial (Valor Online, 2006). Até o momento presente, os serviços de comunicação de dados constituem-se cada vez mais na melhor alternativa para as operadoras de telefonia fixa fugirem da relativa estagnação de suas receitas. Embora ainda representem pouco da receita bruta, são nestes nichos que as companhias telefônicas esperam crescer, ainda segundo (Valor Online, 2006).

Uma das estratégias das operadoras de telecomunicações, tanto as fixas quanto as móveis, para aumentarem suas bases de assinantes, será o provimento de novos serviços de valor agregado, os quais possam gerar mais tráfego na rede, consoante o documento (THE PARLAY GROUP, 2005). Realmente, segundo (ARTIGAS & NUNES, 2007), "em um mercado de grande competitividade como o mercado de telecomunicações, agregar novos valores em serviços é uma ação decisiva para o sucesso do negócio. Novas aplicações e serviços aumentam a fidelidade dos clientes e proporcionam novas fontes de receitas e uma nova janela para diversas oportunidades de negócios". Exemplos atuais de serviços de valor agregado são: troca de fotos, mensagens de texto, acesso à Internet, correio eletrônico, *download* de músicas e transmissão de vídeos ao vivo. No Brasil, os serviços de valor agregado ainda representam cerca de 6% das receitas das companhias de telefonia móvel, mas há uma previsão de que tais serviços deverão representar 7,3% da receita média por usuário das operadoras do país, por volta de 2010, de acordo com (Valor Online,

2006). Portanto, fica evidente que as operadoras necessitam de formas rápidas e econômicas de desenvolver novas aplicações com valor agregado. Resumindo: as companhias de telecomunicações estão apostando no fornecimento de serviços de transmissão de dados em banda larga, criação de novas aplicações de valor agregado e na convergência tecnológica de soluções para o setor destas empresas. Tudo isso para criar meios de conquistar mais clientes, garantir o retorno sobre o investimento e também evitar a evasão dos clientes atuais em direção à concorrência.

Contudo, a convergência tecnológica é aspirada pelas operadoras de telecomunicações, empresas prestadoras de serviços e desenvolvedores de aplicações, desde que haja viabilidade de desenvolvimento e comercialização em grande escala de soluções incitadas por essa convergência. Do ponto de vista técnico, com esta viabilidade, têm-se redes diferentes, criadas para serviços diferentes, convergindo com tecnologias em comum. Estas são as Redes Convergentes. Um exemplo de redes convergentes pode ser visto em[1], as redes de *broadcast*, que estão caminhando para a digitalização de suas infra-estruturas. A parceria da NET com a Embratel, já citada, constitui um exemplo disto.

2.2 NGN

Com a tendência atual de convergir serviços e tecnologias, mais popularmente através da Internet (ou em redes IP controladas), algumas tecnologias usadas nesta rede estão se estabelecendo como padrões para a criação de novos serviços, como o *Internet Protocol* (IP) e a comutação de pacotes. Estes elementos, acrescidos dos seguintes itens: meios de transmissão em banda larga, necessidade por serviços multimídia, criação de novos protocolos como o *Session Initiation Protocol* (SIP) e

---

[1] ALBERTI, A. M. **Redes Convergentes /** Curso de Mestrado em Telecomunicações ministrado no Inatel, Santa Rita do Sapucaí, no período de fevereiro-julho, 2007/

de mecanismos como o *Multi Protocol Label Switching* (MPLS); estão tomando parte na formação da arquitetura das redes convergentes, como dito em (NEXT generation networking, 2008). Estas redes também são chamadas de *Next Generation Networks* (NGNs) - ou Redes de Próxima Geração. Realmente, um dos objetivos das NGNs é possibilitar a convergência das transmissões das informações sobre uma rede totalmente baseada no protocolo IP, como comentado em (CASTRO & LOURENÇO, 200-)**.** Devido a isso, as NGNs também são chamadas de redes *ALL-IP*, conforme o comitê técnico do *Telecoms & Internet converged Services & Protocols for Advanced Networks* (TISPAN), em (EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE, 2008). O TISPAN é um centro de competência, do *European Telecommunications Standards Institute* (ETSI), em redes fixas e para a migração das redes de circuitos comutados em redes baseadas em pacotes com uma arquitetura que servirá para a implantação das NGNs.

Conforme o cronograma definido pelo *International Telecommunication Union* – setor de padrões em telecomunicações (ITU-T), que pode ser visto no *web site* (INTERNATIONAL TELECOMMUNICATION UNION, 2005-2008), a padronização das NGNs ainda se estenderá além de 2008 e, segundo (HEINISCH, 2006), "pode-se concluir que a NGN, com total convergência de serviços, ainda não é uma realidade. Mesmo os fabricantes apresentando produtos e serviços com o rótulo NGN, um cenário puramente NGN ainda está por surgir. Nem mesmo o trabalho de padronização das redes NGNs se encontra finalizado. Os órgãos de padronização continuam trabalhando na definição da arquitetura e dos serviços, e aspectos de segurança, entre outros, ainda estão por ser definidos". Contudo, em (CASTRO & LOURENÇO, 200-) está dito que, "apesar de dificuldades para se concluir a padronização das NGNs, a conversão das redes atuais neste padrão é um caminho irreversível. Todas as empresas deverão adotar, em um futuro razoavelmente próximo, as redes de próxima geração, visando sobreviver no competitivo ramo das telecomunicações". Estas empresas são operadoras de redes fixas e móveis, provedores de serviços, integradores de sistemas e empresas desenvolvedoras de aplicações, que deverão cooperar entre si para possibilitar a

provisão de serviços convergentes avançados, como citado em (MAGEDANZ, et al., 2004). "O sistema de parceria, em que cada um se concentra em suas excelências, vai ser o modelo ganhador para o consumidor, vai chegar de forma relevante e de forma acessível economicamente", segundo Lopes, da operadora Tim, entrevistado pela revista *B2B* (VASQUES, 2007).

De acordo com explicações encontradas em (EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE, 2008), (INTERNATIONAL ENGINEERING CONSORTIUM, entre 2006 e 2007) e (MOYER & UMAR, 2001), através das NGNs, as operadoras suportarão uma nova organização para a arquitetura das redes e também a forma como novos serviços poderão ser oferecidos sobre a arquitetura organizada. Desta forma, estas empresas poderão acelerar a oferta de suas novas aplicações de valor agregado, concorrendo com alternativas atuais de comunicação como o *Skype*, sendo que algumas ofertas de serviços dependerão, em parte, de características tecnológicas sob o domínio das operadoras. Por exemplo, com as NGNs, deseja-se uma separação arquitetural entre o ambiente dos serviços e a infra-estrutura da rede. Isto é, os serviços deverão ser planejados, arquitetados e implementados sem que a rede, que irá suportá-los, tenha alguma influência em alguma dessas 3 fases de construção dos mesmos. Na verdade, o que se quer é uma separação em duas camadas bem definidas: a arquitetura da rede conterá uma camada de serviços e uma de transporte. Isto significa que, sempre que um provedor de aplicações desejar tornar uma nova aplicação disponível na rede, ele poderá fazê-lo utilizando um ou mais serviços definidos diretamente na camada de serviços, sem considerar a camada de transporte, segundo (NEXT generation networking, 2008). Assim, os desenvolvedores de algumas aplicações poderão lidar somente com capacidades da rede declaradas na camada de serviços. Isto é, será possível desenvolver novos sistemas sem a necessidade de se conhecer detalhes de transporte de dados. Neste caso, a gama de desenvolvedores aptos a criar novas aplicações para as redes convergentes estará aumentada e, conseqüentemente, estará também a possibilidade de surgimento de novas aplicações rentáveis às operadoras.

A necessidade de conhecer detalhes somente da camada de serviços realmente facilitará o aparecimento de novas aplicações nas NGNs.

Entretanto, capacidades chaves das redes, necessárias às novas aplicações e declaradas na forma de *interfaces* públicas na camada de serviços, estarão acessíveis somente através de *gateways* disponíveis nos domínios das operadoras, como exemplificado em (MAGEDANZ, et al., 2004). Serão os *gateways* que conterão as implementações destas *interfaces* e será desta forma, através de *gateways*, que as operadoras poderão barrar o acesso às capacidades de suas redes às aplicações julgadas inviáveis comercialmente. O controle através de *gateways* possibilitará, por exemplo, identificação, autenticação e liberação de acesso de uma aplicação às capacidades da rede, bem como a análise do uso das mesmas, para gerar cobranças pelos serviços prestados pela rede. Um *gateway* poderá, então, decidir pela liberação de acesso a uma capacidade, o que corresponderá a liberar interação com alguma *interface* declarada no mesmo. Por exemplo, supondo que um *gateway* dá acesso a um serviço de criptografia, então através do mesmo uma aplicação poderá usufruir de tal capacidade computacional. Para isso, a aplicação deverá interagir com alguma *interface* contendo métodos de criptografar e descriptografar, por exemplo. Isso implica que o *gateway* disponibilizará uma *interface* para o propósito de criptografia e a implementação de tal *interface* pode ser provida pelo próprio *gateway* ou por algum serviço de criptografia disponível em algum lugar da rede, por exemplo.

2.3 Interfaces de Serviços em NGN

No ITU-T há uma iniciativa chamada *Next Generation Network Global Standards Initiative* (NGN-GSI) que foca no desenvolvimento de padrões detalhados, necessários para o emprego das NGNs, de tal forma que existam meios para os provedores de serviços oferecerem uma ampla variedade de aplicações esperadas nessas redes. Mas, antes do NGN-GSI, em 2004 o ITU-T criou o NGN Focus

Group, para lidar com necessidades urgentes por padrões globais para as NGNs relacionados com requisitos de serviços, *Quality of service* (QoS) e a evolução da rede baseada em pacotes, por exemplo. Para as NGNs, a qualidade de serviço está definida pelo NGN-GSI, como uma característica obrigatória das redes. Ou seja, uma NGN, se construída, deverá suportar QoS, mesmo trabalhando sobre o protocolo IP. Então, se alguém necessita criar uma nova aplicação para uma NGN, a QoS já poderá ser considerada um problema resolvido. Na prática, isto poderá ser garantido com mecanismos como o MPLS, já comentado anteriormente neste documento.

O NGN Focus Group teve a sua última reunião realizada em novembro de 2005. Atualmente, este trabalho vem sendo continuado pelo NGN-GSI. A página, *on-line* na *web*, do NGN-GSI pode ser vista no endereço (INTERNATIONAL TELECOMMUNICATION UNION, 2007). O NGN Focus Group (NGN-FG) era formado por sete grupos de trabalho, sendo que cada grupo estava focado em questões específicas diferentes sobre as NGNs. Por exemplo, questões sobre: serviços e capacidades da rede, arquitetura funcional e requisitos, qualidade de serviço, aspectos de controle, questões de segurança, entre outros. Estes grupos produziram vários documentos sobre tais questões e realizaram vários encontros entre 2004 e 2005. Durante o último encontro em 2005, em Genebra, o NGN-FG apresentou o resultado final do trabalho dos sete grupos, constante no artefato (INTERNATIONAL TELECOMMUNICATION UNION, 2005b) e considerado como a *Release I* de NGN.

Algumas das características das NGNs vislumbradas pelo ITU-T, de acordo com as notas de aula[2]**,** e que são componentes da definição de uma nova arquitetura de rede, como comentada anteriormente, são:

-Transferência baseada em comutação de pacotes IP;

---

[2] ALBERTI, A. M. **Redes Convergentes /** Curso de Mestrado em Telecomunicações ministrado no Inatel, Santa Rita do Sapucaí, no período de fevereiro-julho, 2007/

-Separação das funções de controle em transporte, chamada/sessão e serviços/aplicações;

-Separação do fornecimento de serviço do tipo de rede e fornecimento de serviços via interfaces abertas;

-Suporte para uma grande variedade de serviços e aplicações, baseados na construção modular de serviços.

As separações e o fornecimento citados acima serão um desacoplamento entre transporte e serviço, do ponto de vista da definição das arquiteturas e funcionalidades dessas duas camadas e essa visão das NGNs apresenta-se adequada a um ambiente de produção de novos serviços baseados em *software*. Quando se tem uma camada de serviços separada de uma camada de transporte, em termos de arquitetura, os esforços e planejamentos para novos projetos de *software* não precisam levar em consideração detalhes de baixo nível, como tipos de protocolos de rede e como usá-los. Isto possibilita que as atenções sejam totalmente focadas nas funcionalidades dos serviços da rede, em alto nível, quando novas aplicações de valor agregado poderão ser projetadas baseadas em tecnologias de Computação e independentes das características das redes que as conterão. Então, se a produção de novas aplicações estiver desvinculada das características da rede, a implementação das mesmas poderá ser feita até mesmo por profissionais sem conhecimentos detalhados de Telecomunicações. Ou melhor, as novas aplicações, construídas como *software*, poderão ser implementadas por especialistas de Tecnologia da Informação (TI), sem grandes dificuldades.

Se novos sistemas com valor agregado poderão ser feitos para a indústria de Telecomunicações, com o trabalho de especialistas em TI, então esta indústria atrairá um novo 'exército' de profissionais que já trabalham com tecnologias úteis às futuras aplicações que estarão utilizando as capacidades das redes das operadoras, por exemplo. Uma das conseqüências dessa atração por novos profissionais, capazes de implementar aplicações inovadoras de comunicações, será o aumento da chance de surgimento de novas aplicações com valor agregado. Tudo isso é verdade porque muitos projetistas, arquitetos e programadores de *software*, ao entenderem as reais

possibilidades de acesso e utilidades das capacidades das redes de telecomunicações, se voltarão para o mercado das operadoras e prestadores de serviços em NGN, já que há uma demanda reprimida por aplicações úteis com telecomunicações para o dia-a-dia das pessoas. Portanto, a organização tecnológica proposta pelas NGNs criará novas oportunidades financeiramente viáveis para os profissionais de TI, os quais ganharão um novo campo de trabalho a ser explorado com fornecimento de novos projetos de *software*, mas também criará novas oportunidades às operadoras atualmente ávidas por suprir, rapidamente, a demanda por aplicações com valor agregado. Na medida em que as empresas da indústria de telecomunicações puderem suprir tal demanda, ocorrerá também uma maior atratividade para assinantes de novos serviços, ou seja, um maior retorno sobre o investimento nas tecnologias de NGN. Como percebido, há um interesse por um mutualismo entre as indústrias de Telecomunicações e de TI e, realmente, as operadoras esperam que os profissionais de TI venham trabalhar em sistemas que funcionem no domínio de suas redes, para a criação de novas aplicações. Devido a isso, já existem iniciativas, no setor de Telecomunicações, para facilitar a adoção de mão-de-obra de TI.

Por exemplo, para a construção de uma aplicação com valor agregado, que utilize alguns serviços das redes, como SMS, MMS e outros, pode-se esperar que os profissionais de TI venham a projetá-la, implementá-la e testá-la sobre uma dada rede. Ou seja, uma aplicação que utilize SMS e outros serviços deverá funcionar por meio de recursos de, no mínimo, uma operadora de telecomunicações específica. Isto implica que as aplicações de terceiros serão clientes dos serviços de empresas como a Oi, Claro, Brasil Telecom, etc. Mas, será viável a esses profissionais uma forma de garantir que tais aplicações funcionem tanto numa rede de uma operadora, quanto em outra rede de outra operadora qualquer. Assim, o resultado do esforço para o desenvolvimento da aplicação não será afetado, isto é, ao executar a mesma aplicação em redes de operadoras diferentes, as funcionalidades da mesma ainda estarão operantes, sem a necessidade de alterações computacionais. Isto quer dizer que, havendo uma forma de padronizar a execução das novas aplicações sobre redes variadas, haverá também a possibilidade dos especialistas em TI aproveitarem ao

máximo os seus projetos, quando da troca de rede por parte das suas aplicações. Tal possibilidade é realmente almejada por estes profissionais. Mais precisamente, isso será uma forma padrão de expor a rede de telecomunicações às novas aplicações com valor agregado, ou melhor, do ponto de vista dessas aplicações, as redes poderão ser acessadas através de algumas *interfaces* padrões, como explicado em (CARUGI, 2006). Então, estas *interfaces* formularão uma visão de quais capacidades das redes estarão disponíveis num dado momento. Aqui se nota a grande importância de se organizar muito bem a padronização dessas *interfaces*, já que elas serão a conexão entre as aplicações de TI e os serviços da rede de uma operadora. Será essa padronização dos serviços, através de *interfaces*, que garantirá a portabilidade das aplicações entre as redes que estejam seguindo tal padronização. Esse tipo de padronização é um dos objetivos das NGNs. Uma conclusão é que as NGNs estão sendo projetadas também para facilitar a inserção de aplicações de terceiros nas redes das operadoras. Será através desse tipo de iniciativa, que é o uso de *interfaces* padrões e públicas, que as redes de telecomunicações poderão ser abertas aos criadores de aplicações inovadoras. Isto é uma espécie de globalização de aplicações de terceiros.

Como as redes serão abertas para propiciar o surgimento rápido de novas aplicações em seus domínios, dentre as *interfaces* deverá haver uma ou algumas encarregadas de controlar, seguramente, o acesso aos serviços da rede. Desta forma, espera-se que os serviços desenvolvidos por terceiros sejam submetidos a algum tipo de identificação, autenticação e autorização junto à rede, antes de usufruir da mesma, em tempo de execução. O objetivo disso é garantir que as aplicações de terceiros acessem as redes das operadoras segundo regras de segurança definidas e estabelecidas através das *interfaces* apropriadas para esse fim. Neste contexto, as definições das *interfaces* são de grande interesse das principais empresas da indústria de telecomunicações, como por exemplo a British Telecom, Telcordia, Ericsson, etc. Conseqüentemente, estas e outras empresas estão, realmente, definindo as *interfaces* abertas e padronizando esse trabalho para publicá-lo.

Mostrar uma forma de unir as tecnologias das Telecomunicações e TI, para a criação de novas aplicações para as NGNs, e como, efetivamente, criá-las, são algumas das explicações desta dissertação de mestrado, como será exemplificado mais adiante.

A partir das NGNs, as aplicações com valores agregados, interagindo com *interfaces* padrões abertas (públicas), poderão ser portadas em diferentes redes, como explicado anteriormente. Além disso, uma aplicação em uma rede poderá ser substituída por outra aplicação, de mesma utilidade, desde que a segunda utilize as mesmas *interfaces* utilizadas pela primeira. Desta forma, as substituições de aplicações não causarão efeito colateral em outros sistemas, mesmo que tais sistemas dependam das aplicações a serem substituídas. Por exemplo, a substituição de aplicações não implicará na substituição de serviços na rede, quando estas usarem *interfaces* padrões para a comunicação com estes. Nesse cenário, cada nova aplicação ou serviço será como um bloco numa rede formada por blocos. As trocas dos blocos poderão, então, ser realizadas, sem afetar o resto da estrutura da rede. Isto será verdade desde que os fabricantes de sistemas para a rede os construam 100% compatíveis com as *interfaces* padrões disponíveis na mesma. Como resultado, a NGN será construída em blocos podendo ser expandida ou reduzida em módulos de serviços. Este esquema de módulos de serviços se assemelha ao esquema de objetos utilizados num *software* criado com programação orientada por objetos. Deste modo, os módulos de serviços nas NGNs ou os objetos de um *software* são substituíveis e exercem funções determinadas dentro de um contexto: ambiente da rede ou de uma aplicação em *software*. As *interfaces* públicas e padronizadas pelas quais se define e representa funções dos serviços da rede são geralmente chamadas de *Application Programming Interfaces* (APIs), também conforme (INTERNATIONAL TELECOMMUNICATION UNION, 2004).

As APIs para serviços nas NGNs, como comentadas acima, são semelhantes àquelas encontradas em linguagens de programação, como em Java, no que diz respeito às suas definições de funções. Por exemplo, suponha que uma nova aplicação seja

construída para uma rede de telecomunicações e que o serviço SMS seja necessário. Neste caso, tal serviço já deve estar exposto na rede e acessível através de uma API adequada. Assim, a aplicação irá interagir com o serviço SMS, apropriadamente, porque já conhecerá a sua *interface*; ou seja, já saberá como invocar os métodos corretamente para trabalhar com suas capacidades. Se a *interface*, por exemplo, contiver o método

sendSMS(from: address, to: address, message: string),

a aplicação deverá executar o método chamado s*endSMS* fornecendo 3 parâmetros para ele. Os parâmetros devem ser fornecidos na mesma ordem como declarado pela assinatura do método. Ou seja, ao invocar o método para enviar mensagem com SMS, a aplicação deverá executar uma instrução computacional semelhante a:

sendSMS("(35)98312085", "(35)94719000", "Olá Inatel").

Portanto, uma API tem a utilidade de demonstrar quais funções de um dado serviço da rede estão disponíveis e como executar estas funções, apropriadamente, assim como em linguagens de programação. As APIs citadas nessa dissertação são, inicialmente, definidas através de alguma linguagem de modelagem de *interfaces* e dados para, depois, serem realmente implementadas com alguma linguagem de programação, na indústria de Telecomunicações. Conseqüentemente, essas APIs podem ser demonstradas nesse documento como notações formatadas em *Unified Modeling Language* (UML) e código Java ou com outra linguagem de definição de *interfaces*.

De fato, a questão chave aqui é a definição e publicação de um conjunto de *interfaces* entre desenvolvedores de serviços e aplicações clientes dos mesmos serviços. Quando existe um conjunto bem definido e conhecido de *interfaces* públicas, é possível criar serviços portáveis nas NGNs. Além disso, um serviço já existente numa rede, com *interface* pública, pode ser substituído por outro mais

novo, se o segundo tem (implementa) a mesma *interface*, porque, neste caso, nada será feito nos clientes de tal serviço. Assim, uma substituição de um serviço sem modificação de *interface* torna-se algo transparente para as aplicações clientes desse serviço. Isto significa que as *interfaces* são utilizadas para encapsular a implementação dos serviços, quando o desenvolvedor passa a ter liberdade para criar o serviço através de uma solução proprietária de *hardware, software*, ou outra que se fizer necessária. Como conclusão, as *interfaces* devem ser públicas. O mesmo não é obrigatório nas implementações encapsuladas por elas.

Em outro cenário, para uma aplicação de VoIP em uma NGN, o SIP será, provavelmente, a tecnologia preferida no processo de sinalização de estabelecimento das sessões de comunicações, como dito em (FALCARINI & LICCIARDI, 2003) e em (INTERNATIONAL TELECOMMUNICATION UNION, 2005a). Portanto, é desejável que também haja um conjunto de *interfaces* públicas definindo todas as funções desse protocolo, de forma que qualquer desenvolvedor forneça uma pilha SIP implementada com uma *interface* muito bem conhecida entre os serviços relacionados a tal protocolo. Como um exemplo de conjunto de *interfaces* para o SIP, há a API JAIN para SIP (JAIN-SIP-API) da empresa SUN, que pode ser acessada na *web* em (SUN MICROSYSTEMS, 2008a). Através dessa API é possível implementar a pilha SIP, definida na *Request for Comments* (RFC) 3261 da *Internet Engineering Task Force* (IETF), usando Java como linguagem de programação. Segundo a referência (FALCARINI & LICCIARDI, 2003), Java é mesmo uma linguagem de programação indicada para o desenvolvimento de serviços para as NGNs. De acordo com (INTERNATIONAL ENGINEERING CONSORTIUM, 2005), um ambiente de criação de aplicações baseadas em Java permite às operadoras aplicarem no mercado, rapidamente e com baixo custo, uma variedade de aplicações geradoras de lucro, todas derivadas de uma única fonte arquitetural. Quanto à API JAIN-SIP da SUN, ela está sendo implementada pelos esforços do Instituto Nacional de Padrões e Tecnologia dos EUA – *National Institute of Standards and Technology* (NIST) e o projeto respectivo se chama NIST-SIP, o qual pode ser visto em (NATIONAL INSTITUTE OF STANDARDS AND

TECHNOLOGIES, 2008). Então, se um serviço numa NGN necessitar interagir com a pilha SIP, isso poderá ser feito com o uso da pilha NIST-SIP. Este é um exemplo onde existe um protocolo útil, uma *interface* que o descreve e uma implementação da mesma *interface*, a qual poderá ser usada como servidora de requisições de serviços clientes a este protocolo. Nesse caso, quando a pilha SIP entrar em execução, serão os algoritmos da NIST-SIP que realmente estarão sendo executados. Mesmo assim, a aplicação cliente da pilha SIP só conhecerá a *interface* JAIN-SIP-API como uma descrição sobre as capacidades do protocolo SIP.

2.4 Serviços e Capacidades das NGNs

O NGN Focus Group do ITU-T adotou uma perspectiva baseada em *releases* para a liberação de especificações e relatórios técnicos sobre as NGNs. Na *Release 1*, que pode ser vista em (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), está presente um documento chamado Requisitos de Serviços, produzido pelo Grupo de Trabalho 1. O documento sobre Requisitos de Serviços dá uma descrição dos requisitos em linhas gerais e uma visão em alto nível das características funcionais das NGNs a serem consideradas.

A **Figura 1** foi obtida a partir da Release 1 de NGN do ITU-T e mostra a separação entre os serviços e as funções de transporte da rede.

O IP *Multimidia Service Component*, visto na **Figura 1**, é um componente de serviços cuja parte de transporte é baseada nas capacidades do *IP Multimedia Subsystem* (IMS) (LAURETTI, 2004). As funcionalidades do IMS para as NGNs, segundo a *Release 1*, empregam controle de serviço baseado em SIP. A arquitetura IMS proverá uma plataforma de acesso para uma gama de tecnologias tais como *Global System for Mobile communications* (GSM), *Wi-Fi*, *Cable* e xDSL, ou seja, elementos de rede, com tecnologia IMS, poderão acessar a rede IP, na camada de

transporte, via estas tecnologias. Contudo, esta dissertação não mostrará detalhes destas tecnologias, da arquitetura IMS e nem da utilidade do protocolo IP nas NGNs.

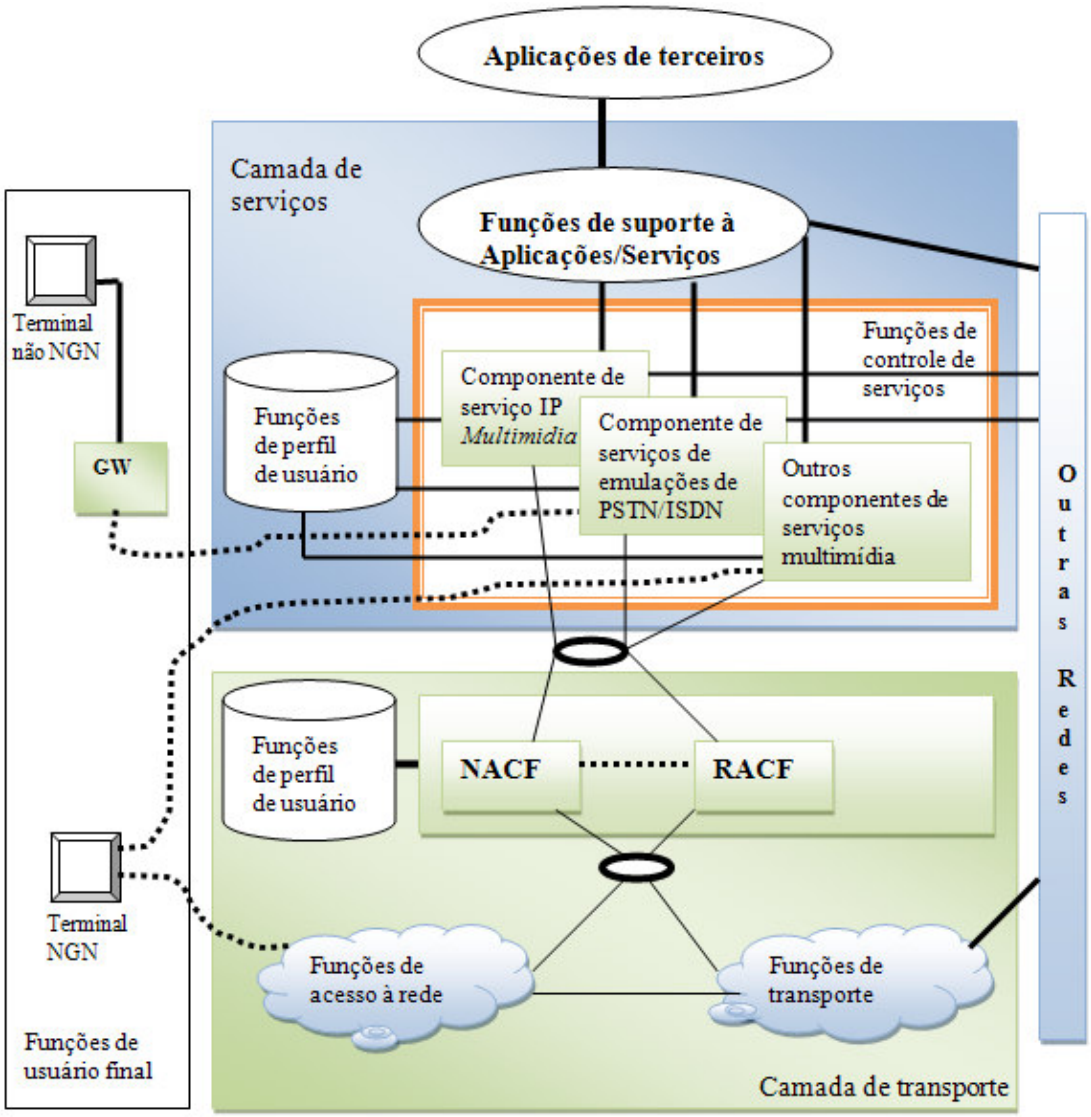

Figura 1 - *NGN com camadas distintas de serviços e transporte.*

Por outro lado, esta dissertação mostra o significado da camada de serviços das NGNs, desacoplada da rede, em termos de definição de aplicações, e como tal camada poderá beneficiar o aparecimento rápido de aplicações de valor agregado, nessas redes, quer sejam feitas por provedores, operadoras de telecomunicações ou desenvolvedores de aplicações.

O *PSTN/ISDN Emulation Service Component* é um componente planejado para possibilitar cenários onde as tecnologias *Public Switched Telephone Network /Integrated Services Digital Network* (PSTN/ISDN) devam ser substituídas. Este componente suportará um conjunto de *interfaces* de serviços legados nas PSTN/ISDN, que serão acessíveis pelos equipamentos de clientes das redes, de tal forma que o usuário poderá ter uma experiência, ao usar um serviço numa NGN, igual a usar um serviço legado de uma PSTN, por exemplo.

De acordo com (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), as NGNs realmente poderão ajudar na criação e oferta de novos serviços, porque adotarão a maneira de expor suas capacidades através das *interfaces* padronizadas, como explicado anteriormente neste documento. Isto proverá um método consistente de ganhar acesso às capacidades da rede e os desenvolvedores de aplicações poderão utilizar essas *interfaces* definidas ao programarem novas aplicações. Também, como já comentado antes, a indústria de telecomunicações está trabalhando para definir estas *interfaces* de serviços nas redes. Dentre estes trabalhos, surgem algumas classes de ambientes de serviços abertos. Exemplos destas classes, usando *interfaces* padronizadas da rede, incluem OSA/Parlay e Parlay X, conforme (INTERNATIONAL TELECOMMUNICATION UNION, 2004), (CARUGI, 2006) e (MOYER & UMAR, 2001). Segundo o que está explicado em (THE PARLAY GROUP, 2002a), os termos *Parlay*, *Open Service Access* e a sigla *Open Service Access* (OSA), existem por razões históricas. *Parlay* refere-se ao trabalho gerado pelo Grupo Parlay e 'OSA' refere-se ao trabalho produzido pelo *3rd Generation Partnership Project* (3GPP). Estas duas especificações de APIs foram harmonizadas e publicadas juntamente pelo Grupo Parlay, ETSI e 3GPP. Portanto, é comum ver a mesma API referenciada como Parlay/OSA, ou OSA/Parlay, ou OSA.

A **Figura 1** também mostra que deve haver funções capazes de fornecer dados de perfis de usuários e estes dados também devem ser gerenciados e estarem disponíveis para outras funções da NGN. Um perfil de usuário, numa NGN, é um conjunto de

informações relacionadas com o usuário, como seu endereço, nome completo, formas de ser contatado, etc. Então, uma gama de informações sobre os usuários deve ser gerenciada nas NGNs, se necessário. Este é um aspecto dinâmico de uma NGN, ou seja, o estado da rede pode depender de mudanças relacionadas com os seus usuários, a cada momento. Diferentemente de redes anteriores, as NGNs adapter-se-ão às condições de seus usuáros e à existência de novas aplicações executadas em locais variáveis.

Segundo (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), um ambiente com *interfaces* abertas e padronizadas para aplicações em rede deve facilitar as seguintes características das NGNs:

- Independência de Serviços

O ambiente de serviços abertos deve suportar os seguintes requisitos de serviços independentes:
- Independência dos fornecedores de redes: as funcionalidades, operações e o gerenciamento de aplicações de terceiros, bem como os serviços devem ser independentes da tecnologia dos provedores de rede e infra-estrutura.
- Independência dos fabricantes: em NGN deve ser suportado um ambiente de serviços abertos de vários fornecedores, provendo aos usuários uma ampla variedade de aplicações num ambiente competitivo.

- Transparência

O ambiente de serviços abertos deve suportar os seguintes requisitos de transparência:
- Transparência de localização: os serviços devem ser acessíveis de qualquer lugar, através de uma variedade de redes de acesso, independente da localização física atual dos servidores de capacidades relevantes aos serviços.
- Transparência de rede: um ambiente de serviços abertos deve permitir que os serviços sejam independentes de tecnologia e terminais.

- Transparência de protocolo: a transparência de protocolos deve ser alcançada através do uso de ferramentas de programação que utilizem *interfaces* de programação dos protocolos padronizados, o que poderá liberar o desenvolvedor dos detalhes da rede e, ao mesmo tempo, proteger as funcionalidades da rede que não poderão ser acessadas ou executadas indevidamente. Por exemplo, o protocolo SIP teve sua *interface* padronizada com a API JAIN-SIP. Essa API está sendo implementada constituindo a pilha do protocolo completa. Portanto, ao implementar um projeto que precise do protocolo SIP, um desenvolvedor precisará lidar apenas com as *interfaces* do projeto NIST-SIP. Assim, a manipulação das sessões estabelecidas na rede estará encapsulada no projeto NIST-SIP e, portanto, protegidas de ações indevidas por parte de aplicações de terceiros.

- Coordenação dos serviços

O ambiente de serviços abertos deve fornecer a capacidade de coordenar identidades, sessões e serviços. A coordenação dos serviços também deve ser suportada onde já existem mecanismos, por exemplo, em ambientes de serviços abertos, como OSA/Parlay e Parlay X citados por (CARUGI, 2006) e (MAGEDANZ, et al., 2004).

- Interação entre serviços e rede

A capacidade da NGN de propiciar um ambiente de serviços abertos deve permitir a interação entre serviços e entidades da rede para a criação e fornecimento dos mesmos.

- Suporte a desenvolvedores

Suporte aos desenvolvedores é uma questão chave na cadeia de entrega de novos serviços, nas NGNs. Custos mais baixos no ciclo de vida dos serviços e aplicações podem ser atingidos, automatizando-se o processo de produção dos mesmos. Além disso, formas para diagnosticar as aplicações, antes que elas sejam empregadas, podem ser criadas para aumentar a facilidade de gerência do processo de

desenvolvimento, isolando falhas. Nesse cenário, a NGN deve prover um ambiente eficiente de suporte ao desenvolvimento com as seguintes características:

- construção de novas aplicações;
- emprego das aplicações;
- remoção de aplicações;
- reuso e intercâmbio de componentes;
- suporte completo ao ciclo de vida de componentes que vai de instalação passando por configuração, administração, declaração pública, controle de versão, até manutenção e remoção;
- suporte a um ambiente consistente de *multi* vendedores e espaço de aplicações.

Para a cadeia de fornecimento de aplicações nas NGNs, os desenvolvedores terão um papel fundamental, como dito em (INTERNATIONAL TELECOMMUNICATION UNION, 2005b). Para que haja aplicações de valor agregado nas redes, deverá haver quem os faça, ou melhor, os desenvolvedores de *software* terão um papel importante no fornecimento das novas aplicações. Conseqüentemente, será viável facilitar o uso de capacidades das redes para os desenvolvedores, no momento da implementação dos novos sistemas. Esta facilidade pretendida pela *release 1* de NGN, do ITU-T é, justamente, a maneira de abrir as redes, através das *interfaces* padrões públicas, como já comentado anteriormente.

2.4.1 Serviços nas NGNs

Uma rede, que seja uma NGN, não precisa, obrigatoriamente, implementar todos os serviços e capacidades demonstrados na ilustração acima, ou todas as suas combinações. De fato, uma NGN pode ser constituída por um conjunto arbitrário dos serviços definidos naquela ilustração, bem como configurações e capacidades. Abaixo está listado um conjunto de exemplo de tipos de serviços suportados em uma NGN, definido pelo ITU-T na *release 1* (INTERNATIONAL TELECOMMUNICATION UNION, 2005b):

- Serviço de Voz, em tempo real e operável juntamente com a PSTN.
- Serviços de mensagem, tais como SMS, MMS e *Instant Messaging* (IM). IM, ou *instant messaging,* é um serviço pelo qual dois usuários podem dialogar, via texto, em tempo real, como através do *software MSN* da Microsoft.
- Serviços relacionados com presença e notificações gerais: Como explicado em (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), um serviço de Presença fornece acesso a informações sobre presença, que então ficam disponíveis para usuários ou aplicações na rede. Pela definição em (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), Presença é um conjunto de atributos caracterizando propriedades correntes de uma entidade, tais como: *status*, localização, etc. Uma entidade neste contexto é qualquer dispositivo, serviço ou aplicação que seja capaz de prover informações de Presença. Disponibilidade, por outro lado, denota habilidade e pretensão que uma entidade tem para se comunicar, baseadas em várias propriedades e políticas associadas com tal entidade. Por exemplo, capacidades da entidade, momento preferido para comunicação, etc. Os termos *presença* e *disponibilidade* são quase sempre usados juntos, para fornecerem um conjunto completo de informações sobre a presença. De acordo com (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), usuários de uma NGN deveriam ser capazes de fornecerem informações sobre suas condições de Presença e também de requererem informações sobre Presença de entidades na rede. Os usuários estariam, então, na condição de *presentities* e *watchers* (observados e observadores), respectivamente. No documento (INTERNATIONAL TELECOMMUNICATION UNION, 2005a) podem ser vistos alguns *slides* da empresa Lucent Technologies que demonstram uma aplicação útil baseada em capacidades de Presença.

2.4.2 Capacidades das NGNs

As NGNs promoverão novos meios de suportar o surgimento de uma gama de novas aplicações, incluindo aplicações com funcionalidades avançadas e complexas, de forma que tais sistemas não sejam invalidados ao serem portados em redes

diferentes. Baseado no conjunto de exemplos de serviços citados acima e na possibilidade de surgimento das aplicações com funcionalidades avançadas, as NGNs deverão prover capacidades apropriadas para suportar os mesmos. Dentre estas capacidades, que estão descritas no documento (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), algumas delas são acessadas ou usadas diretamente pelos serviços e aplicações de usuários, dentre elas:

Service Discovery, Service Registration e Presença.

Service Discovery

Service Discovery, ou descoberta de um serviço na rede, é o primeiro passo para determinar onde um dado serviço está disponível para a interação subseqüente. Portanto, as NGNs devem suportar capacidades de descobertas de serviços, o que permitirá que os usuários e seus dispositivos possam descobrir serviços na rede, aplicações e outras informações na rede ou recursos de seus interesses. A capacidade de descobrir serviços disponíveis na rede é essencial para possibilitar a mobilidade de seus usuários, por exemplo. Segundo (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), a mobilidade de um usuário na rede ocorre quando o mesmo pode mudar de terminal, de tal forma que seja possível se registrar em algum servidor, possibilitando a existência de uma associação entre o usuário, o terminal que ele usa e a localização de tal terminal na rede. Ou seja, independentemente da localização do usuário, um dado serviço disponível a ele terá condições de saber que terminal é usado, como cobrar pelo uso da rede/serviços e que usos de capacidades da rede são permitidos ao usuário. Para o usuário, em qualquer localização que ele esteja, deverá ser possível acessar um dado serviço, ou seja, o serviço deverá ser localizado e requisitado, mesmo que o usuário esteja num terminal diferente daquele onde ele usou o mesmo serviço pela primeira vez. Portanto, a descoberta de serviços permitirá ao usuário se registrar em algum servidor, independente de sua localização na rede, e o servidor tornará disponíveis somente serviços definidos para o perfil do usuário em questão. Neste caso, o usuário será identificado univocamente. Um exemplo de capacidade de *Service Discovery* é

implementado em *Web Services*. *Web Services* estão definidos e explicados mais adiante neste documento, na seção 3.3. Resumidamente, com *Web Services* é possível exportar serviços situados na rede para algum tipo de diretório e usar o protocolo *Universal Description, Discovery and Integration* (UDDI), também explicado mais adiante, na seção 3.3.3, para implementar *Service Discovery* em relação a esses serviços. Ou seja, usar UDDI para a localização dos serviços já presentes em algum diretório ou servidor na rede.

Service Registration

O ambiente de serviços abertos deve fornecer meios para gerenciar o registro de serviços. Portanto, deve haver uma forma de instalação, configuração, publicação e remoção de serviços, como um local apropriado para conter as principais informações de acessibilidade a estes serviços. As informações sobre modificações nos *status* dos serviços registrados, como atualizações ou expirações, devem ser distribuídas de alguma forma aos interessados. A capacidade de registrar serviços na rede, em diretórios de um ambiente aberto, possibilita que eles estejam acessíveis por aplicações ou outros serviços. De acordo com (CARUGI, 2006), isto significa que um serviço registrado pode ser localizado e acessado por outras entidades na rede. Em *Web Services,* por exemplo, novos serviços podem ser registrados num diretório, quando é desejável expor tais serviços. Neste caso, os serviços são registrados em *registries.* Um *registry* é um diretório que, além de apontar algum serviço para os usuários, também permite o registro prévio do mesmo.

Presença

Serviços de presença são suportados por três grupos de capacidades:

a) Capacidade de coletar dados de presença.

   A rede providencia uma forma de coletar dados que descrevem o *status* de conectividade do dispositivo usado pelo usuário. Esta capacidade torna-se útil, por exemplo, para determinar o estado de conectividade à rede de um

assinante de algum serviço determinado. Por outro lado, o próprio usuário também poderá prover informações, como sua disponibilidade para aceitar uma chamada, por exemplo.

b) Capacidade de distribuir dados de presença

Esta capacidade permite que um usuário da rede tenha as informações sobre o estado de outro usuário específico, como sua presença atual em algum terminal. Com esta capacidade, um serviço também poderá acessar as informações sobre o estado de um usuário específico.

c) Capacidade de gerenciar dados de presença

As NGNs devem fornecer um conjunto de capacidades para gerenciamento de Presença de usuários, conforme a privacidade e requisitos de regras de acesso. A capacidade de gerenciar Presença permite que um usuário requisite informações de presença de outro usuário e também exponha suas próprias informações. As informações a serem expostas são controladas de tal forma que quem as expõe possa decidir quando aceitará, ou não, as requisições destas informações, por parte de outros usuários.

Através de um serviço de Presença, um usuário, ou um dispositivo, ou uma aplicação poderá expor as informações sobre as suas presenças na rede, da forma como for mais conveniente ao usuário, ou seja, *status* de presença poderão ser declarados com valores diferentes, dependendo de quais outros usuários, dispositivos ou aplicações irão acessar as tais informações. O uso dos serviços de Presença fará com que grande parte dos serviços atuais, como mensagens instantâneas, SMS e MMS sejam enriquecidos com valor adicional. Ou seja, deixarão de ser serviços corriqueiros, porque poderão oferecer algo mais, relacionado com o controle de informações de presença dos usuários, dispositivos e aplicações. De fato e conforme (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), serviços como esses poderão obter dados de presença do usuário, como *status*, disponibilidade e preferências, acessando as informações sobre presença relacionadas com os

dispositivos e serviços do usuário, para iniciar algum tipo de comunicação. Abaixo estão descritos alguns exemplos de enriquecimentos desses serviços, como explicado em (INTERNATIONAL TELECOMMUNICATION UNION, 2005b):

- Se um usuário A, que precisa telefonar para um usuário B, vê que B está com *status* de ocupado, na sua lista de contatos, então esse usuário A pode decidir enviar uma mensagem de SMS para B, não sendo necessário discar o número de B para descobrir que o mesmo já está ocupado. Neste caso, uma informação de *status* de presença, para uma chamada, é obtida de um servidor de Presença.
- Um usuário A, através de sua lista de contatos, identifica que o seu amigo B está *on-line* em um jogo. Então, A decide entrar no jogo, através de *software* apropriado, e passa a convidar outros amigos a participar do mesmo jogo, por exemplo, enviando SMS para aqueles que estão com *status* de ocupado, solicitando-os a desligar suas chamadas correntes e a participar do jogo. Neste exemplo, um servidor de Presença pode interagir com o servidor da aplicação do jogo, para manter dados dos jogadores, como *status on-line*, disponibilidade para jogar, etc.

Gerenciamento de Localização

Gerenciamento de localização é uma capacidade que permite a obtenção da localização de uma aplicação ou usuário na rede. Com a localização do usuário ou dispositivo definida, uma aplicação poderá exercer funções específicas relevantes ao seu contexto de localização. Por exemplo, a localização do usuário poderá repercutir sobre o valor de uma cobrança devida pelo serviço utilizado.

Capacidade de Gerenciamento de Perfil de Usuário

Segundo (INTERNATIONAL TELECOMMUNICATION UNION, 2005b), um perfil de usuário é um conjunto de informações armazenadas relacionadas ao usuário (ou a um assinante de serviço). Num ambiente de NGN, o gerenciamento de informações de perfil de usuário é especialmente importante porque estas são requeridas para a implementação de um número de capacidades, incluindo autenticação do usuário, autorização para uso de algum serviço, assinatura de

serviço, mobilidade, localização e cobrança. O perfil de usuário inclui informações relacionadas com transporte, mídia, serviços e, comumente, inclui as seguintes informações:

- Identidade do usuário, que pode ser um nome ou número único na rede;
- Localização do usuário;
- Presença do usuário;
- Informações sobre quais aplicações o usuário é assinante;
- Preferências do usuário, como preferências por certas mídias;
- Informações pessoais, como dados de contato;
- Informações de conta como, por exemplo, o endereço onde um boleto de cobrança deve ser entregue.

Alguns dos requisitos gerais para o gerenciamento de perfil de usuário estão listados abaixo:

- incluir informação sobre recursos associados com o usuário e regras específicas dependentes do mesmo (ex: tipos de terminais que o usuário possui, capacidades dos terminais – *terminal capabilities* -, preferências dos usuários, etc.).
- notificar as entidades de controle da NGN sobre mudanças nos dados de perfil de algum usuário.
- possibilidade para recuperar informações geográficas de localização do usuário, num momento de chamada de emergência, por exemplo. A NGN deve providenciar meios para identificar a localização do usuário, de qualquer dispositivo de rede reconhecido. Evidentemente, porém, nem sempre será possível localizar um usuário. Contudo, a capacidade de localização deve estar, no mínimo, definida na NGN. Por exemplo, usando um telefone fixo, o endereço do usuário pode ser recuperado, através do número do telefone. Já, para telefones móveis, a informação geográfica pelo *Global Positioning System* (GPS) pode ser usada. Como visto em (SUN MICROSYSTEMS, 200-), alguns celulares já são equipados com módulos GPS *onboard* (Ex: Celular Nokia N95). Já, em redes locais sem fio, a posição do usuário estará associada ao ponto de acesso utilizado.

Capacidade de Gerenciamento de Perfil de Dispositivo

A NGN também deve gerenciar o perfil de equipamentos dos usuários. As informações do perfil do dispositivo do usuário podem incluir:
- Identificação do terminal ou dispositivo, como endereço ou nome.
- Informação geral do terminal, como número serial, modelo, tipo.
- Mídia suportada, ou seja, vídeo, texto, áudio.
- Atributos estáticos, como protocolos suportados, velocidade de transmissão, largura de banda e poder de processamento.
- Atributos sobre o que muda dinamicamente, como a modificação de uma aplicação que executa no terminal do usuário ou a localização do mesmo.
- Versão do *software* do sistema operacional.
- *Status* de *On/Off*.

2.5 Análise em Resumo

Nota-se que o mercado de telecomunicações está passando por uma mudança de bases tecnológicas e do comportamento de seus usuários. Tais mudanças culminarão na substituição de tecnologias, implicando no uso daquelas baseadas em redes IP e promovendo o sugimento de serviços de telecomunicações desejados atualmente pelas pessoas. Na verdade, o mercado de telecomunicações necessita inovar, ou seja, criar serviços inovadores, com o propósito de atrair mais clientes para as empresas operadoras nesta área. Caso contrário, ocorrerá estagnação do lucro ou mesmo falência para elas. A questão financeira relacionada à conquista de novos clientes, por meio de serviços inovadores, tornou-se tão relevante que órgãos de padronizações, como o ITU-T, estão definido padrões para a arquitetura das redes de telecomunicações de próxima geração e tipo de serviços necessários, sendo que um dos intuitos é possibilitar a existência de dispositivos, de diferentes utilidades (Ex: TV, computador, celular, fax, etc) conectados entre si sobre redes IP. Para estes dispositivos, deseja-se também a criação de aplicações de software com

funcionalidades que sejam muito úteis à vida das pessoas. E, evidentemente, tais aplicações dependerão das capacidades de telecomunicações suportadas por estes dispositivos e as redes das operadoras. Para facilitar o surgimento destas aplicações, que de alguma forma estarão usufruindo das redes de telecomunicações, faz-se necessária a definição das *interfaces* descritoras das capacidades das redes. Por estas *interfaces*, será possível criar acesso às redes das operadoras e definir, de forma segura, funções utilizáveis pelas aplicações de software. A padronização de tais *interfaces* será realizada por órgãos comentados no próximo capítulo e esta padronização proporcionará a criação de *gateways* de acesso às redes. Portanto, é necessário saber sobre a existência destas *interfaces* e sobre a possibilidade de implementá-las em entidades computacionais chamadas *gateways*, para o entendimento do que vem a seguir neste documento. Os próximos capítulos explanarão sobre estas *interfaces* para a abertura das redes de próxima geração e sobre os *gateways*. Por exemplo, será mostrado como prover um novo serviço numa rede de telecomunicações, baseado em capacidades de estabelecimento de ligações entre terminais, requisitadas por terceiros. E para prover este serviçso será necessário a implementação de uma *interface* específica e manter essa implementação num *gateway* específico. Portanto, até aqui está demonstrado o assunto das *interfaces* para a abertura das redes, que serão imprencindíveis para a compreenção sobre a vantagem da utilização dos *gateways* a serem explicados mais adiante.

3. JAIN, Parlay e Web Services

Como demonstrado, o sucesso das NGNs dependerá, dentre outros fatores, da abertura destas redes, ou seja, da exposição das suas capacidades para suportar aplicações de valor agregado a serem desenvolvidas pela comunidade de profissionais de TI. Segundo o artigo (GLITHO, 2004), a promessa chave das NGNs é, justamente, a habilidade para desenvolver e empregar aplicações de *software* lucrativas e inovadoras, rápida e eficientemente. Portanto, há trabalhos de grupos de especialistas em TI e em telecomunicações para a formulação das *interfaces* que exibirão as capacidades das redes. Exemplos de iniciativas que organizam tais grupos de especialistas são *Java APIs for Integrated Networks* (JAIN) (SUN MICROSYSTEMS, 2004) e Parlay.

3.1 JAIN

JAIN é uma iniciativa da comunidade de especialistas em linguagem de programação Java, com a finalidade de desenvolver APIs para a criação de serviços de telefonia (voz e dados). Segundo (JAVA APIs for integrated networks, 2007), o objetivo principal das APIs JAIN é representar a rede de forma abstrata, tanto que os serviços na rede sejam desenvolvidos de forma independente das tecnologias da mesma, quer seja a PSTN ou NGN. Portanto, o trabalho da iniciativa JAIN vem ao encontro à visão de criação de serviços portáveis nas redes abertas formulada com a NGN.

A iniciativa JAIN lida com as necessidades das NGNs desenvolvendo um conjunto de APIs definidas pela indústria de telecomunicações. Os membros da comunidade JAIN uniram forças para criar APIs abertas baseadas na plataforma Java da SUN, permitindo que os provedores de serviços criem e empreguem, rapidamente, novos sistemas, flexíveis e lucrativos. Estas APIs trazem portabilidade de serviços, independência da rede e desenvolvimento aberto para redes de dados, telefonia e comunicação *wireless*. Portanto, tais APIs também têm o objetivo de modificar o mercado das comunicações, reduzindo o uso de sistemas fechados de muitos proprietários e aumentando a utilização de ambientes abertos, como explicado em (SUN MICROSYSTEMS, 2004). Abrindo a rede para Java, as operadoras de rede podem estender os seus portfólios e tornar o desenvolvimento de aplicações de comunicações de próxima geração mais rápido, simples e barato. A remoção das barreiras proprietárias irá possibilitar a existência de um mercado aberto onde os fornecedores de equipamentos de redes, vendedores independentes de *software*, vendedores de pilhas de protocolos e provedores de serviços poderão comercializar uma variedade de componentes com a tecnologia Java.

Algumas APIs JAIN podem ser acessadas no *web site* (SUN MICROSYSTEMS, 2008b). Por exemplo, pode-se ver a API JAIN *Presence*, cuja *release* final é de 15 de março de 2006. Essa API define *interfaces* que permitem trabalhar com o serviço Presença e é direcionada para clientes deste serviço, isto é, para *watchers*. A API JAIN *Presence and Availability Management* (PAM), que é focada em gerenciamento de disponibilidade e presença, é direcionada às necessidades de Presença para um servidor na rede, ou seja, para a aplicação que implemente as capacidades de PAM no lado do servidor. Considerando desenvolvedores de *software*, a API JAIN *Presence* deve prover um *framework* padrão para o desenvolvimento e emprego de novas aplicações Java, de Presença, sem a necessidade de conhecimento prévio do protocolo suportando as capacidades de PAM. Exemplos de protocolos e aplicações que usam estes protocolos são *Wireless Village*, *Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions* (SIMPLE), *Extensible Messaging and Presence Protocol* (XMPP), *AOL*

Instant Messenger (AIM), etc. Posteriormente, uma aplicação Java que invoque métodos da JAIN *Presence* poderá ser empregada em qualquer implementação compatível com a JAIN *Presence*. Portanto, a JAIN *Presence,* quando implementada, deverá encapsular vários protocolos relacionados com presença na rede, de tal forma que isso resultará numa significante redução de esforços de desenvolvimento de novas aplicações e num crescimento da base de clientes, conforme (DEBBABI et al, 2006). O trabalho para a realização desta API foi baseado, no primeiro momento, em protocolos bem definidos sobre Presença criados por muitos consórcios, como o *Parlay* PAM *Workgroup*. De fato, boa parte das APIs do Parlay vem sendo codificada em Java pela iniciativa JAIN, como dito também em (JAVA APIs for integrated networks, 2007).

Alguém que acesse a JAIN *Presence* API pretendendo implementar um projeto com Presença, verá uma *interface* chamada *PresenceWatcher*, por exemplo. Esta é uma das várias *interfaces* definidas nesta API. A *PresenceWatcher* contém os métodos pelos quais uma aplicação obtém informações de presença de outra entidade na rede. O aprendizado desta API necessita o entendimento de todas as suas outras *interfaces* e isto pode ser alcançado, por exemplo, estudando a documentação da mesma, que está no formato de *Java Doc*.

Como visto, a iniciativa JAIN realiza um trabalho apropriado para a adoção de novas aplicações nas NGNs, porque propicia a implementação das *interfaces* necessárias por tal adoção. Por estas *interfaces*, os protocolos que funcionam na rede estarão encapsulados e os serviços suportados por tais protocolos "verão" somente as *interfaces*. Assim, os serviços expondo as *interfaces* destas APIs poderão ser portados em diferentes redes, porque serão compatíveis com as aplicações clientes, desenvolvidas por terceiros. Da mesma forma, tais aplicações serão portáveis nas redes onde os serviços estiverem acessíveis através destas *interfaces* padronizadas. Portanto, a iniciativa JAIN já exerce um papel importante na definição das NGNs, como também pode ser visto na implementação das APIs Parlay, tudo isso focando o

desenvolvimento com a linguagem Java. Segundo (SUN MICROSYSTEMS, 2004), Java pode estar em todos os lugares no domínio das comunicações. Por exemplo, há APIs Java para *mobile handsets*, para *web servers* e sistemas de bilhetagem. Além disso, há também APIs Java proprietárias que completam o cenário Java nas telecomunicações.

Empresas como Ericsson, Siemens, Fujitsu, NEC, Huawei e Lucent estão colocando a tecnologia JAIN em seus produtos, como pode ser visto em (SUN MICROSYSTEMS, 200-). Na indústria já existem produtos que foram implementados com a tecnologia JAIN e já são certificados neste campo. Alguns produtos certificados com a tecnologia JAIN são ferramentas de desenvolvimento.

Exemplo de produtos certificados:

- JAIN Call Control 1.0a. da empresa TrueTel,
- SIP Servlets 1.0 da empresa Siemens,
- Jain SIP 1.1 do NIST,
- JAIN TCAP 1.1 da HP.

Exemplos de ferramentas de desenvolvimento certificadas:

Jain SIP 1.1 do NIST;
JAIN *Java Call Control* (JCC) da NTT *Network Service Systems Laboratories*.

3.2 Parlay

Conforme a própria definição encontrada em (THE PARLAY GROUP, 200-?), o grupo Parlay é um consórcio formado com o objetivo de propiciar o desenvolvimento das *interfaces* de programação de aplicações (APIs), que podem habilitar a implementação de outras aplicações operantes, através das redes convergentes. O trabalho do Parlay provê a integração entre tecnologias de telecomunicações e de TI, necessária ao sucesso das NGNs, como comentado anteriormente. Mais precisamente, as APIs do Parlay são como um contrato entre telecomunicações e TI, o que possibilita o desenvolvimento de novas aplicações de valor agregado para a indústria de telecomunicações, através do trabalho de profissionais de TI. Esta integração entre telecomunicações e TI é feita de forma segura e mensurável para cobranças por usos de serviços das redes.

Desde 2002 os trabalhos do Parlay envolvem entidades como o 3GPP e o ETSI. Estas entidades conseguiram alinhavar as suas *releases* a um ponto que tornaram possível a existência de um único conjunto de especificações de APIs para os desenvolvedores de aplicações, chamada Parlay/OSA, ou OSA/Parlay, ou Parlay, de acordo com (GUPTA, 2002). Como dito em (GLITHO, 2004), a API Parlay foi selecionada pelo 3GPP como a base para o seu principal *framework* padronizado de desenvolvimento de aplicações, o OSA.

As novas especificações Parlay/OSA oferecem características destacadas para segurança, controle de chamada, cobranças baseadas em conteúdo, capacidades de terminais e gerenciamento de presença e disponibilidade.

O conjunto Parlay/OSA pode ser considerado como APIs padrões designadas para prover ao desenvolvedor de aplicações uma visão em alto nível das capacidades

(funções) da rede de telecomunicações. Além disso, permite melhorias de seguração porque dificulta a construção de código que cause algum dano (ex: usar uma cota de serviço, como bytes transmitidos com o *File Transfer Protocol* (FTP), além do limite estabelecido pela operadora) à rede, já que, desta forma, o desenvolvedor vê e acessa somente o que está nas *interfaces*. Ou seja, as *interfaces* declaram um conjunto do que pode ser acessado e utilizado na rede e as aplicações clientes têm que aceitar tal conjunto de possibilidades, caso sejam aplicações criadas para funcionar em qualquer NGN com *interfaces* padronizadas para suas capacidades.

Através de um subconjunto de APIs Parlay uma operadora de telecomunicações pode especificar e declarar quais capacidades da sua rede estão realmente liberadas para acesso e uso. Assim, caso um desenvolvedor deseje criar uma aplicação a ser portada na rede de tal operadora, este mesmo desenvolvedor terá que conhecer as APIs em questão. Isto é, terá que saber quais funções podem ser invocadas na rede. Este conhecimento pode ser obtido com as funções declaradas nas *interfaces* da API utilizada pela operadora. Por exemplo, se uma nova aplicação com funções de SMS será criada, então as *interfaces* relacionadas com SMS, da API Parlay, deverão ser conhecidas pela aplicação. Assim, a aplicação terá condições de invocar métodos que trabalham com SMS, na rede da operadora, através das *interfaces* apropriadas. Estudar as especificações das APIs Parlay é uma forma para um desenvolvedor de *software* se preparar para a criação de aplicações para as NGNs. Geralmente, porém, a operadora é quem decidirá que funções dessas *interfaces* realmente estarão disponíveis e restrições também poderão ser criadas na implementação dessas *interfaces*. Por exemplo, pode-se criar restrição no número de pessoas participando numa chamada em conferência, com um limite menor que aquele definido na *interface* usada.

Para se usar uma *interface*, ou seja, para interagir com os métodos da mesma, é necessária a sua implementação. Isto significa que toda API do Parlay deve ser implementada, antes que possa ser utilizada. De fato, as especificações do Parlay são

compostas de documentos textos e modelagens de *interfaces* em alto nível, utilizando UML. Isto implica que o Parlay não disponibiliza algo já pronto para ser executado numa rede de telecomunicações. Mais detalhadamente, as seguintes ações são necessárias até que um serviço de rede esteja operante e encapsulado por uma *interface*:

a) Estudo e compreensão das *interfaces* das APIs Parlay relacionadas com o serviço;
b) Codificação dessas *interfaces* em alguma linguagem de programação, como Java. Isso implica em codificar as descrições em UML;
c) Implementação (realização) das *interfaces* com a mesma linguagem de programação. Isto implica em definir os algoritmos para os métodos declarados nas mesmas. O resultado é a complementação do código iniciado em (b). Em Java isso requer a definição (construção) de classes[3];
d) Liberação e execução, na rede, da implementação da interface.

Portanto, se uma operadora desejar abrir as capacidades de Presença da sua rede por exemplo, será necessário que os serviços relacionados com Presença estejam implementados, encapsulados e funcionais, através das *interfaces* de Presença do Parlay. Assim, esta rede poderá portar aplicações clientes de valor agregado que demandam serviços com capacidades de Presença. Algumas APIs Parlay já estão sendo codificadas em Java, pela iniciativa JAIN.

A implementação e disponibilidade de serviços na rede, através das *interfaces* Parlay, requerem a inclusão de *gateways* na rede, como explicado em (VENTERS, 2004). Estes *gateways* podem ser vistos como os *containers* dos serviços implementados e encapsulados pelas APIs. Ou seja, para acessar as capacidades da rede, é necessário

---

[3] Uma classe é um módulo de código fonte, que, quando em execução no computador, define um objeto no contexto do respectivo software. Um objeto é uma entidade abstrata, mas com atributos e métodos para acesso ou alteração de tais atributos. Uma classe, em seu algoritmo, define a lógica dos métodos da interface que ela implementa.

acessar um *gateway*, porque será através deste elemento de rede que uma aplicação cliente realmente terá condições de alcançar e usufruir de tais capacidades. Este tipo de *gateway* é chamado de *Gateway* Parlay.

Um *Gateway* Parlay não somente esconde os detalhes da rede, como protocolos de transporte e sessão, mas também evita que alguma aplicação de terceiro cause algum dano no domínio da rede. As operadoras de telecomunicações serão, geralmente, as responsáveis por instalar os *gateways* pelo mundo e, portanto, estarão muito interessadas na forma como eles serão implementados e, conseqüentemente, no tipo de API acessível através dos mesmos. O resultado disto é que as operadoras de telecomunicações também estão interessadas em participar do trabalho do Parlay. Participar do trabalho do Parlay implica em participar da definição das partes das redes que estarão abertas nas NGNs. Se a indústria de telecomunicações pode definir o que fica e o que não fica aberto nas redes e, se tal abertura é controlada nos *gateways*, então isso significa que os donos dos *gateways* terão o controle desejado sobre suas redes. Isto possibilitará às empresas aceitarem ou evitarem aplicações de terceiros, no domínio de suas redes, o que implicará em filtrar o que for mais rentável ao mercado do setor de telecomunicações. Então, os *gateways* também podem ser vistos como um tipo de *firewall* entre o domínio seguro da rede e o domínio não seguro, como citado em (THE PARLAY GROUP & BRITISH TELECOMMUNICATIONS, 2005). Conclusão: a abertura das redes implica na inclusão de *gateways* proprietários.

Tendo isto em vista, as seguintes ações são necessárias até que uma aplicação de valor agregado esteja pronta para atuar na rede de uma operadora de telecomunicações e usar alguma capacidade desta rede, pertencente a um serviço:

a) Estudo e compreensão da *interface* Parlay relacionada com o serviço;
b) Implementação da aplicação cliente que poderá interagir com o serviço encapsulado com esta *interface*;

c) Obtenção de acesso ao *gateway* da operadora de telecomunicações, que deverá expor a *interface* em questão;

d) Emprego da aplicação cliente na rede.

Portanto, uma aplicação cliente, que precise explorar as capacidades da rede, deverá interagir com um *gateway* e não terá acesso às capacidades que não estiverem abertas através do mesmo.

A **Figura 2** representa um *Gateway* Parlay e foi retirada da apresentação (MAGEDANZ, 2004). Este tipo de *gateway* consiste de 2 partes principais:

- O *Framework*.

- O Serviço.

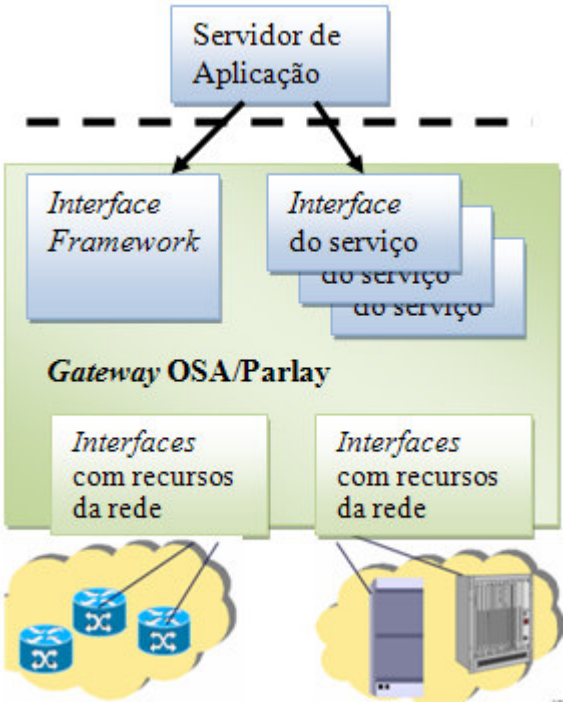

Figura 2 - *Gateway OSA/Parlay.*

A linha pontilhada no desenho acima representa a API Parlay/OSA e também a divisão entre o domínio seguro da rede e o domínio de aplicações de terceiros. Neste caso, qualquer aplicação hospedada no Servidor de Aplicação interage com esta API, ao interagir com o *gateway*. Ou seja, a visão que a aplicação tem do *gateway* é uma visão de *interfaces* desta API e, portanto, a única forma de utilizá-lo é invocando

métodos das mesmas. A primeira *interface* que deve ser utilizada pela aplicação, antes de acessar as *interfaces* das capacidades da rede, é a *Framework*. A implementação da *interface Framework* fornece a habilidade para a operadora de rede negociar com o fornecedor de aplicação (aplicação cliente). É um ponto inicial de contato para a aplicação cliente descobrir quais são as capacidades oferecidas pela rede e que podem ser utilizadas por ela. Este ponto também consiste em precauções de segurança requeridas e define qualquer *Service Level Agreement* (SLA). Através de um SLA, podem ser especificadas quais capacidades de serviços realmente estarão disponíveis para uma aplicação. Ou seja, atualmente há a prática de formular um acordo entre uma operadora de rede e um provedor ou desenvolvedor de aplicações, para que fique bem definido o que é obrigação da operadora, como fornecedora de capacidades de telecomunicações, e o que deve ser cumprido pela aplicação cliente. Assim, somente serviços já acordados no SLA estarão acessíveis, quando a aplicação obtiver acesso via o *Framework*. Como um exemplo simples, retirado de (ODADZIC & JANKOVIC, 2003), a característica Controle de Chamadas – *Call Control* (CC) -, dentre os serviços de uma rede de telecomunicações, pode ser comentada: se algum provedor de aplicações assina um SLA (documento) para uso máximo de 5 usuários conectados numa chamada (*conference call*), a aplicação cliente correspondente irá obter, na fase de descoberta e seleção de serviço, no caso da API Parlay via o *Framework*, uma instância de CC restrita a controlar 5 usuários conectados, no máximo, mesmo se a rede subjacente puder lidar com chamadas tendo mais usuários conectados.

É através do *Framework* que uma aplicação de terceiro poderá ser identificada, autenticada e liberada para interagir com as outras *interfaces* disponíveis no *gateway*, na parte de serviços. Portanto, o *Framework* controla como uma aplicação de terceiro acessa a rede.

A parte de serviços consiste em um número de *interfaces* individuais que são designadas para permitir às aplicações de terceiros acessarem as capacidades da rede.

Por exemplo: mobilidade, gerenciamento, controle de chamada, presença e disponibilidade. Portanto, caso uma aplicação de valor agregado necessite usar capacidades da rede relacionadas com Presença, ou SMS, por exemplo, será necessária a utilização de um *gateway* contendo meios de acessar tais serviços e, então, interagir com as implementações das *interfaces* dos serviços necessários.

O *gateway* pode interagir com outros elementos da rede, como *switches,* para prover capacidades individuais de serviços, tais como controle de chamadas ou localização. Para tal, a própria implementação do *gateway* pode utilizar acesso às *interfaces* proprietárias de elementos da rede ou de protocolos. Por exemplo, um *Gateway* Parlay que necessite do serviço de estabelecimento de sessão via SIP, poderá interagir com alguma pilha funcional deste protocolo. Se o *gateway* é construído para interagir com pilhas encapsuladas com a JAIN-SIP-API, então, a pilha NIST-SIP se torna válida para servir tal *gateway*. Felizmente, para um desenvolvedor de aplicação de valor agregado para as NGNs, profissional de TI, este nível de detalhamento fica todo encapsulado pela API do Parlay, portanto não sendo necessário o conhecimento do mesmo. Assim, todo o trabalho de pesquisa realizado nesta dissertação também foi feito do ponto de vista de um profissional de TI, o que desobriga a apresentação de detalhes de tecnologias de redes, neste documento. De fato, este trabalho pode existir independente das tecnologias de redes, confirmando que as mesmas não precisam ser conhecidas ao se planejar a criação de aplicações para as NGNs. Este é um dos propósitos do Parlay.

3.2.1 Exemplos da API OSA/Parlay

Toda a especificação da API OSA/Parlay, versão 5.1, está dividida em 15 documentos, os quais podem ser obtidos no próprio *web site* do Parlay, em (THE PARLAY GROUP, 200-?). Esta versão 5.1 é a última versão que foi concluída. Uma nova versão 6.0 está em andamento, ainda considerada rascunho, a qual conterá novas *interfaces* acrescidas à versão anterior 5.1, sendo que a sua última alteração

ocorreu em abril de 2007, considerando o momento em que esta dissertação foi escrita. Na API OSA/Parlay 5.1 são encontradas várias *interfaces* que permitem acesso às funções nas redes de telecomunicações, organizadas em grupos, tais como: controle de chamada, gerenciamento de contas, interação de usuário, cobrança, *terminal capabilities*, *Multi-Media Messaging*, *Generic Messaging*, *Data Session Control*, presença e gerenciamento de disponibilidade, mobilidade, dentre outros. Portanto, estas *interfaces* estão divididas em grupos relacionados com estas funções. Cada documento representa um grupo de *interfaces* e está no formato *Portable Document Format* (PDF). As *interfaces* para a construção de um *gateway* OSA/Parlay estão divididas em *interfaces* de serviços e a *interface* do Framework, como visto na **Figura 2**. Abaixo, seguem breves resumos explicativos do significado de alguns destes grupos citados de *interfaces* da API OSA/Parlay, como pode ser visto nos respectivos documentos:

3.2.1.1 Framework

Dentre as APIs do Parlay, existe uma chamada *Framework*. A implementação desta API provê uma entidade no *gateway* Parlay e está representada na **Figura 3**. Esta entidade, que também pode ser chamada de *Framework*, provê a habilidade da operadora de rede negociar com o provedor de aplicação, dentre outras possibilidades. Na verdade, o *Framework* é formado por um conjunto de *interfaces*, sendo que há a *interface* apropriada para a comunicação entre ele e o provedor de aplicação. Esta e as outras *interfaces* estão comentadas mais adiante, ainda nessa Seção 3.2.1.1. Por exemplo, se o provedor de aplicação deve negociar com a operadora de rede, talvez para estabelecer de quais serviços da rede ele terá permissão de uso, isso será fixado através do *Framewok*. Isto significa que, se alguma aplicação hospedada no provedor, cliente dos serviços do *gateway*, necessita, por exemplo, do serviço MMS, então, antes de efetivamente usá-lo, será necessário um primeiro contato com o *framework*. Neste caso, o framework, já configurado apropriadamente segundo as negociações entre a operadora de rede e o provedor de aplicação, irá permitir ou barrar o acesso ao serviço demandado. Irá barrar, se tal

serviço não fizer parte do acordo de prestação de serviços da rede para a provedora de aplicações. Então, o *Framework* serve para garantir as questões negociadas entre a operadora de rede e seus clientes, além de ter outras funções explicadas a seguir.

A API do *Framework* está documentada no padrão (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007a), o qual foi atualizado em janeiro de 2007, na versão Parlay 5.1.

A **Figura 3** foi retirada do documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007a) e mostra que o *Framework* contém *interface* com a operadora, com a aplicação cliente e com as capacidades da rede.

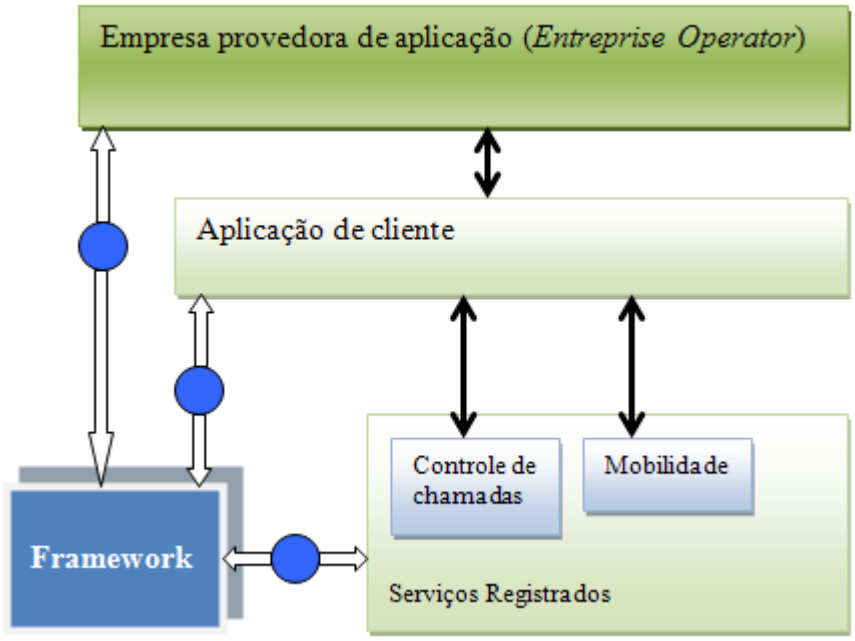

Figura 3 - *Interfaces do Framework do gateway OSA/Parlay.*

Estas *interfaces* estão representadas pelas 3 bolas na figura. Através da *interface* entre o *Framework* e a aplicação cliente ocorrem: autenticação (a aplicação cliente deve, primeiramente, se autenticar junto ao *gateway*, se tal ação faz parte de um acordo de serviço pré-estabelecido, de forma *off-line*, entre o cliente da aplicação e o

fornecedor da mesma, por exemplo); autorização (define quais serviços podem ser usados por uma aplicação já autenticada); descoberta de quais capacidades da rede estão disponíveis via *gateway*; estabelecimento de um acordo de serviço e acesso às capacidades da rede. Através da *interface* entre o *Framework* e a operadora (empresa provedora de serviços) ocorre uma função de inscrição em serviço. Ou seja, a operadora pode se inscrever como assinante ou cliente de certos serviços, o que reflete, então, um acordo, como por escrito, entre a operadora e serviços abertos na rede. Pela terceira interface, pode-se registrar no *Framework* quais são os serviços disponíveis na rede e reconhecidos pelo *gateway*. Assim, uma aplicação cliente pode saber, através do *Framework*, quais são os serviços disponíveis. O documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007a) contém a especificação de todas as *interfaces* da API *Framework* e explicações de como utilizá-la. Isto tudo está documentado com diagramas de seqüência, diagramas de classe e de estado da UML.

A operadora de um *Framework* é uma empresa de telecomunicações, como a AT&T, *British Telecom* (BT), etc. Uma *Enterprise Operator* é uma empresa que fornece algum serviço que usa capacidades da operadora de telecomunicações, como uma instituição financeira. Por exemplo, banco, empresa de seguro, etc. Este tipo de empresa contém a aplicação cliente, que é utilizada pelos usuários/consumidores e que interagem com os serviços da rede. Portanto, uma empresa que queira fornecer algum serviço que use telecomunicações poderá contratar desenvolvedores de TI, conhecedores das *interfaces* Parlay, para implementarem as aplicações necessárias. Enquanto isso, as operadoras de telecomunicações permanecem responsáveis pela implementação dos *gateways,* com as *interfaces* Parlay, que responderão corretamente às requisições das aplicações clientes.

3.2.1.2 Algumas *interfaces* de serviços

- **Controle de chamada** (*Call Control*) :

Provê controle de chamadas, como *Multi-Party Call Control, Multi-Media Call Control* e *Conference Call Control*.

❖ **Serviço genérico de mensagens** (*Generic Messaging*):

Grupo de *interfaces* que definem serviços capazes de tratar caixas de mensagens (*mail boxes*) como, ler mensagens, remover e colocar mensagens em tais caixas. A especificação das *interfaces* para estes serviços encontra-se no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007) e contém, por exemplo, diagramas de seqüência que mostram como abrir e fechar um *MailBox* e como obter uma mensagem do mesmo. Como os serviços de mensagens e caixas de mensagens já são conhecidos atualmente pelos usuários, em geral, de recursos de telecomunicações, as *interfaces* deste grupo podem ser convenientemente demonstradas e comentadas aqui. Assim, esta será a primeira exemplificação de *interfaces* Parlay, no documento corrente. Isso ajudará no entendimento de como as APIs são especificadas teoricamente.

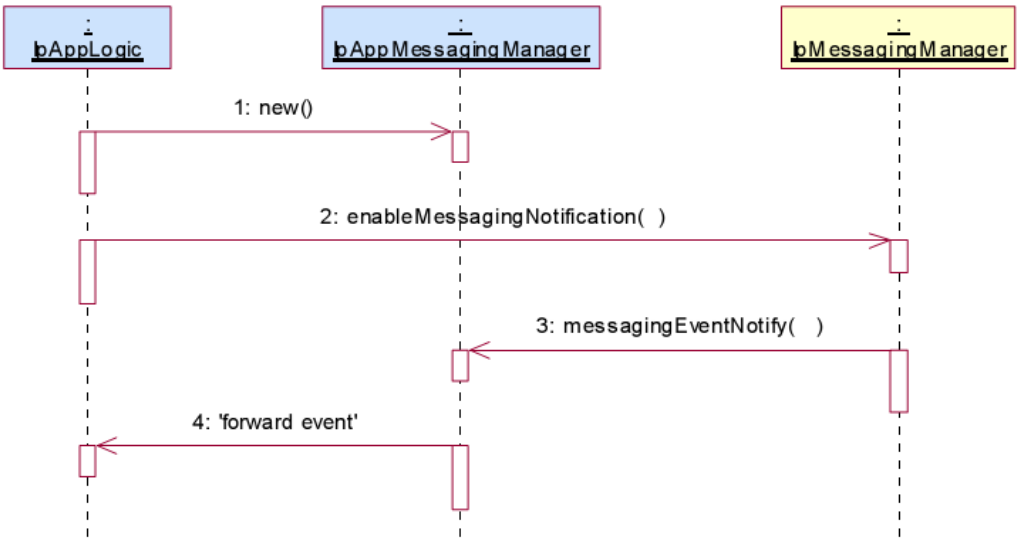

Figura 4 -*Diagrama de seqüência Prepare Mailbox.*

A **Figura 4** mostra um diagrama de seqüência em UML visto no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b). Este diagrama descreve como preparar uma

caixa de mensagens para receber uma nova mensagem. As *interfaces* representadas na **Figura 4** – *interfaces* cujo nome tem o prefixo *IpApp* (*IpAppLogic* e *IpAppMessagingManager*) -- devem ser implementadas pela aplicação cliente, ou seja, a implementação destas *interfaces* é de domínio da aplicação de valor agregado que fará uso das capacidades da rede relacionadas com o serviço genérico de mensagens. Já a *interface IpMessagingManager* é implementada e disponível através do *gateway*. Como descrito na **Figura 4**, um objeto *IpAppLogic* -- objeto que implementa esta *interface* -- deve criar um objeto *IpAppMessagingManager*. Isto está representado pela instrução 1, acima (*new()*). Em seguida, o objeto *IpAppLogic* invoca o método *enableMessagingNotification()* do objeto *IpMessagingManager*, como uma segunda instrução. Esta instrução 2 implica numa interação entre a aplicação e o *gateway* e é, através dela, que o objeto *IpAppLogic* "liga" no objeto *IpMessagingManager* a função de notificações sobre novas mensagens. Ou seja, quando há novas mensagens a serem recebidas, o objeto *IpMessagingManager* toma alguma providência para avisar o objeto *IpAppLogic* sobre isto. Os detalhes da implementação do método *enableMessagingNotification()* não serão conhecidos pela aplicação cliente, já que tal implementação é de domínio da operadora de telecomunicações ou de quem seja o desenvolvedor do *gateway* sendo utilizado neste caso. Quando uma nova mensagem é recebida pela rede, o objeto *IpMessagingManager* invoca o método *messagingEventNotify()* no objeto *IpAppMessagingManager*. Esta instrução 3 é a forma como o objeto *IpMessagingManager* avisa a aplicação sobre a existência de nova mensagem. O objeto *IpMessagingManager* também não precisa saber sobre os detalhes de implementação do método *messagingEventNotify()*. Tal implementação é construída conforme necessidades da aplicação cliente. Cada objeto cliente precisa apenas saber como invocar uma função em outro objeto servidor. No exemplo acima, o objeto *IpMessagingManager* necessita de uma referência ao objeto *IpAppMessagingManager* e essa referência, provavelmente, deve ser passada como um parâmetro do método *enableMessagingNotification()*. Este tipo de questão também está especificado, detalhadamente, na API, neste caso,

no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b).

Os diagramas de seqüências seguintes descrevem as outras *interfaces* da API do serviço genérico de mensagens, quais são os métodos disponíveis em cada uma e a ordem correta para invocá-los:

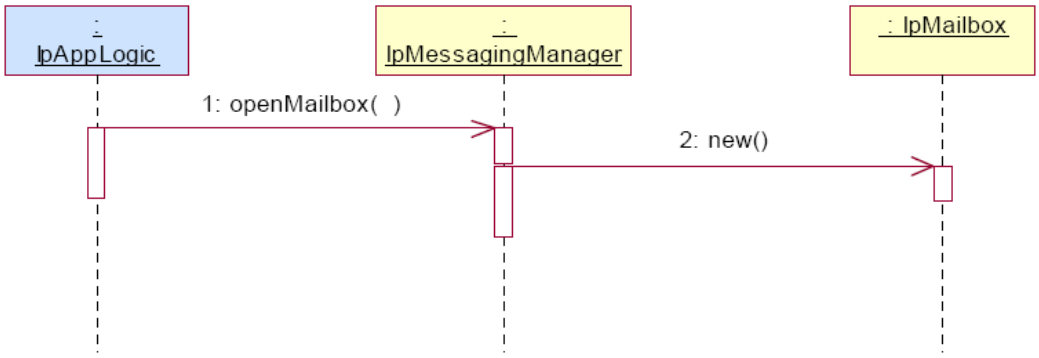

Figura 5 - *Diagrama de seqüência Open Mailbox.*

A **Figura 5**, vista em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b) , mostra como a aplicação cliente abre uma caixa de mensagens. A aplicação cliente requisita ao objeto implementando a *interface IpMessagingManager* para que ele crie um objeto implementando a *interface IpMailbox*. De acordo com o diagrama anterior, é necessário, então, conhecer também a *interface IpMailbox*, a qual deve estar implementada por um objeto, o que é responsabilidade do lado do *gateway*. O diagrama da **Figura 6** exemplifica outros métodos disponíveis no objeto *IpMailbox*, que podem ser invocados pela aplicação cliente, quando necessário.

A **Figura 6**, vista em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b), mostra como a aplicação cliente solicita acesso a uma mensagem. Duas novas *interfaces,* descritas acima, também devem ser do conhecimento da aplicação cliente, para a

obtenção da mensagem: *IpMailboxFolder* e *IpMessage*. O método *openFolder()* do objeto implementando a *interface IpMailbox* retorna uma referência para um objeto implementando a *interface IpMailboxFolder*.

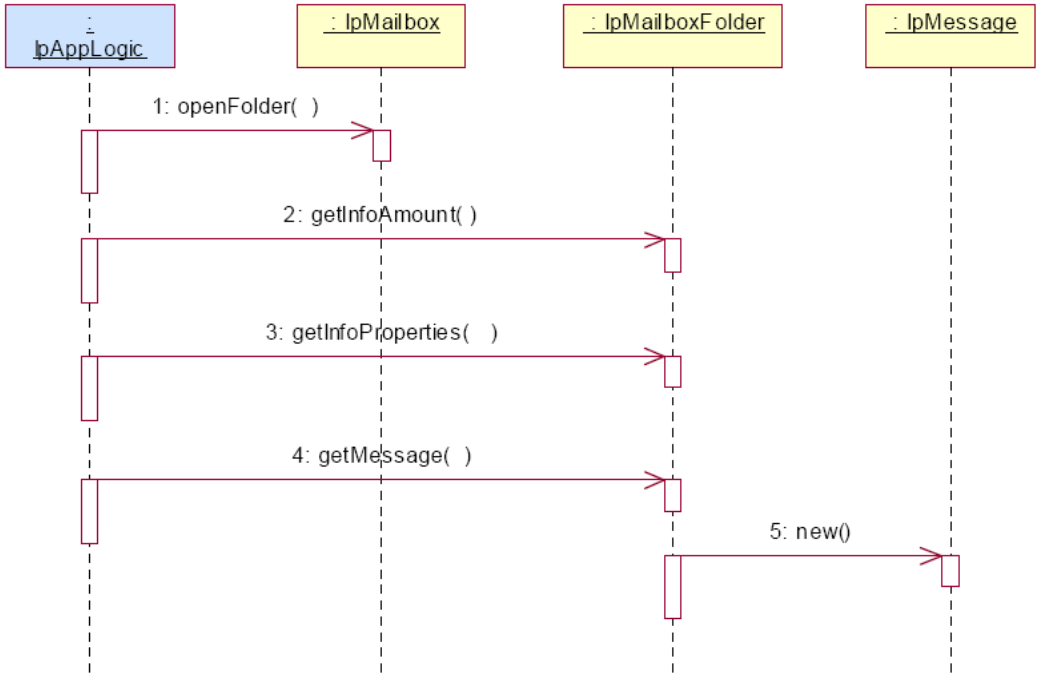

Figura 6 - *Diagrama de seqüência Get Message.*

Com tal referência, a aplicação pode, por exemplo, invocar métodos no objeto *IpMailboxFolder*. De acordo com o diagrama anterior, o método *getMessage()* implica a criação de um objeto implementando a *interface IpMessage*. Provavelmente, em seguida, o objeto *IpAppLogic* recebe uma referência à mensagem encapsulada no objeto *IpMessage*. Este último detalhe não está descrito em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b).

A **Figura 7**, vista também em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b), mostra as *interfaces* que devem ser conhecidas pela aplicação cliente, quando se deve utilizar o serviço genérico de mensagens.

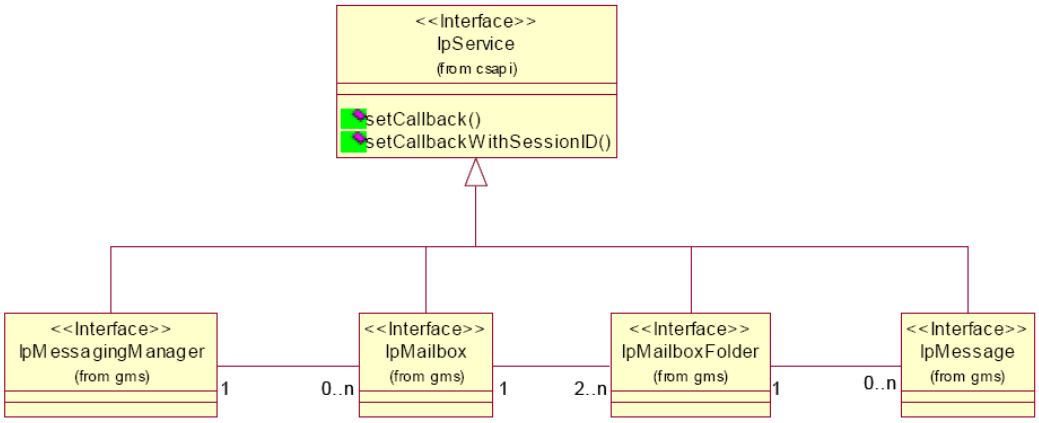

Figura 7 - *Diagrama de classes Get Message.*

Mostra também que há uma *interface*, chamada *IpService*, da qual as outras derivam. Os documentos do Parlay, além de mostrarem as *interfaces* relacionadas com cada serviço, mostram também os métodos disponíveis em cada uma, como na **Figura 8**:

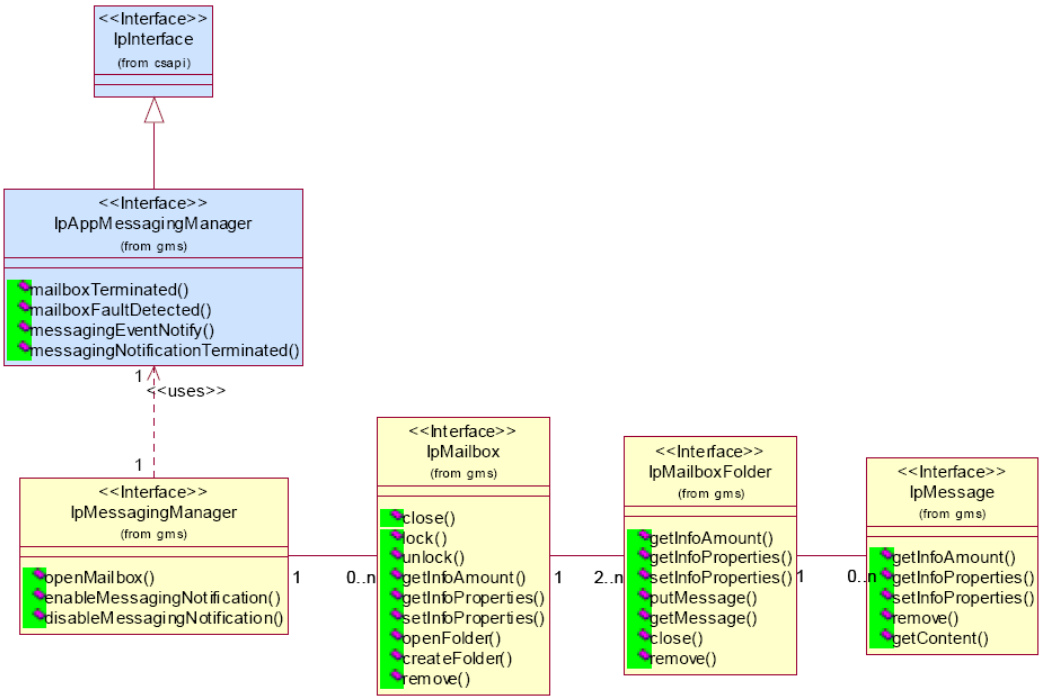

Figura 8 – *Diagrama de classes Application and Service Interfaces for messaging.*

A **Figura 8**, encontrada também em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b), mostra que a *interface IpAppMessagingManager*, que deve ser

implementada pela aplicação cliente, deriva da *interface IpInterface*. Os métodos da *interface IpAppMessagingManager* são os métodos que podem ser invocados por um objeto do domínio do serviço em uso. Por exemplo, se houver alguma falha na manipulação de mensagens na rede, um objeto implementando a *interface IpMessagingManager* poderá invocar o método *mailboxFaultDetected*() do objeto implementando a *interface IpAppMessagingManager*. Desta forma o serviço ofertado pela rede tem condições de passar mensagens à aplicação cliente, sempre que necessário. Pode ser visto, então, a importância de fazer com que a aplicação cliente implemente os métodos que podem ser chamados pelo serviço. Ou seja, a aplicação servidora, ou serviço na rede, também pode invocar funções na aplicação cliente, como retorno de requisições feitas ao serviço, o que é um esquema chamado de ***callback,*** neste caso. Assim, quando o método *mailboxFaultDetected*() é invocado, um algoritmo adequado à aplicação cliente será executado, ou seja, a aplicação cliente define o que deve ser feito, se este método for chamado numa *callback*. A metodologia de definir sistemas usando *interfaces*, classes e métodos planejados para *callbacks* está presente na linguagem Java. E este é mais um motivo que torna esta linguagem de programação adequada para a implementação da API Parlay. No diagrama acima pode ser vista também a multiplicidade das associações entre as *interfaces*. Por exemplo, um *IpMailbox* deve estar associado, no mínimo, a dois *IpMailboxFolder*. Um sendo *inbox* e outro *outbox*. Para se conhecer os parâmetros de cada método de cada classe e como utilizá-los, explicações detalhadas também estão presentes na API, neste caso no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007b).

- **Obtenção de dados de capacidades de terminais** (*Terminal Capabilities*):

  Este grupo da API Parlay mostra como podem ser obtidas informações das capacidades de um dado terminal de usuário. De acordo com a especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007c), uma aplicação pode requisitar estas

informações, sempre que necessário, ou então receber notificações sempre que houver modificações nas informações, ou seja, sempre que alguma capacidade mudar no terminal em observação.

- **Serviço de mensagens multimídia** (*Multi-Media Messaging*):

  Grupo de *interfaces* que definem serviços capazes de realizarem envios e recebimentos de mensagens multimídia.

- **Gerenciamento de presença e disponibilidade** (*Presence and Availability Management*):

  Conforme a especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007d), este grupo padroniza as *interfaces* criadas para lidar com os serviços e processamentos relacionados com o armazenamento, recuperação e publicação de informações sobre Presença e Disponibilidade de entidades de várias formas de comunicações e o contexto no qual elas se encontram.

- **Interação de usuário** (*User Interaction*):

  Grupo de *interfaces* que definem como as aplicações interagem com os usuários finais.

- **Mobilidade** (*Mobility*):

  Grupo de *interfaces* que descrevem as capacidades da rede em reportar, a uma aplicação, a localização de uma entidade (um usuário ou dispositivo, por exemplo) sempre que há uma modificação de local ou sempre que a aplicação requisita dados de localização. A capacidade da rede em fornecer um relatório sobre a localização de um usuário é interessante, a tal ponto que, se possível, dados de altitude também são fornecidos, como comentado no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007e). Neste caso, a altitude é dada em metros acima do nível do mar.

- **Cobrança** (*Charging*):

  Grupo de *interfaces* relacionadas com cobranças de serviços utilizados pelos usuários. Por exemplo, uma aplicação de telecomunicações, disponível num servidor de aplicações, pode ser utilizada por um usuário ou mais, que se tornam clientes da empresa provedora de tal aplicação. Neste caso, a empresa deverá cobrar seus clientes pelo uso da aplicação. Por sua vez, tal aplicação demandará recursos da rede de uma operadora e, então, o custo de uso destes recursos influenciará no custo a ser passado para o usuário final. Assim, a aplicação deverá interagir com as *interfaces* disponíveis, via um *gateway*, responsáveis em receber e encaminhar requisições ao serviço de cobranças disponível na rede. Neste caso, se um usuário deseja reservar um tempo de uso da aplicação, comprando créditos para isso, a aplicação, ao interagir com as *interfaces* do serviço de cobrança, poderá relacionar os créditos comprados ao usuário determinado e também poderá aferir o quanto resta destes créditos, após o uso de certo tempo dos recursos necessários da rede. As *interfaces* de cobrança permitem que uma aplicação faça uma reserva de volume, para um usuário, ou faça uma compra de créditos. Ou seja, o usuário da aplicação pode comprar créditos em R$ ou reservar uma quantidade de *kilobytes*, emails, *html-pages*, a serem usados num serviço, como numa *conference-call* via VoIP.

O exemplo de diagrama de seqüência seguinte, retirado da especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007f), ilustra as ações da interação entre a aplicação cliente e o serviço de cobrança da rede, considerando um usuário que deseja utilizar tal aplicação e, então, deverá ser cobrado por isso.

Conforme a **Figura 9**, as instruções de 1 a 6 mostram como uma aplicação pode utilizar um *IpChargingSession* para fazer uma reserva de créditos. Tais créditos são debitados, pela aplicação, caso um usuário utilize a mesma. A reserva desses créditos é feita em quantidades definidas, através de uma invocação do método

reserveAmountReq() do *IpChargingSession*, ou seja, do objeto de *software* que implementa esta *interface*.

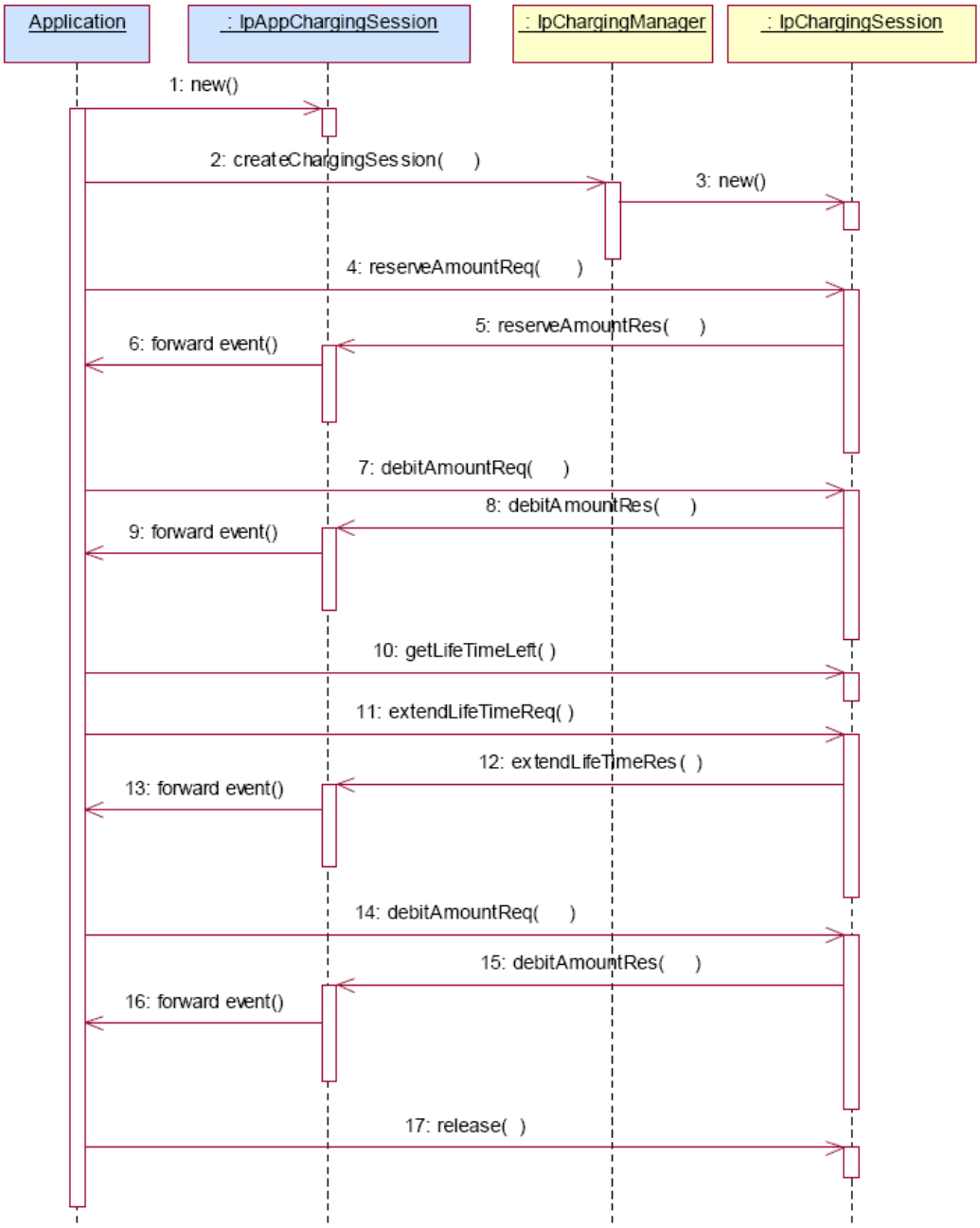

Figura 9 – *Diagrama de seqüência Payment in parts.*

A instrução 5 mostra uma resposta enviada à aplicação e para que a aplicação consiga receber tal resposta, apropriadamente, ela deve implementar a *interface IpAppChargingSession*. A instrução 5 mostra, então, que a resposta é enviada quando se tem o método *reserveAmountRes()* invocado no objeto com a

interface IpAppChargingSession. Como todas estas *interfaces* estão especificadas numa mesma API, fica possível implementar um objeto que pode executar métodos corretamente, em outros objetos, já que cada objeto conhece a *interface* de outro. Uma referência ao objeto implementado a *interface IpChargingManager*, deve ser, inicialmente, obtida pela aplicação, através de um *gateway*, por exemplo interagindo com o *Framework*. Conforme o documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007f), a aplicação utiliza tal objeto para então obter uma referência a outro objeto implementando a *interface IpChargingSession*.

Quando o usuário usa uma dada quantidade do serviço disponível via a aplicação, por exemplo, 1 minuto, ou 1 *kilobyte*, ou um envio de mensagem com SMS, a aplicação utiliza o método assíncrono *debitAmountReq*() para debitar tal quantidade dos créditos reservados pelo usuário. A aplicação pode saber quanto de crédito ainda resta para o usuário utilizá-la por mais algum tempo, por exemplo, invocando o método na instrução 10 da **Figura 9**. Mais uma vez, está mostrado que a API Parlay, neste caso, para cobrança, mostra quais são as *interfaces* já implementadas que a aplicação deve encontrar num *gateway* e quais são as *interfaces* que a própria aplicação deve implementar, para receber o retorno das requisições feitas às capacidades da rede. O diagrama de classes seguinte (**Figura 10**), retirado do documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007f), mostra os métodos de cada *interface* relacionada com cobrança e o uso das *interfaces* entre si.

Pela **Figura 10**, pode ser notado que o objeto *IpChargingSession* deve usar o objeto *IpAppChargingSession*. Por exemplo, se a chamada ao método *reserveAmountReq()* causar algum erro no serviço da rede, o objeto *IpChargingSession* deverá reportar o fato à aplicação cliente. Neste caso, sob um

padrão de nomenclatura seguido na API Parlay, este último objeto saberá que deve ser invocado o método *reserveAmountErr()* no objeto *IpAppChargingSession*. As semelhanças entre os nomes de métodos e *interfaces* fazem parte da nomenclatura, para facilitar a programação da implementação das *interfaces* e dos algoritmos proprietários das operadoras e dos provedores de aplicações. Ou seja, o que estiver programado no método *reserveAmountErr()* não importa ao objeto *IpChargingSession*, mas é interessante ao objeto *IpAppChargingSession*.

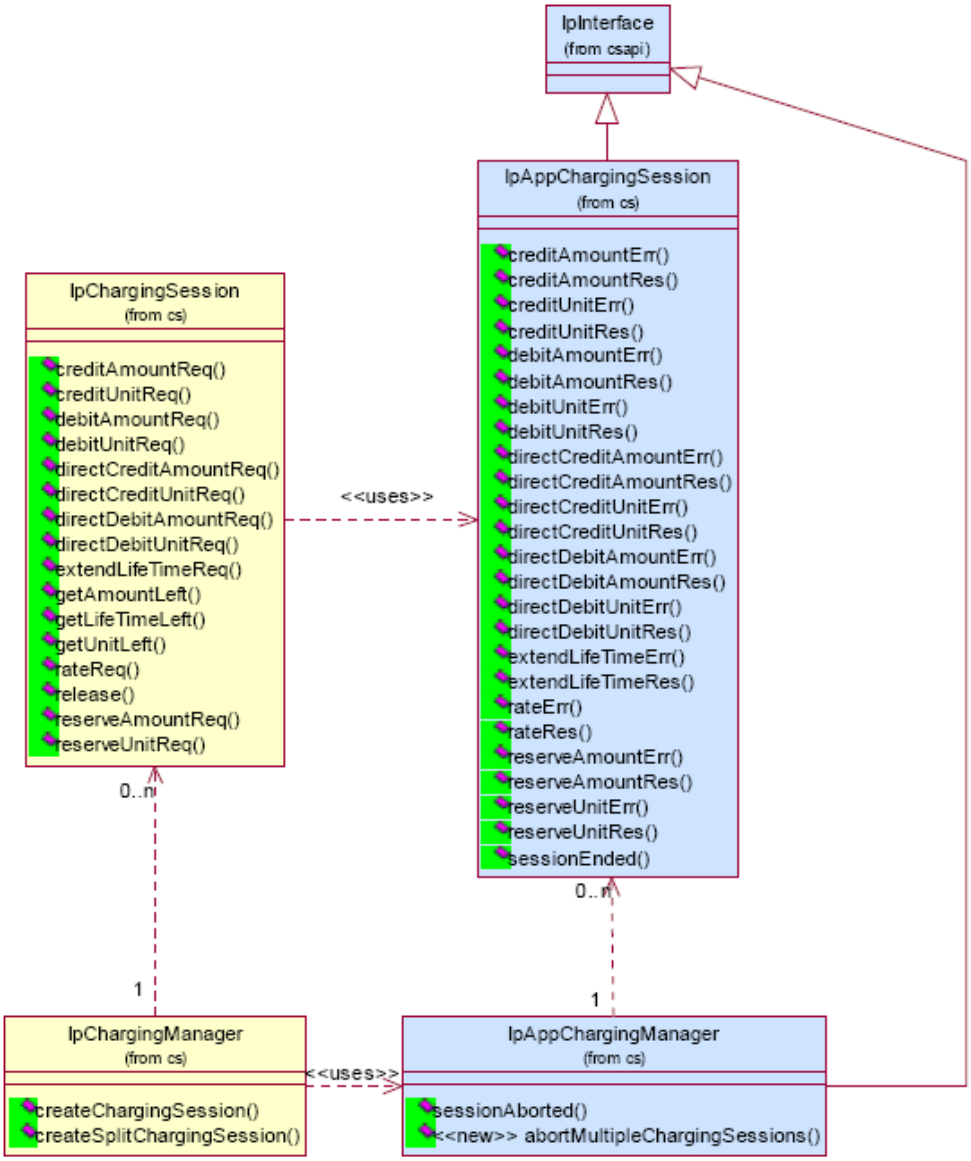

Figura 10 – *Diagrama de classes Application interfaces for Charging.*

Por fim, o que deve ser compreendido aqui é a necessidade de se conhecer, previamente, as *interfaces*, ao planejar um novo serviço de valor agregado. Ou melhor, se um objeto A deverá utilizar um objeto B, numa interação entre aplicação cliente e a rede, então A deverá conhecer os métodos disponíveis em B e saber como eles podem ser invocados corretamente. As regras de como invocar cada método, que métodos estão disponíveis em quais *interfaces*, quais são as *interfaces* que devem ser implementadas em cada uso de serviço e a ordem em que as coisas devem funcionar são descritas nas documentações das APIs Parlay. Isto mostra que o trabalho do Parlay, realmente, dita como as coisas funcionarão entre o domínio seguro de uma operadora de telecomunicações e o domínio das aplicações de terceiros. Estas regras ditadas já funcionam como se o consórcio Parlay estivesse dizendo: "*Atenção profissionais do futuro em TI e de Telecomunicações: aprendam como será possível usufruir de nossas redes, se quiserem continuar no mercado! Criem suas aplicações obedecendo às nossas especificações, para que nossas redes lhes forneçam o que precisam. Estas especificações proverão a união entre TI e Telecomunicações, necessária no nosso mercado.*"

3.2.2 Participantes do Parlay

Figura 11 – *Empresas do consórcio Parlay.*

A **Figura 11** mostra uma coleção de logomarcas de algumas empresas que participam do grupo Parlay:

Os membros do Parlay Group incluem empresas líderes na indústria de TI, vendedores de serviços Internet, desenvolvedores de *software*, vendedores e fornecedores de dispositivos de redes, fornecedores de aplicações, grandes e pequenas empresas. Uma dada empresa pode se tornar um membro do grupo Parlay, associando-se como um *Full Member* ou um *Affiliate Member.*

Tornando-se um membro do grupo Parlay, uma empresa terá as seguintes vantagens: compartilhar de versões futuras das especificações, contribuindo intelectualmente nos documentos e também nas especificações de testes. A empresa também será convidada a participar de todos os encontros e eventos Parlay, comitês e acessar a lista de e-mails do grupo. Também é dado acesso a um setor particular do *web site* do Parlay, disponível somente aos membros. Para se tornar um membro do grupo Parlay basta pagar uma taxa anual de U$ 20.000,00 ou uma anual de U$ 7.500,00, como *Full Member* ou *Affiliate Member* respectivamente.

3.3 Web Services

Quando uma aplicação do domínio de TI necessita utilizar um recurso do domínio de telecomunicações, como uma capacidade da rede, esta aplicação fará uso do OSA/Parlay *gateway*. Tal aplicação poderá estar num ponto, na rede, diferente do local onde estiver o OSA/Parlay *gateway*. Neste caso, a aplicação cliente e o *gateway* estarão compondo um sistema onde o cliente e o servidor não estarão no mesmo local, ou seja, estas entidades deverão utilizar alguma tecnologia para trocarem mensagens de requisições e respostas entre seus objetos. Em outras palavras, ocorrerão invocações de funções remotas. Como visto na **Figura 12**, retirada de

(LECLERC, 2003), pode-se usar a *Common Object Request Broker Architecture* (CORBA) como a tecnologia para a comunicação entre os objetos distribuídos. Assim, via CORBA, a aplicação cliente fará uma chamada a uma *procedure* remota do *gateway* e poderá receber a resposta necessária. Neste cenário, o *gateway* funciona como um serviço remoto que pode ser acessado.

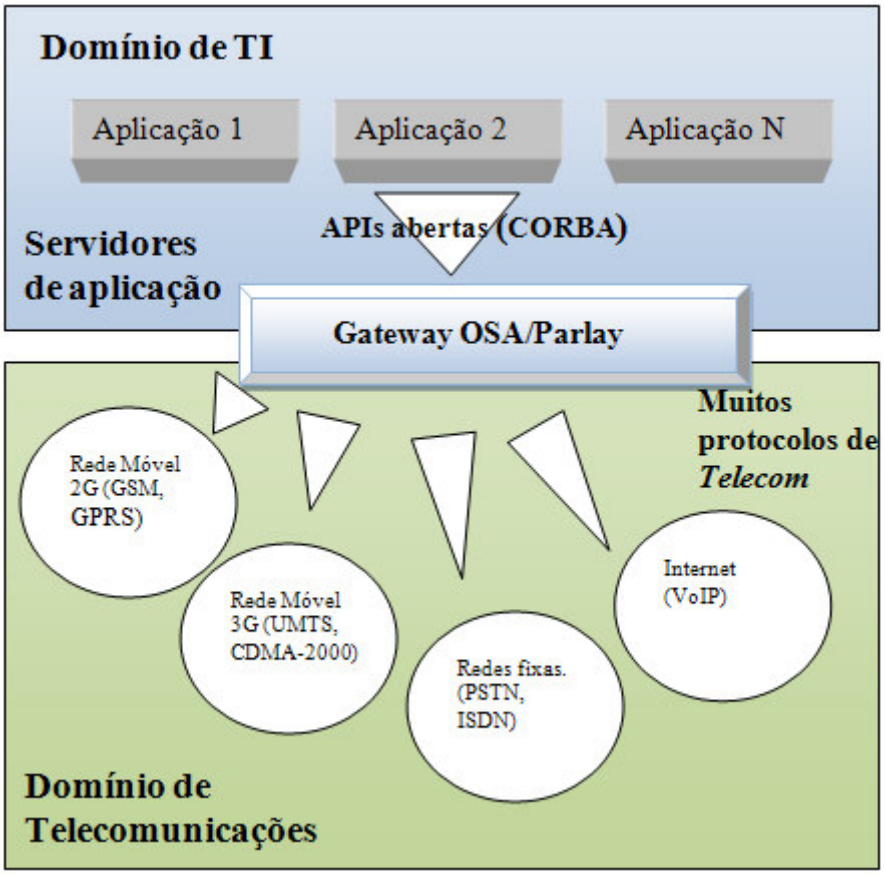

Figura 12 – *CORBA para comunicação com o gateway.*

Outra forma de tornar serviços disponíveis remotamente é usar a tecnologia de *Web Services*. Com essa tecnologia os objetos das aplicações também podem se comunicar, remotamente, distribuídos em máquinas diferentes, fazendo requisições e recebendo respostas, usando tecnologias da Internet, como o protocolo *Hypertext Transfer Protocol* (HTTP). Com a tecnologia *Web Services*, pode-se criar um *web service*.

Um *web service* é um sistema de *software* funcionando em alguma máquina na rede e, geralmente, é descrito em termos de sua *interface* com seus métodos. Possivelmente está publicado, isto é, com suas características declaradas e acessíveis para consulta em algum lugar da rede, e pode ser contatado para receber requisições remotas de outros sistemas clientes, tudo pela Internet. Ou seja, é um serviço acessível. A forma de se descrever um *web service*, via *interfaces*, é semelhante à forma de se descrever um serviço de telecomunicações na rede, isto é, um sistema cliente de um *web service* terá condição de saber quais são os métodos disponíveis num serviço e como invocá-los corretamente, esperando retornos com tipos determinados ou tratando exceções passíveis de ocorrer em caso de erro. Publicar um *web service* significa cadastrar informações sobre o mesmo, num ponto bem conhecido na rede e acessível por aplicações clientes. Tal ponto pode ser um serviço de diretório, por exemplo. Assim, uma aplicação cliente identifica como contatar o *web service*, como invocar métodos no mesmo e como implementar meios para receber notificações deste serviço via *web*. Este tipo de declaração sobre o que pode ser obtido com um *web service* é semelhante às declarações de serviços disponíveis via *gateway* Parlay, com as *interfaces* do OSA/Parlay. Por conseguinte, um *gateway* Parlay também pode estar disponível como um conjunto de *web services*, para que, então, as aplicações clientes tenham condição de interagir com o mesmo, via Internet. Mas isto sugere um mapeamento entre as *interfaces* Parlay, em UML, e as *interfaces* usadas para descrever os *web services*. Uma linguagem apropriada para tanto é chamada *Web Services Description Language* (WSDL) e está explicada mais adiante nesta dissertação, na seção 3.3.2. Por hora, o diagrama de blocos seguinte dá uma visão geral sobre o relacionamento de uma aplicação cliente e um servidor de *web service*:

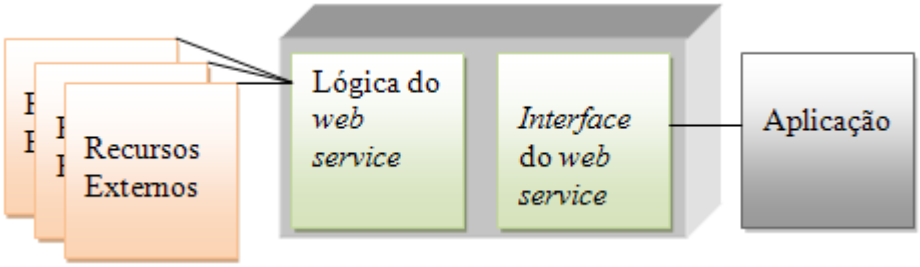

Figura 13 - *Visão geral de web service e Aplicação cliente.*

Na **Figura 13**, retirada de (THE PARLAY GROUP, 2002a), da esquerda para a direita estão representados os recursos da rede, um *web server* contendo *web service* e uma aplicação que pode ser um cliente de um serviço na *web*. O *web server* contém as *interfaces* dos serviços *web*, descritas em WSDL, que são úteis à aplicação cliente. O mesmo servidor também contém a implementação dos serviços descritos pelo WSDL, chamado de *web service logic*. As *interfaces* WSDL funcionam como uma definição de *gateway* entre a aplicação cliente e a implementação dos serviços necessários.

O *World Wide Web Consortium* (W3C) define um *web service* como "um sistema de *software* designado a suportar interações entre máquinas pela rede.", como visto em (WEB service, 2008). As interações se dão entre clientes (consumidores de *web services*) e servidores (*os web services*) utilizando mensagens em *eXtensible Markup Language* (XML) para a comunicação. A **Figura 14**, retirada de (CERAMI, 2002), ilustra esta idéia.

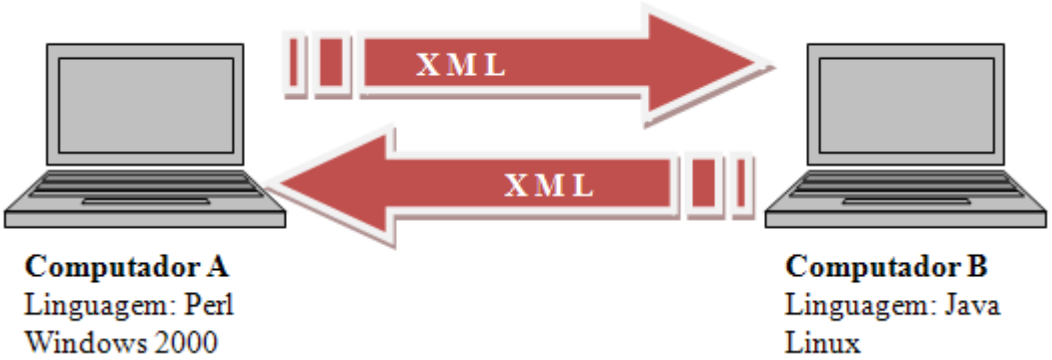

Figura 14 – *Comunicação entre máquinas com XML.*

A linguagem XML é independente de sistema operacional, protocolos e linguagens de programação. Assim, as aplicações de *software* podem interagir entre si, com trocas de mensagens em XML, mesmo se tais aplicações forem construídas em sistemas operacionais diferentes, com linguagens de programação diferentes.

As mensagens trocadas com XML podem obedecer ao padrão *Simple Object Access Protocol* SOAP (CERAMI, 2002). Portanto, as partes que se comunicam podem utilizar o protocolo SOAP. Na verdade, o protocolo SOAP foi criado para estabelecer as regras das trocas de mensagens de objetos distribuídos que interagem entre si. Desta forma, um objeto A pode acessar um objeto B remoto e invocar métodos no mesmo, num esquema cliente/servidor. Portanto, pode-se usar o SOAP para executar chamadas de procedimentos remotos - *Remote Procedure Call* (RPC). As mensagens trocadas sob as regras do SOAP estão em XML, como na **Figura 14**. Nestas mensagens XML podem estar descritos, por exemplo, os tipos de dados definidos pelas aplicações e uma representação das chamadas de métodos remotos, bem como das respostas. Tais mensagens seguem então uma gramática XML especializada, que padroniza o formato das suas estruturas. Estas mesmas mensagens compõem o método fundamental de troca de informações entre os *web services* e seus consumidores. Para um melhor entendimento, tem-se o exemplo descrito a partir da Figura 3-1 do livro (CERAMI, 2002), representada na **Figura 15** desta dissertação.

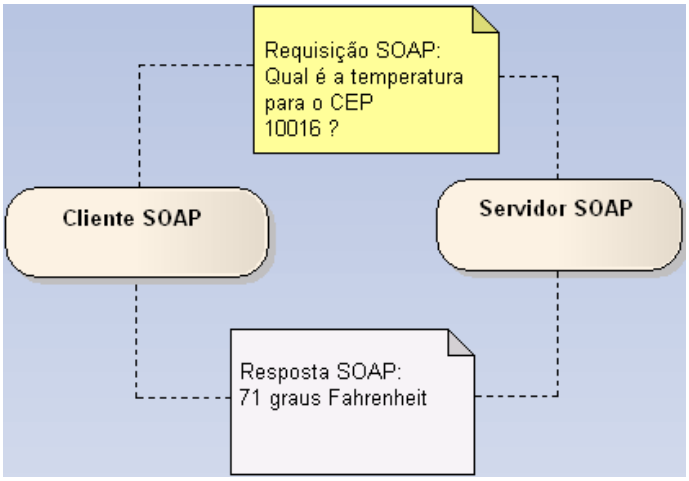

Figura 15 – *Exemplo de mensagens SOAP.*

Conforme o exemplo presente em (CERAMI, 2002), um serviço disponível na *web* é capaz de fornecer a temperatura em Fahrenheit, baseada na localização geográfica referenciada por um Código de Endereçamento Postal (CEP) fornecido previamente. Tal serviço encontra-se num servidor SOAP, identificado por *XMethods.net* e aceita requisições de temperatura, via mensagens SOAP. O método a ser invocado para a

obtenção da temperatura é identificado na mensagem SOAP, através da palavra *getTemp*. Tal método, ao ser chamado, deve conter um parâmetro que seja o CEP da região de onde se quer obter a temperatura. A temperatura obtida é um valor numérico *float,* ou seja, é um numeral com ponto decimal flutuante. Segue abaixo o exemplo da mensagem SOAP para a requisição da temperatura:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getTemp
      xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <zipcode xsi:type="xsd:string">10016</zipcode>
      </ns1:getTemp>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Na mensagem anterior há o Envelope e o corpo (*Body*), que devem constar obrigatoriamente. No corpo da mensagem há a especificação do método *getTemp()* sendo chamado e o valor do parâmetro *zipcode*. A mensagem anterior também usa 4 *namespaces*. Estes *namespaces* ajudam na identificação do Envelope (Ex: versão), na identificação dos esquemas XML usados para descreverem os tipos de dados transmitidos e também na identificação do provedor do serviço, neste caso: *urn:xmethods-Temperature.* O único parâmetro passado nesta invocação de método é o *zipcode* e seu formato é do tipo "*xsd:string*". Dependendo do método que está sendo invocado, outros parâmetros de outros tipos podem estar definidos na requisição SOAP. O "*payload*" principal da mensagem SOAP vem localizado no seu corpo (*Body*). Como visto, alguns detalhes sobre o SOAP estão apresentados neste documento, para possibilitar o entendimento da arquitetura usada na implementação do trabalho descrito mais adiante, no capítulo 5. Contudo, o protocolo SOAP não será exaustivamente detalhado, o que realmente não é necessário aqui.

A mensagem SOAP da resposta à requisição mostrada acima está no código XML seguinte:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getTempResponse
      xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <return xsi:type="xsd:float">71.0</return>
      </ns1:getTempResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Nesta resposta, o único elemento do *body* é o *getTempResponse*, que identifica uma resposta para a requisição feita ao método *getTemp()*. Neste elemento de resposta está presente o resultado "*xsd:fload*" valendo 71,0. Uma resposta contendo a indicação de uma falha, quando uma requisição é feita, está exemplificada a seguir:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
         <faultstring xsi:type="xsd:string">
            Failed to locate method (ValidateCreditCard) in class
            (examplesCreditCard) at /usr/local/ActivePerl-5.6/lib/
            site_perl/5.6.0/SOAP/Lite.pm line 1555.
         </faultstring>
      </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Neste exemplo, no *Body*, há o elemento *SOAP-ENV:Fault* indicando que a mensagem é um aviso de falha. O elemento *faultcode* indica que a falha foi causada pela requisição do cliente. Neste caso, devido a uma requisição ao método *ValidateCreditCard* não existente. O elemento *faultstring* traz uma mensagem inteligível sobre o que ocorreu.

3.3.1 SOAP via HTTP

As mensagens SOAP, que são textos XML obedecendo a uma gramática específica, precisam de uma regra – protocolo - para serem transportadas de nodo a nodo na rede. O protocolo SOAP não está amarrado a qualquer protocolo de aplicação da pilha TCP/IP. De fato, segundo explicações constantes em (CERAMI, 2002), o SOAP pode ser transportado via *Simple Mail Transfer Protocol* (SMTP), FTP e HTTP, por exemplo. Entretanto, a especificação SOAP inclui detalhes apenas sobre HTTP, sendo que este último permanece o mais popular protocolo para o transporte do SOAP. Além disso, as requisições SOAP são enviadas via HTTP *request* e as respostas SOAP são retornadas como conteúdo da HTTP *response*. Mesmo sendo possível enviar requisições SOAP via HTTP *GET,* a especificação do SOAP detalha apenas HTTP *POST*. O HTTP *POST* é preferido, porque a maior parte dos servidores limitam a quantidade de caracteres aceitáveis numa requisição *GET,* conforme explicações em (CERAMI, 2002). O código abaixo mostra uma requisição HTTP carregando uma requisição SOAP:

```
POST /perl/soaplite.cgi HTTP/1.0
Host: services.xmethods.com
Content-Type: text/xml; charset=utf-8
Content-Length: 538
SOAPAction: "urn:xmethodsBabelFish#BabelFish"

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:BabelFish
      xmlns:ns1="urn:xmethodsBabelFish"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <translationmode xsi:type="xsd:string">en_fr</translationmode>
         <sourcedata xsi:type="xsd:string">Hello, world!</sourcedata>
      </ns1:BabelFish>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Um requisito adicional é que as mensagens clientes de um *web service* devem especificar um cabeçalho SOAPAction. Tal cabeçalho é uma identificação -*Uniform Resource Identifier* (URI) - específica do servidor, usada para indicar a intenção da

requisição cliente. Isso possibilita determinar, rapidamente, a natureza da requisição SOAP, tornando desnecessário um exame do "*payload*" da mensagem SOAP. Na prática, este cabeçalho é lido pelos *firewalls*, como um mecanismo para bloquear ou despachar, rapidamente, uma mensagem ao servidor destino. Por exemplo, para acessar o serviço de tradução AltaVista BabelFish, hospedado por XMethods, deve-se especificar o "urn:xmethodsBabelFish#BabelFish" como um cabeçalho *SOAPAction*. Nesta requisição SOAP, é invocado o método BabelFish, que leva consigo dois parâmetros: *translationmode* e *sourcedata*. Segue um exemplo de resposta à requisição citada acima:

```
HTTP/1.1 200 OK
Date: Sat, 09 Jun 2001 15:01:55 GMT
Server: Apache/1.3.14 (Unix) tomcat/1.0 PHP/4.0.1pl2
SOAPServer: SOAP::Lite/Perl/0.50
Cache-Control: s-maxage=60, proxy-revalidate
Content-Length: 539
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <namesp1:BabelFishResponse xmlns:namesp1="urn:xmethodsBabelFish">
         <return xsi:type="xsd:string">Bonjour, monde!</return>
      </namesp1:BabelFishResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Para se criar um serviço na *web*, como um *web service* visto aqui, inicialmente deve-se construir o *software* que será o serviço em si. Por exemplo, se um serviço chamado Calculadora terá a capacidade de calcular raiz quadrada, então pode-se criar uma classe Java, chamada Calculadora com um método chamado raizQuadrada. Este método poderá esperar um parâmetro do tipo inteiro, ao ser invocado, e poderá retornar um valor real. Esta nova classe, que será um serviço na *web*, pode ser construída como uma classe comum Java, por exemplo. Ou seja, nada sobre *web service* será incluído nesta classe. Em seguida, do ponto de vista do servidor, deve haver um *web server* ativo em algum lugar da rede, capaz de interpretar as requisições HTTP para raiz quadrada. Além disso, deve haver um servidor SOAP,

capaz de interpretar as requisições SOAP transportadas nas mensagens HTTP. Quando uma requisição HTTP chega ao servidor *web*, este servidor passa o conteúdo da mensagem HTTP a um objeto do servidor SOAP que, por sua vez, interpreta a requisição. Se a requisição for de raiz quadrada, por exemplo, o objeto no servidor SOAP irá criar uma instância da classe Calculadora e executar o método *raizQuadrada*. Este mesmo objeto do servidor SOAP irá encapsular o resultado da raiz quadrada numa resposta SOAP, a qual será enviada numa resposta HTTP, pelo servidor *web*, para o cliente. Neste caso, a aplicação cliente também deve conter uma forma de receber e interpretar uma mensagem HTTP. Do ponto de vista da aplicação cliente, deve ser possível montar a requisição SOAP e enviá-la num HTTP POST. O envio deve ser para o endereço correto onde o servidor *web* em questão estiver ativo.

Atualmente, há muitas implementações do protocolo SOAP disponíveis, as quais contêm ferramentas para a criação dinâmica das mensagens em XML, de tal forma que o programador não necessita despender tempo com a programação obedecendo à sintaxe da gramática XML usada. Por exemplo, num projeto de *web service* pode-se usar a implementação Apache[4] do protocolo SOAP, conhecia como Axis2 (THE APACHE SOFTWARE FUNDATION, 2008). Esta é uma implementação em Java do SOAP, com código aberto à comunidade de desenvolvedores de *software*. Um exemplo de facilidade provida pelo Apache SOAP é a classe *Call*, disponível para uso, cuja instância pode receber o nome do método a ser invocado no serviço, bem como os parâmetros, para, então, criar, dinamicamente, a mensagem SOAP de requisição e enviá-la, via HTTP até o *web service* correto.

---

[4] A *Apache Software Foundation* dá suporte a uma comunidade para projetos de código aberto. Os projetos Apache são caracterizados por um processo de desenvolvimento de *software*, de consenso colaborativo e licença aberta, além de um desejo por *software* de alta qualidade.

3.3.2 WSDL

Considerando o *Parlay*, os serviços disponíveis na rede são declarados através das *interfaces* da API Parlay/OSA, como explicado anteriormente. Estas *interfaces* são descritas nos documentos texto, em PDF, disponíveis no *web site* do Parlay e apresentam diagramas em UML, os quais mostram as partes das mesmas, como os nomes, os métodos disponíveis em cada uma e como executá-los passando parâmetros necessários de forma correta. Posteriormente, estas *interfaces* podem ser mapeadas, quando codificadas, em linguagens de programação, como Java ou C++. Analogamente, os serviços na *web* – *web services* – também podem estar descritos através de *interfaces* com métodos e parâmetros. Tal descrição pode ser representada em XML. Ou seja, código XML também pode ser utilizado descrevendo quais são os serviços disponíveis com: (1) suas funções públicas e (2) os tipos de dados para todas as mensagens de requisições e respostas. O código XML pode informar também: (3) dados sobre o protocolo de transporte a ser usado e (4) o endereço da localização de um serviço específico. O interessante agora é que as *interfaces* Parlay, descritas com UML, também podem ser mapeadas neste código XML, de tal forma que os serviços da rede estejam disponíveis na forma de *web services*.

Como definido em (CERAMI, 2002), a WSDL é uma especificação de como descrever *web services* numa dada gramática, com XML. Portanto, com WSDL, fica possível mapear um *web service* em XML, provendo as 4 informações citadas no parágrafo anterior. Um serviço Parlay pode então ser mapeado através da WSDL. De fato, as APIs Parlay já são disponíveis juntamente com código XML que mapeia os serviços descritos por elas. Ou seja, em se tratando de código programável, uma *interface* Parlay pode estar representada com código Java (*interfaces* Java), mas também com arquivos XML. Neste contexto, WSDL representa um contrato entre quem requisita um serviço (objeto cliente), e quem o provê (*web service*), da mesma forma que uma *interface* Java representa um contrato entre código cliente e o objeto que a implementa. Para se ter serviços na *web*, como *web services*, é necessário que haja a descrição dos mesmos em WSDL. Portanto, no caso de *web services*, somente

o mapeamento de *interfaces* Parlay em *interfaces* Java não é suficiente, para se ter um sistema completo com serviços e clientes, como será visto mais adiante.

Segundo (CERAMI, 2002), a especificação WSDL diz que um documento XML, que mapeia um *web service*, deve conter alguns componentes obrigatórios. Estes componentes estão representados na **Figura 16**. O componente *definitions* deve ser o elemento raiz (primeiro e principal) em todos os documentos WSDL. Este componente define o nome do *web service*, declara alguns *namespaces* usados no documento e contém os outros elementos descritos a seguir:

- *Types*: este elemento descreve todos os tipos de dados usados entre o cliente e o servidor. Se o serviço usa somente tipos simples de dados, como inteiros e *strings*, o elemento *types* não é necessário. Exemplo retirado do trabalho (RECKZIEGEL, 200-?):

```
<wsdl:part name="endereco" type="xsd:string" />
<wsdl:part name="numero" type="xsd:int" />
```

- *Message*: este elemento descreve os atributos da mensagem, como o nome da mesma, e mostra o conjunto de elementos <*part*>, que podem ser parâmetros ou o retorno da mensagem. Exemplo retirado do trabalho (RECKZIEGEL, 200-?):

```
<wsdl:message name="getCidade">
        <wsdl:part name="cidade" type="xsd:string" />
</wsdl:message>
<wsdl:message name="getCidadeResponse">
        <wsdl:part name="getCidadeReturn" type="xsd:string" />
</wsdl:message>
```

- *PortType*: este elemento combina múltiplas mensagens para formar uma operação. Por exemplo: um *portType* pode combinar uma requisição e uma resposta numa única operação. Um *portType,* geralmente, define múltiplas operações.

- *Binding*: contém informações sobre qual protocolo será utilizado para transportar a mensagem SOAP. Exemplo retirado do trabalho (RECKZIEGEL, 200-?):

```
<wsdl:binding name="CidadeSoapBinding" type="impl:Cidade">
<wsdlsoap:binding style="rpc" transport=http://schemas.xmlsoap.org/soap/http
/>
```

- *Service*: este elemento define o endereço para invocar o serviço especificado. Comumente este elemento inclui uma identificação de localização - *Uniform Resource Locator* (URL) - para a chamada ao serviço. Exemplo retirado do trabalho (RECKZIEGEL, 200-?):

```
<wsdl:service name="ConsultaCidadeService">
    <wsdl:port binding="impl:ConsultaCidadeSoapBinding"
      name="ConsultaCidade">
     <wsdlsoap:address
      location=http://localhost/wsdl/ConsultaCidadeServer.php/>
    </wsdl:port>
</wsdl:service>
```

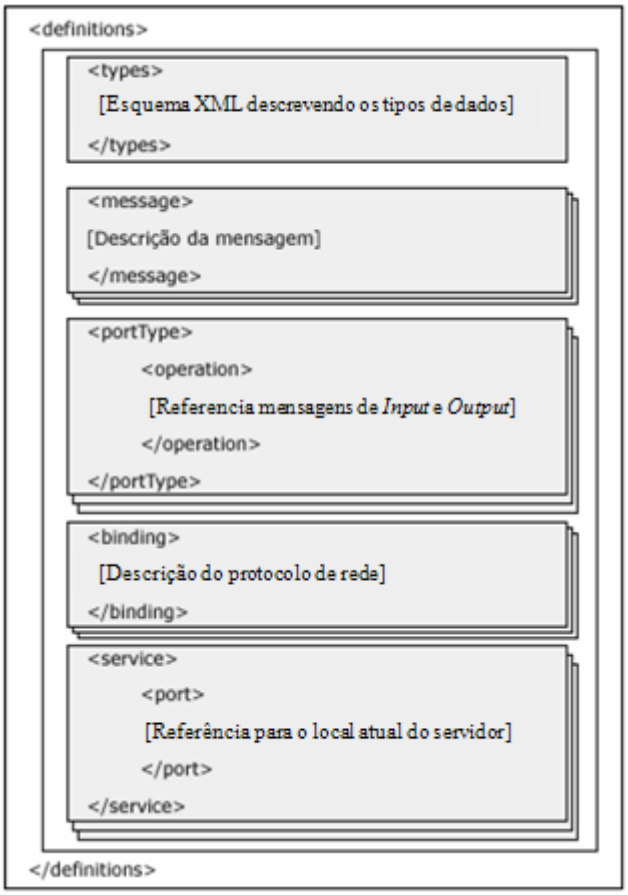

Figura 16 – *Componentes de um documento XML conforme a WSDL.*

A **Figura 16** mostra os componentes do XML, conforme a WSDL, e foi retirada do trabalho de conclusão de curso (RECKZIEGEL, 200-?). Além destes seis elementos, geralmente obrigatórios, a WSDL também especifica mais dois opcionais:

- *Documentation*: este elemento contém documentação inteligível aos desenvolvedores do sistema em si e pode aparecer contido em qualquer um dos elementos citados na **Figura 16**.
- *Import*: este elemento é usado para importar outros documentos WSDL, por exemplo. Assim os documentos WSDL podem ser criados numa forma modular.

Para um melhor entendimento das partes de um arquivo WSDL, abaixo, há um exemplo da especificação de um serviço simples, o qual tem a função de receber uma *string*, por exemplo *world* e, em seguida, retornar outra *string*, por exemplo *Hello, world*.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
  targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
  </message>
  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>

  <portType name="Hello_PortType">
    <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
    </operation>
  </portType>

  <binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="encoded"/>
      </input>
```

```
        <output>
          <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
        </output>
      </operation>
    </binding>

    <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
        <soap:address
          location="http://localhost:8080/soap/servlet/rpcrouter"/>
      </port>
    </service>
  </definitions>
```

Esse é um exemplo retirado da referência (CERAMI, 2002). O elemento *definitions* especifica que este documento é um mapeamento do serviço *HelloService*. Este elemento também define vários *namespaces* a serem utilizados adiante no mesmo documento. Dois elementos *message* estão definidos: o primeiro representando uma requisição e o segundo uma resposta. A mensagem de requisição, por exemplo, contém uma parte que é um parâmetro chamado *fistName*, cujo tipo é igual a uma "*xsd:string*". Este tipo é uma *string* definida conforme um dado esquema XML de definição de dados. O prefixo *xsd* referencia o *namespace* para o esquema XML, definido anteriormente com o elemento *definitions*. Assim, o processamento do sistema terá condição de saber qual é exatamente o esquema XML utilizado na definição deste dado *string*. Isto pode ser útil para uma análise sintática do código XML proposto. O elemento *portType* define uma operação, através do agrupamento de duas mensagens, uma de input e outra de *output*. Uma mensagem de *fault* também teria sido definida, se fosse necessário descrever alguma falha possível de ocorrer. Com o elemento *binding* é definido qual é o protocolo de transporte que deve ser utilizado para a transmissão das mensagens da operação já determinada, através da Internet. Dentro de um documento WSDL, é possível definir alguns elementos relacionados com o SOAP, que é o caso do exemplo acima. Assim, fica possível especificar atributos do SOAP úteis para a transmissão de requisições a serem direcionadas ao serviço especificado por esse WSDL. Por exemplo, a linha de código "<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>"

define que o transporte da mensagem será com SOAP sobre o HTTP. Se fosse usado "http://schemas.xmlsoap.org/soap/smtp", o transporte seria com SOAP sobre SMTP. O estilo *rpc,* nesta instrução, indica que o WSDL, contendo uma parte especificando elementos do SOAP, conterá em tal parte a descrição para um método de *input* e seu correspondente de *output*. Finalmente, o elemento *service* indica a localização do *web service*: location="http://localhost:8080/soap/servlet/rpcrouter"/>. Pela localização, tem-se que o *web service* está na própria máquina onde está o WSDL => o local é o *localhost*. O servidor *web* espera requisições pela porta 8080. Este mesmo servidor, quando recebe uma requisição, entrega-a para algum objeto capaz de decifrar a mensagem SOAP. Tal objeto deve estar localizado, relativamente, através do caminho */soap/servlet/rpcrouter*. O objeto localizado em */soap/servlet/rpcrouter* deve, então, ser capaz de traduzir a requisição SOAP numa invocação de um objeto que, realmente, irá receber uma string e concatenar o prefixo Hello na mesma.

Dado um WSDL, um cliente SOAP pode ser criado manualmente, para invocar o serviço. Ou seja, com o entendimento sobre o que está descrito no arquivo WSDL, entendimento tal que não requer aprofundamento no escopo desta dissertação, um programador poderia decidir por codificar manualmente um *software* em Java, com classes capazes de enviar requisições ao *web service* também correspondente a tal WSDL. Uma alternativa melhor é invocar o serviço, automaticamente, via uma ferramenta própria para isso. Tal ferramenta recebe o WSDL e gera as requisições SOAP necessárias. Ou seja, um dado *software* capaz de interpretar um arquivo WSDL, poderia compor as suas próprias mensagens SOAP e enviá-las ao *web server,* onde está o *web service* também correspondente ao mesmo WSDL.

Veja a **Figura 17**, presente também no livro (CERAMI, 2002).

Pelo visto, para se conseguir executar um método remotamente, num objeto servidor, basta ter acesso ao WSDL para o serviço alvo e uma ferramenta capaz de interpretar tal WSDL, para gerar a invocação correta no objeto do *web service*.

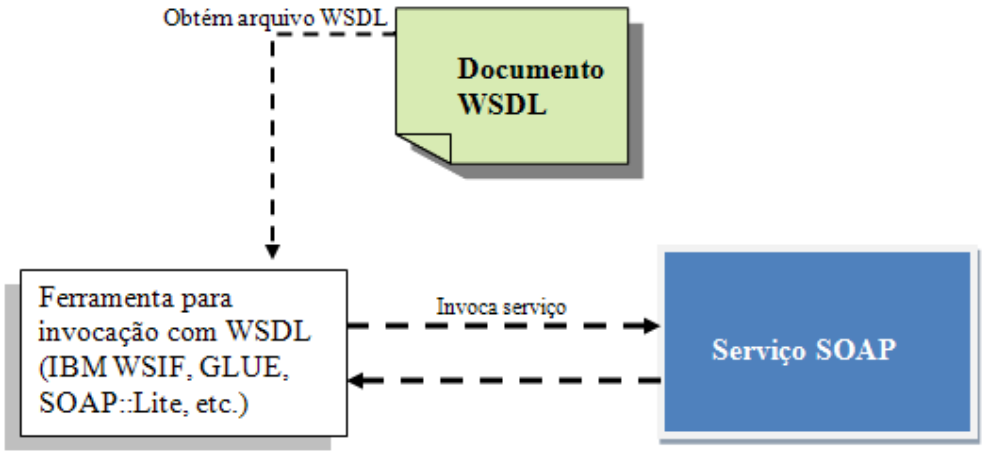

Figura 17 – *Ferramentas de requisição SOAP a partir de WSDL.*

3.3.2.1 Geração Automática de Arquivos WSDL

Uma das vantagens da WSDL é que, atualmente, existem várias ferramentas que permitem a criação de documentos nesta linguagem, a partir de classes já programadas de serviços. Ou seja, a partir de um serviço, pode-se definir o documento WSDL correspondente. Então, os documentos WSDL não demandam necessariamente grande esforço de programação. Um exemplo de ferramenta é o *software* Enterprise Architect, da empresa Sparx Systems.

3.3.3 UDDI

Enquanto WSDL descreve os *web services*, o SOAP transporta as mensagens de requisições e respostas destes serviços e, quando um provedor de *web service* publica o WSDL respectivo, neste mesmo documento está indicado o local onde o serviço é executado. Assim, o cliente do serviço, de posse do WSDL, terá condições de enviar

as invocações às funções necessárias, para o local correto. Contudo, pode haver casos em que uma aplicação cliente demanda um dado *web service*, mas tal serviço ainda não é conhecido. Por exemplo, se uma aplicação é responsável por enviar dados sobre o clima, para alguns usuários de telefones celulares, via SMS, então esta aplicação necessitará acessar um *web service,* em alum loca na rede*,* fornecendo dados como temperatura e previsão do tempo. Assim, há a necessidade de um ponto em comum (nodo na rede), onde os *web services* sejam publicados e descritos, ao mesmo tempo em que possam ser pesquisados e encontrados. Isto significa que é necessária a existência de um tipo de diretório para cadastro e pesquisa de serviços. Neste caso, um provedor de *web service* sobre o clima poderá, de alguma forma, cadastrar dados sobre seu serviço. Por exemplo, cadastrar o nome do serviço, as funções disponíveis no mesmo e o local de sua implantação (URL, por exemplo). Por outro lado, uma aplicação cliente poderá acessar este serviço de cadastros e pesquisar um *web service,* talvez pelo nome, ou tipo de serviço. Uma tecnologia que já existe para a implementação deste tipo de diretório, desde setembro de 2000, lançada pela Microsoft, IBM e Ariba, conforme citação em (CERAMI, 2002), se chama UDDI.

O UDDI é uma especificação técnica para a descrição, descoberta e integração de *web services*. Portanto, a UDDI é uma parte crítica na pilha emergente de funcionalidades dos *web services*. Esta tecnologia apresenta sua própria API, com funções que permitem o cadastro e pesquisa de informações sobre *web services* existentes pelo mundo.

Nesta dissertação, serão consideradas somente situações em que um serviço UDDI não é necessário. Isto porque as localizações dos arquivos WSDL, na *web,* serão previamente conhecidas, quando estes forem capacidades de redes de telecomunicações, por exemplo. Neste caso, os detalhes de UDDI, como sua API não estão descritos aqui.

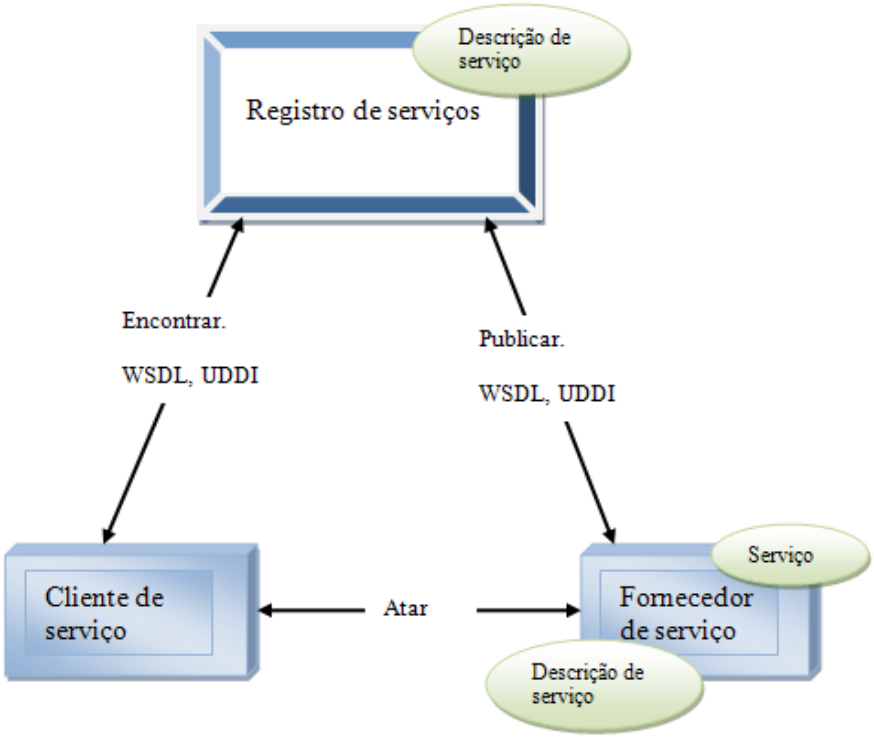

Figura 18 - *Arquitetura da tecnologia Web Service.*

A **Figura 18**, retirada de (THE PARLAY GROUP, 2002b), mostra a arquitetura da tecnologia *Web Services*, numa visão geral sobre o que foi explicado até aqui em relação a esta tecnologia.

3.3.4 Sumário sobre Web Services

Considerando as explicações dadas até aqui, neste capítulo, caso seja desejável tornar disponível uma ou mais capacidades de uma rede de telecomunicações, para aplicações de terceiros, então uma solução para tal disponibilidade poderia ser implementada da seguinte forma:

a) Escolher quais *interfaces* da API Parlay devem ser implementadas para realizar, apropriadamente, as capacidades demandadas.
b) Codificar e implementar as *interfaces* escolhidas. A implementação pode ser em Java, por exemplo. Até este ponto, tal trabalho ainda não precisará considerar a arquitetura distribuída com W*eb Services*. Mas, uma forma de

traduzir os arquivos WSDL respectivos em código Java inicial, se disponível, é algo de grande valia.

c) Um servidor *web*, como o Apache Tomcat e um servidor SOAP, como o Apache SOAP (Axis2), devem ser postos a funcionar numa máquina que seja acessível via Internet.
d) Na mesma máquina, pôr a funcionar as implementações das interfaces.
e) Criar um *web service* capaz de receber requisições direcionadas às instâncias das *interfaces* Parlay.
f) Criar um objeto capaz de receber requisições HTTP e passar a requisição SOAP respectiva ao *web service*.
g) Criar o arquivo WSDL que mapeia o *web service,* se ainda não existir, e mantê-lo num local acessível às aplicações clientes de terceiros.

O entendimento completo destes itens de (a) a (f) se dará no fim da leitura deste documento, o que culminará na última figura mostrada no mesmo. A ação descrita em (g) está fora do escopo deste documento e não será detalhada.

Felizmente, a API OSA/Parlay está disponível juntamente com os arquivos WSDL, respectivos, e também com as *interfaces* já codificadas em Java. Como o próprio Parlay já disponibiliza os arquivos WSDL a serem usados, isto evita dúvidas sobre as versões, por exemplo, de esquemas usados de codificações de dados e *namespaces*.

3.3.5 Versões de Parlay para Web Services

Como o Parlay/OSA, a tecnologia *Web Services* fornece um conjunto de APIs que permite a entidades de uma rede baseada em IP participarem de um mercado de serviços, de uma forma aberta e padronizada, segundo (THE PARLAY GROUP, 2005). Mas, o principal problema dos *web services,* na perspectiva de uma operadora de rede, é como integrá-los às redes de telecomunicações, sem expor essas redes aos riscos de uso desenfreado dos seus recursos ou mesmo ao ataque intencional, ao

mesmo tempo preservando a habilidade das operadoras de garantir qualidade de serviço e disponibilidade integral das redes. Para resolver estas questões, a especificação Parlay/OSA foi estendida, provendo as tecnologias Parlay Web Services e Parlay X Web Services, que provêem a ligação entre redes de telecomunicações e a tecnologia *Web* Services, como dito em (THE PARLAY GROUP, 2005). Ao mesmo tempo, isto aumenta a base de desenvolvedores que podem acessar as funcionalidades de telecomunicações. Parlay Web Services serão serviços na *web* definidos através das *interfaces* Parlay, já citadas neste documento. Para tal, tem-se os arquivos WSDL que mapeiam estas *interfaces*. Enquanto que Parlay X especifica como expor os serviços de telecomunicações na rede, também através de *Web Services*, mas de uma forma simplificada, ou seja, com funcionalidades mais simples de utilização, considerando isso pela ótica de um desenvolvedor de *software* sem conhecimentos em telecomunicações.

3.4 Parlay X

Do ponto de vista comercial, tanto Parlay quanto *Web* Services são tecnologias que buscam aumentar, de forma expressiva, a quantidade de serviços oferecidos pelas redes, promovendo APIs direcionadas à grande comunidade de desenvolvedores de Tecnologia da Informação. Como Parlay/OSA é uma tecnologia madura, de acordo com (THE PARLAY GROUP, 2005), ela fornece um ponto adequado de entrada em um novo modelo de negócios onde as operadoras são as detentoras dos serviços de telecomunicações. Neste modelo de negócios as operadoras podem utilizar Parlay/OSA Web Services e Parlay X Web Services para aumentarem seus "suprimentos" de redes de desenvolvedores e provedores de conteúdos, por exemplo.

A tecnologia Parlay Web Services provê a definição de *interfaces* e a definição da infra-estrutura para o uso de *web services* no ambiente de telecomunicações. Quando estas tecnologias são combinadas, um novo elemento é introduzido na rede: o Parlay

Web Services *Gateway*. A **Figura 19**, retirada de (THE PARLAY GROUP, 2002a), mostra este novo elemento.

3.4.1 Arquitetura de um *gateway* Parlay Web Services

Na arquitetura mostrada na **Figura 19**, o *gateway* Parlay/OSA é o mesmo *gateway* já citado neste documento, ou seja, a sua funcionalidade continua a mesma e também pode se comportar como um fornecedor de serviços ao *gateway* para Parlay Web Services. Por outro lado, pode-se considerar que o *Parlay Web Services Gateway* é um *link* entre a aplicação que demanda serviços de telecomunicações e o *gateway* Parlay/OSA que fornece acesso aos serviços.

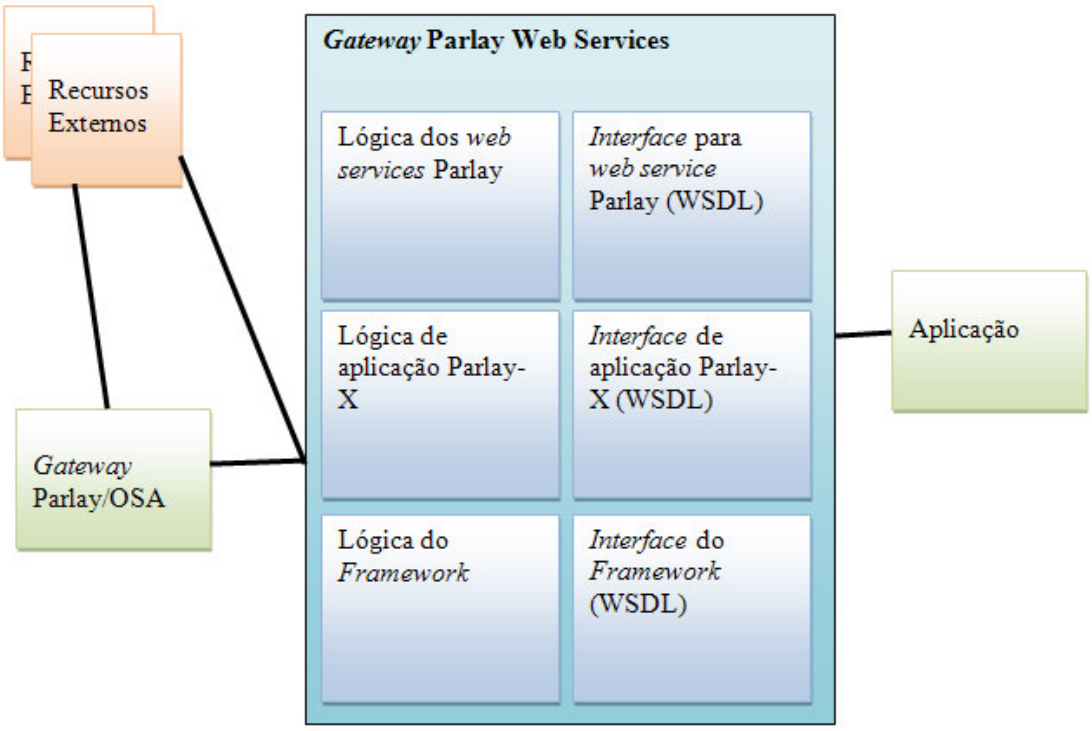

Figura 19 – *Parlay Web Services Gateway.*

A vantagem evidente deste *link* é a possibilidade de expor serviços de telecomunicações numa forma de serviço via *web*, como já citado anteriormente. Os serviços *web* também podem se comunicar diretamente com os recursos de rede,

mas, para os recursos não disponíveis diretamente, o *gateway* Parlay/OSA é o ponto de entrada seguro numa rede de uma operadora, por exemplo.

Os arquivos WSDL, correspondentes às *interfaces* Parlay/OSA, foram mapeados diretamente da modelagem UML. De fato, os serviços encontrados e disponíveis num *gateway* Parlay Web Services são aqueles que podem ser conhecidos e acessados usando-se os arquivos WSDL já disponíveis pelo Parlay. Isto inclui o WSDL que descreve as funções da *interface* Framework.

3.4.2 A API Parlay X

Além dos arquivos WSDL que descrevem os serviços mapeados nas *interfaces* da API OSA/Parlay, para uso em *gateways Parlay Web Services*, há também um grupo de *interfaces* mais simples resultante de uma padronização do grupo Parlay, ETSI e 3GPP, para aplicações demandando acesso às capacidades das redes de telecomunicações. Este grupo de *interfaces* mais simples é o que compõe a tecnologia Parlay X. Enquanto a API OSA/Parlay já foi mapeada em WSDL e também arquivos Java, a API Parlay X é mapeada somente em WSDL, porque esta API foi criada para a descrição dos *web services,* somente. O conjunto das *interfaces* Parlay X, ou um subconjunto deste, quando implementado para formar um conjunto de *web services*, vem a constituir o *gateway* Parlay X.

Já que a API Parlay X é mais simples que a API OSA/Parlay, como afirmado em (YIM, et al., 2006) e (WEGSCHEIDER, et al., 2005), isto aumenta ainda mais a possibilidade de desenvolvedores de *software* criarem aplicações com valor agregado para as telecomunicações. Isso é verdade porque, segundo (THE PARLAY GROUP, 2002c), o conjunto de *interfaces* de alto nível do Parlay X é orientado para os níveis de conhecimento em telecomunicações dos desenvolvedores de aplicações para a *web* e, segundo (YIM, et al., 2006), eles poderão trabalhar sem "*network expertise*".

Além disso, a especificação da API Parlay/OSA não é fácil de se compreender, para leitores sem conhecimentos sobre telefonia baseada em circuitos comutados, como dito em (GLITHO, 2004). O fato é que muitos conceitos nos quais Parlay/OSA se baseia são provenientes desta telefonia.

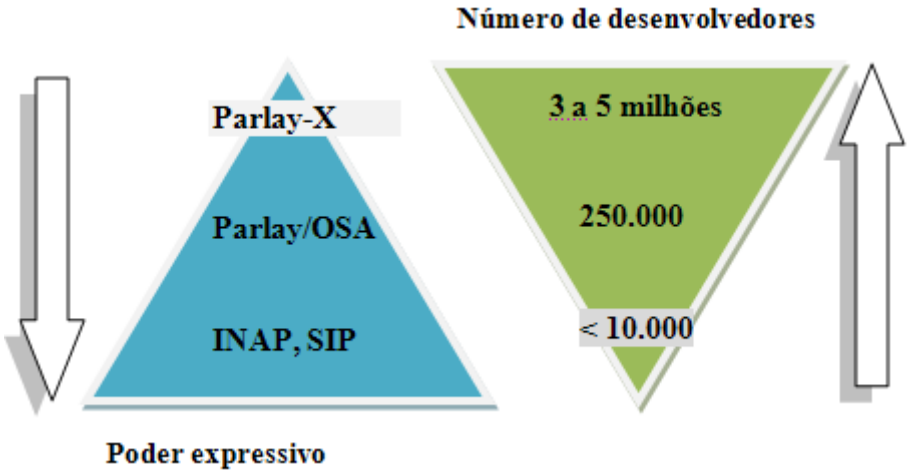

Figura 20 – *Distribuição de desenvolvedores por tecnologia.*

A **Figura 20**, retirada de (ERICSSON, 2006), mostra que o número de desenvolvedores capazes de trabalharem em novas aplicações para as NGNs é, inversamente, proporcional ao nível de complexidade das tecnologias a serem utilizadas para tal fim. Por outro lado, tecnologias menos complexas, na pirâmide acima, fornecem poder menos expressivo de controle de aplicações nas redes de telecomunicações.

Na verdade, as tecnologias que estão mais no topo, como Parlay-X, trabalham num nível de abstração mais elevado que aquelas subjacentes. Desta forma, o desenvolvedor da aplicação cliente de serviços de telecomunicações pode abstrair-se de detalhes desta área, sendo que neste caso ele tem acesso a um número de funções reduzido, para lidar com os serviços das redes, se comparado ao número que teria ao trabalhar diretamente com tecnologias de mais baixo nível, como SIP. Contudo, a quantidade de serviços oferecida através de APIs Parlay X, conforme deve estar entendido entre as empresas do consórcio Parlay, é suficiente para a criação das aplicações inovadoras desejadas. Isto significa que não adiantaria liberar acesso às

funções complexas de uma rede de telecomunicações, se nem mesmo aplicações simples poderiam ser cridas, em número satisfatório, pelos desenvolvedores de *software*, devida à maior dificuldade de construí-las neste caso. Assim, pode-se utilizar Parlay X, que é mais simples que Parlay/OSA.

3.4.3 Comparação entre Parlay X e Parlay/OSA

É pertinente comparar a especificação da API OSA/Parlay com a especificação da API Parlay X, de alguma forma, para reparar que a segunda é mais simples que a primeira. Por exemplo, nas duas especificações existem *interfaces* para funções de controle de chamadas de terceiros. Na API Parlay X, versão 2.1, uma única *interface* de controle de chamadas de terceiros está no documento chamado *Third Party Call* e na API OSA/Parlay estas *interfaces* estão no documento chamado *Generic Call Control*. O documento *Third Party Call* contém 14 páginas, enquanto que o documento *Generic Call Control* contém 69. Além disso, o documento *Generic Call Control* faz referência a outro documento de definição de dados, chamado *Call Control Common Definitions*, o qual contém mais 32 páginas. A *interface* definida no documento *Third Party Call* se chama ***ThirdPartyCall***. A *interface ThirdPartyCall* contém apenas quatro operações de fácil entendimento, para a realização de controles de chamadas:

- *makeCall;*
- *getCallInformation;*
- *endCall;*
- *cancelCall;*

O método *makeCall* é simples e exige apenas 2 parâmetros: o número de quem solicita a chamada (terminal chamador) e o número de quem é chamado (terminal chamado). De acordo com (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a), uma aplicação comum de convergência é a *Click to Dial*, na qual um portal (*web site*) estilo *selfservice* fornece uma página *web* que pode iniciar uma chamada entre 2 telefones. Isto é, uma página *web* onde se pode citar 2 números de telefones e solicitar uma

ligação entre eles. A seguir está presente o diagrama de seqüência da UML, retirado de (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a), representando este caso e demonstrando a simplicidade da situação do ponto de vista do desenvolvedor de tal serviço *selfservice* na *web*.

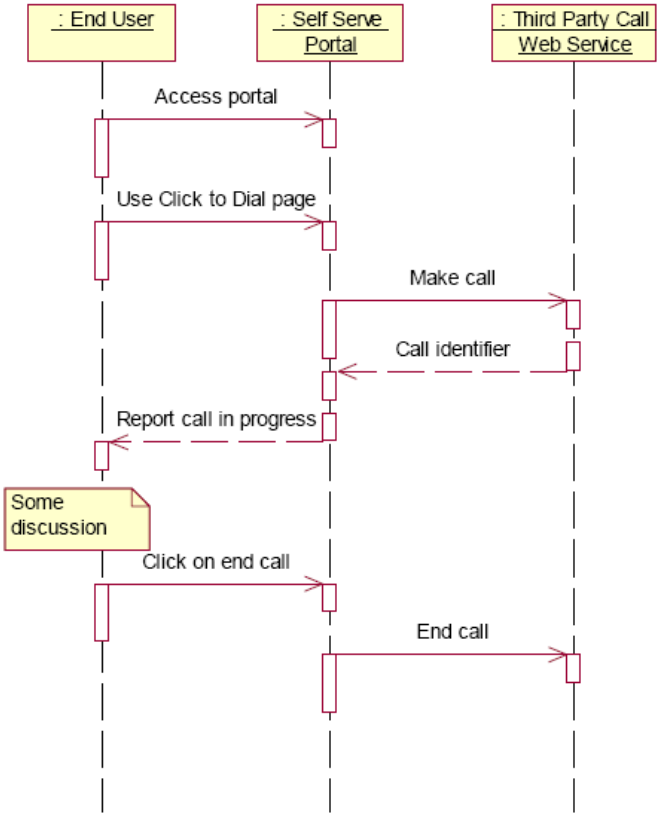

Figura 21 – *Diagrama de seqüência Click to Dial web portal.*

Já, para se fazer controle de chamada utilizando serviços encapsulados pelas *interfaces* da API OSA/Parlay, é necessário o entendimento de quatro *interfaces* diferentes. Estas quatro *interfaces* contêm juntas 32 métodos, como pode ser visto em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2007).

Atualmente (versão 2.1), os seguintes documentos descrevem a API Parlay X: *Common, Third Party Call, Call Notification, Short Messaging, Multimedia Messaging, Payment, Account Management, Terminal Status, Terminal Location,*

Call Handling, Audio Call, Multimedia Conference, Address List Management, Presence, Message Broadcast, Geocoding, Application-driven Quality of Service, Device Capabilities and Configuration, Multimedia Stream Control e *Multimedia Multicast Session Management*. Estes documentos foram atualizados em junho de 2007, como rascunho para a versão 3.0, e podem ser obtidos no *web site* do Parlay ou no *web site* do ETSI (THE EUROPEAN TELECOMMUNICATIONS STANDARDS, 2008).

3.4.4 Relacionamento entre *gateway* Parlay e IMS

Se um *gateway* Parlay ou Parlay X for imaginado, considerando a **Figura 1**, então ele estará situado na elipse com o texto "Funções de Suporte à Aplicações/Serviços". Isto é verdade porque este tipo de *gateway* constitui a função das NGNs de suportarem o relacionamento entre as aplicações de terceiros e os serviços provenientes das redes. Os serviços provenientes das redes podem ser suportados sobre IMS, numa NGN, segundo o artigo da IBM (IBM, 2008). Neste caso, a **Figura 22**, que foi retirada de (IBM, 2008), mostra a localização do *gateway* Parlay. A camada de controle vista na **Figura 22** está representada pelas funções de controle de serviço vistas na **Figura 1.** A comunicação entre o *gateway* OSA e a *Call Session Control Function* (CSCF), vista na **Figura 22** está representada, na **Figura 1**, pela comunicação entre as funções de suporte a aplicações/serviços e o componente de serviços IP Multimídia. A CSCF é um nome geral que se refere a servidores SIP ou *proxies* e é um dos principais elementos na camada de controle. A CSCF manipula registros SIP de terminais e processa mensagens SIP que vêm da camada de serviços. O *Home Subscriber Server* (HSS) é um banco de dados para manter dados únicos de usuários dos serviços. O uso de tal banco de dados está representado pelas funções de perfís de usuários, da camada de serviços, vista na **Figura 1.**

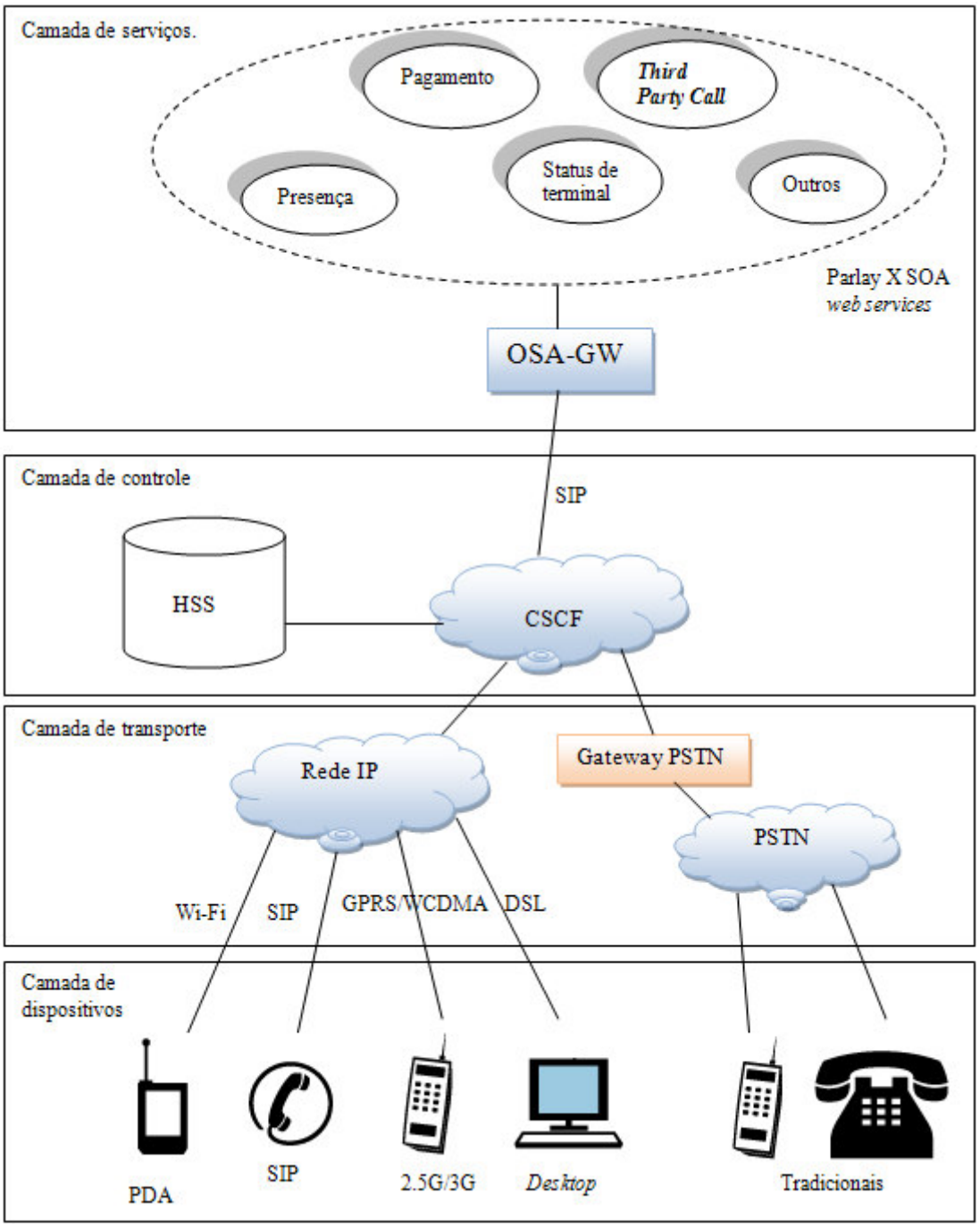

Figura 22 – *Diagrama da arquitetura IMS.*

3.4.5 Exemplos de Aplicações com Parlay X

Para um bom entendimento da aplicabilidade de Parlay X e como essa tecnologia é encaixada, por exemplo, num serviço envolvendo telecomunicações, seguem abaixo

2 exemplos que a utilizam. O primeiro exemplo faz uso do SMS e suas capacidades. O segundo exemplo utiliza o serviço que controla *Third Party Call*. Sendo que os dois serviços são especificados com Parlay X. Aqui, é interessante notar que esses serviços e suas capacidades são demonstrados em usos, cujos contextos produzem valores agregados às aplicações dependentes das mesmas. A maior ou menor valorização dada a uma nova aplicação, que usa serviços de telecomunicações, depende da utilidade desta aplicação, da demanda reprimida pela mesma e de outros fatores não tecnológicos.

3.4.5.1 SMS

Se uma aplicação deve usar recursos de SMS para envio e recebimento de mensagens, então pode ser necessário o acesso às funções disponíveis através de elementos de rede como um *Short Message Service Center* (SMS-C). Este é um elemento de rede capaz de entregar mensagens curtas em redes de telefones móveis. Esta idéia requer um grau de "*network expertise*", conforme comentado em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006b). Alternativamente, é possível usar Parlay/OSA, num *gateway* respectivo. Mas, ainda conforme (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006b) as *interfaces* da API Parlay/OSA são usualmente consideradas complexas pelos desenvolvedores de aplicações em TI. Neste caso, pode-se seguir a idéia de interagir com um *gateway* Parlay X. Tal *gateway* deverá conter as implementações para as *interfaces* descritas no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006b). Este documento, especifica uma *interface* chamada *SendSms*. Esta *interface* é facilmente compreendida e pode ser usada de forma simples. Ou seja, seus métodos são descritos numa forma que torna o uso deste serviço fácil de ser compreendido pelos desenvolvedores de aplicações. De fato, os criadores das aplicações podem programar as requisições de envio de mensagens, sem conhecimento de tecnologias de telecomunicações.

A API (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006b) descreve operações para envio de mensagem SMS, mas também um mecanismo para monitoramento do *status* de entrega da mensagem. Além disso, existe também um método de notificação assíncrona, que permite, por exemplo, notificar uma aplicação, cliente do *gateway,* sobre o recebimento de uma mensagem SMS enviada a partir de um terminal.

Para o envio de uma mensagem de SMS, a aplicação deve interagir com o *gateway* Parlay X e então invocar o método *sendSms* da *interface SendSms*. Tal método, segundo a API, requer 5 parâmetros, sendo que 3 são opcionais. A **Tabela 1**, copiada do documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006b), descreve os parâmetros da operação *sendSms*:

Tabela 1 - *Parâmetros da operação sendSms*

Nome	Tipo	Opcional	Descrição
addresses	xsd:anyURI [1..ilimitado]	Não	Endereço do destinatário
senderName	xsd:string	Sim	Nome do remetente
charging	commom:ChargingInformation	Sim	Cobrança a ser aplicada por essa mensagem
message	xsd:string	Não	Texto a ser enviado no SMS
receiptRequest	common:SimpleReference	Sim	Define o nodo da rede onde a aplicação está, o nome da interface do objeto que receberá as chamadas assíncronas do *gateway* e um *correlator* (identificador) da mensagem SMS em questão. O *receiptRequest* é usado para notificar a aplicação sobre o envio com sucesso ou falha da mensagem ao destinatário.

O campo *receiptRequest*, quando presente, contém as informações referentes à aplicação que envia a mensagem, como a sua localização na rede e o nome de uma

interface implementada por ela. Tais informações são, posteriormente, usadas pelo *gateway* Parlay X, quando for necessário notificar à aplicação que a mensagem foi entregue com sucesso, por exemplo. Para isso, o *gateway* Parlay X invoca uma operação implementada na aplicação, encapsulada pela *interface* identificada, sendo que essa implementação é acessível na rede, na localização declarada no campo *receiptRequest*. Este mecanismo de notificação assíncrona é perfeitamente implementável com a linguagem de programação Java. É o mecanismo de *callback*. A *interface* para isso se chama *SmsNotification* e o método a ser invocado se chama *notifySmsReception*. Portanto, a aplicação deve implementar a *interface SmsNotification,* o que garantirá ao *gateway* que o método *notifySmsReception* estará também implementado e acessível.

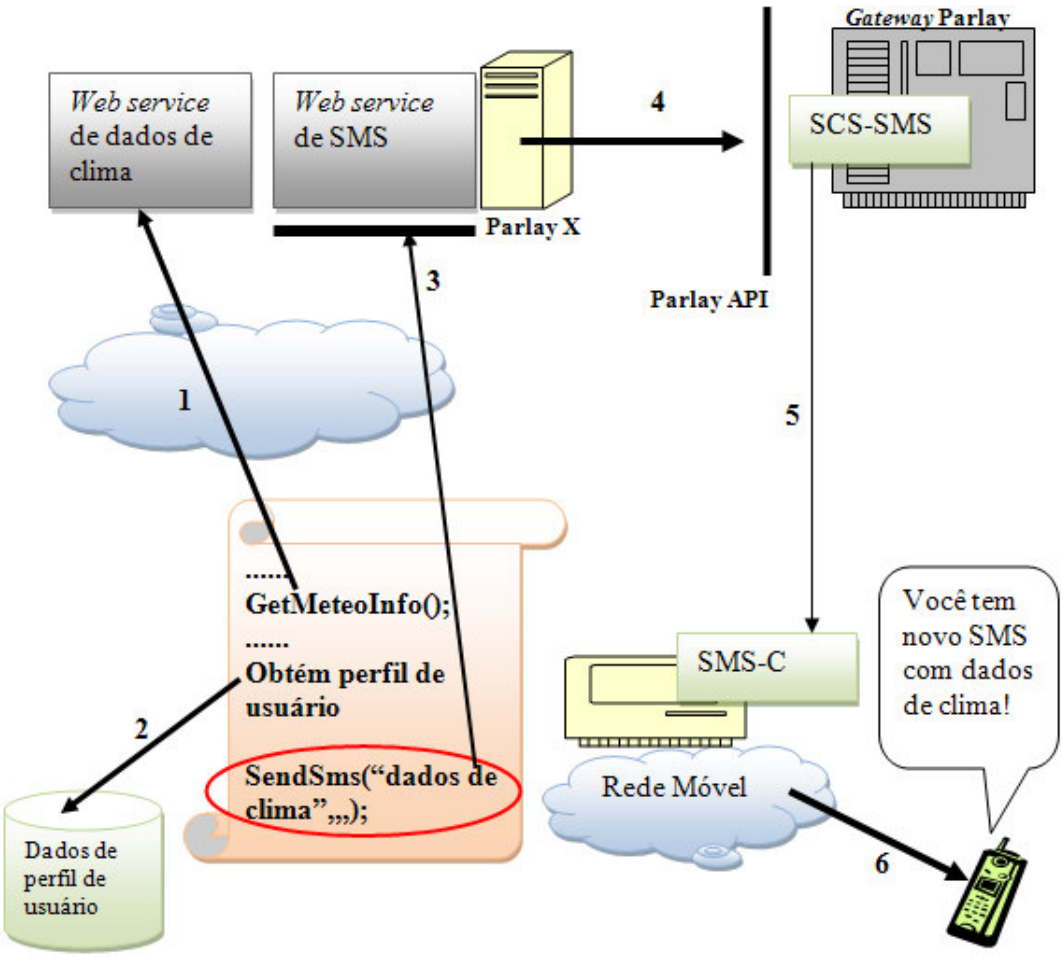

Figura 23 - *Cenário de envio de SMS.*

A **Figura 23** constitui um "diagrama de blocos" mostrando um exemplo de aplicação que utiliza a API Parlay X – *Short Messaging*, vista em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006b).

A **Figura 23** representa uma aplicação responsável em obter dados de meteorologia e, então, passar estes dados ao dispositivo móvel de um assinante desta mesma aplicação. A seqüência numérica de eventos fica como segue:

1 – A aplicação, provavelmente criada por um desenvolvedor de TI, requisita dados de meteorologia, acessando um serviço de informações do tempo, via um w*eb service* apropriado. Aqui, não é obrigatório o uso de *Web Services*. Também poderia ser usado apenas HTTP, ou CORBA, ou *JINI* (NEWMARCH, 2008), por exemplo.

2 – Em seguida, a aplicação obtém os dados do perfil do assinante, provavelmente acessando um banco de dados local, para saber, por exemplo, se a temperatura deve ser informada em Celsius ou Fahrenheit. Com todas as informações necessárias obtidas, a aplicação deve obter um meio de enviá-las até o terminal do assinante.

3 – A aplicação, utilizando a tecnologia *Web Services*, interage com um *gateway* Parlay X e invoca a função *sendSms* no objeto que implementa a *interface SendSMS.* Este objeto é implementado pelos desenvolvedores do *gateway* e a requisição do serviço é enviada com SOAP. Como explicado na seção 3.3.1, a mensagem SOAP pode seguir contida numa mensagem HTTP.

4- O *gateway* Parlay X deve enviar a mensagem de SMS usando recursos da rede de telecomunicações ou usando recursos de um *gateway* Parlay/OSA. Na **Figura 23**, está representada a utilização de um *gateway* Parlay/OSA.

5 – O *gateway* Parlay/OSA interage com um SMS-C, para requisitar o envio da mensagem de SMS através de uma rede específica capaz de prover acesso ao dispositivo móvel do assinante.

6- A rede de telecomunicações entrega a mensagem ao dispositivo correto.

Como visto neste exemplo, o serviço de envio de mensagens de textos curtos foi incorporado numa aplicação, que passa a ser interessante do ponto de vista de um assinante de uma rede de telecomunicações. Neste caso, a aplicação contém um valor agregado, além de apenas usar SMS, que é a capacidade de obter dados de uma fonte de informações sobre o clima e distribui-los aos interessados, corretamente.

3.4.5.2 Third Party Call

Na API Parlay X existe a especificação chamada *Third Party Call*[5], que pode ser vista no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a), como já comentado anteriormente nesta dissertação. As operações definidas na *interface ThirdPartyCall*, deste documento, são:

- *makeCall* : permite que uma aplicação de terceiros requisite um estabelecimento de chamada entre 2 terminais.
- *cancelCall*: permite cancelar a requisição feita através da operação *makeCall*. Não surte efeito se os terminais já estiverem conectados, conforme requisição com *makeCall.*
- *getCallInformation:* permite obter a informação de *status* em relação à requisição feita com *makeCall.* Exemplos de status: *ligação conectada*, *ligação iniciada* e *ligação terminada*.
- *endCall:* permite encerrar uma ligação que esteja com *status* de iniciada ou conectada.

Para invocar o método *makeCall,* é necessário fornecer 2 parâmetros obrigatórios a ele:

- callingParty : especifica o número do terminal considerado chamador.

---

[5] *Third Party Call*, ou chamada de terceiro, significa a requisição imposta a uma rede de telecomunicações (ex: rede de operadora de telefonia), com o objetivo de solicitar o estabelecimento de uma conexão telefônica entre dois terminais fixos e/ou móveis.

- calledParty: especifica o número do terminal considerado chamado.

Quando o método *makeCall* é invocado, a aplicação que requisita o serviço recebe uma *string* como um identificador da chamada a ser estabelecida com a ajuda do *gateway*. Assim, se a aplicação necessitar cancelar a requisição feita com o *makeCall,* bastará invocar o método *cancelCall,* passando como parâmetro obrigatório a mesma *string*. A *string* que identifica a requisição de estabelecimento de chamada é utilizada também, como parâmetro, nos métodos *getCallInformation* e *endCall.*

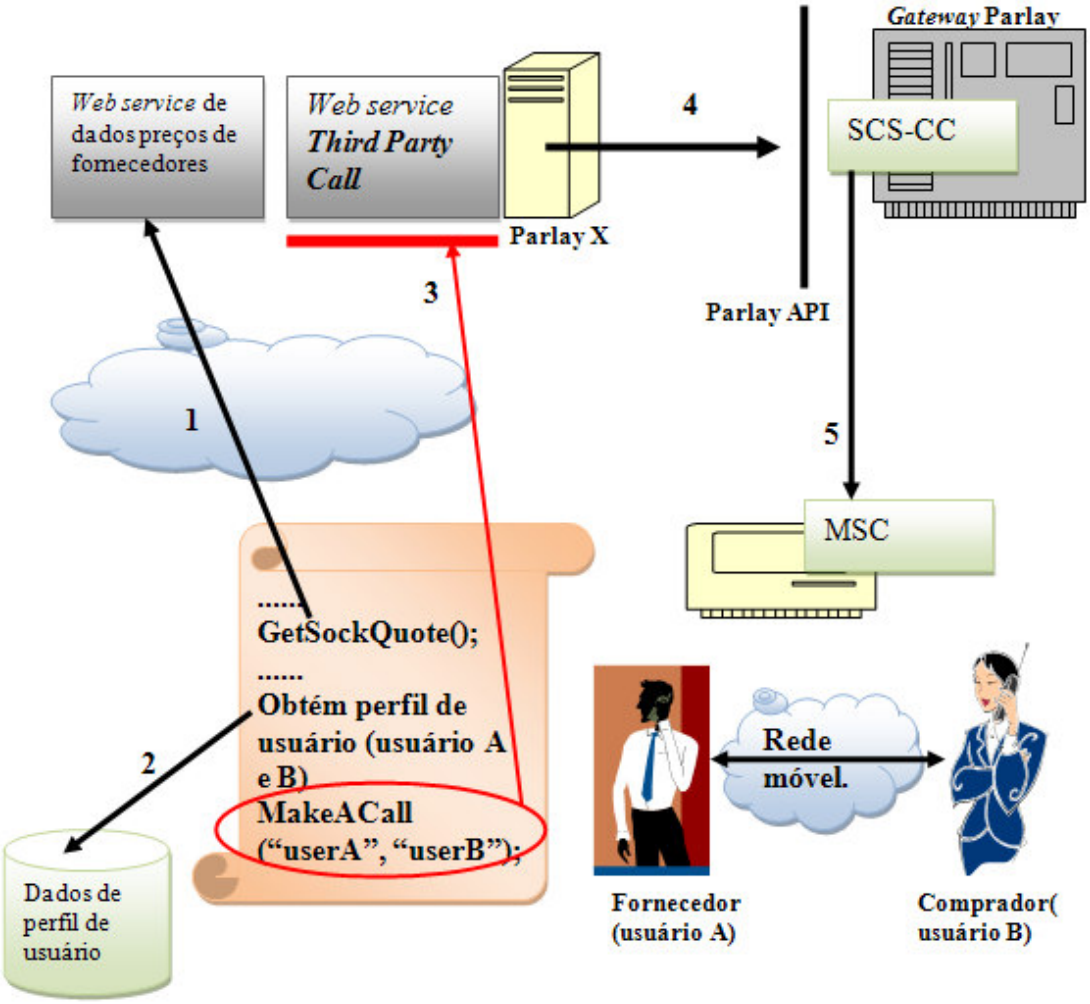

Figura 24 - *Cenário de uso da API Third Party Call.*

A **Figura 24,** retirada do documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a), mostra uma situação em que:

1 – Existe uma aplicação responsável em obter orçamentos de materiais necessários a alguns clientes de um determinado conjunto fornecedores. A obtenção dos dados sobre orçamentos é feita com consultas, através da tecnologia *Web Services*. Neste caso, está suposto que os fornecedores dos materiais necessários contêm, no mínimo, um *web service* capaz de responder consultas sobre preços.

2 – Depois que um preço é obtido, se o valor está abaixo de uma margem limite desejada pelo cliente, a aplicação obtém os dados do fornecedor e do cliente, como o número do telefone de cada um.

3 – A aplicação interage com o *gateway* Parlay X e invoca a execução do método *makeCall,* através do objeto que implementa a *interface ThirdPartyCall*. O objeto *ThirdPartyCall* está no domínio do *gateway* e, por exemplo, o algoritmo do método *makeCall* não interessa à aplicação.

4 - O *gateway* Parlay X se comunica com um *gateway* Parlay/OSA e solicita o estabelecimento de uma chamada entre os terminais *callingParty* e *calledParty*. Embora mostrado acima, não é obrigatório que o *gateway* Parlay X sempre seja um cliente de um *gateway* OSA/Parlay. Na verdade, estes *gateways* podem ser combinados entre si, como comentado na seção 3.4.6 adiante. Isto também vale para a **Figura 23**. Contudo, quando um *gateway* Parlay/OSA está disponível, isto torna mais fácil a implementação do *gateway* Parlay X, porque as interações com protocolos de rede já estarão programadas.

5 – O *gateway* Parlay/OSA providencia o estabelecimento da chamada.

O que ocorre nos itens 4 e 5 não precisa ser conhecido em detalhes pelo desenvolvedor desta aplicação de valor agregado.

Um terceiro parâmetro, não obrigatório, para o método *makeCall* é uma informação sobre cobrança, caso seja necessário que o *gateway* providencie uma mensagem de cobrança aos assinantes do serviço, por exemplo.

O seguinte códgo XML é um exemplo de uma parte do arquivo WSDL chamado *parlayx_third_party_call_interface_2_3.wsdl.*

```
<xsd:element name="makeCall"
type="parlayx_third_party_call_local_xsd:makeCall"/>
   <xsd:complexType name="makeCall">
     <xsd:sequence>
       <xsd:element name="callingParty" type="xsd:anyURI"/>
       <xsd:element name="calledParty" type="xsd:anyURI"/>
       <xsd:element name="charging"
                                 type="parlayx_common_xsd:ChargingInformation"
        minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
```

Este arquivo está disponível no *web site* do Parlay, juntamente com o documento texto que especifica a *interface ThirdPartyCall*. Através desse código, o desenvolvedor pode perceber que o método *makeCall,* realmente, espera 3 parâmetros, mas o parâmetro *charging* pode ter uma ocorrência igual a zero, no momento em que o método é invocado. O código XML anterior é um mapeamento em WSDL advindo da especificação da *interface ThirdPartyCall*, vista em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a), a qual contém a seguinte tabela:

Tabela 2 - *Parâmetros para o método makeCall da interface ThirdPartyCall.*

Nome	Tipo	Opcional	Descrição
callingParty	xsd:anyURI	não	Número do telefone da parte chamadora.
calledParty	xsd:anyURI	não	Número do telefone da parte chamada
charging	common:ChargingInformation	sim	Informações relacionadas com cobrança, se necessário aplicar.

O desenvolvedor de uma aplicação, que usa *gateway* Parlay X não é obrigado a ler os arquivos WSDL, mesmo que o XML seja inteligível. Basta que ele estude os documentos texto das especificações. Além disso, atualmente existem ferramentas de *software* capazes de interpretar os arquivos WSDL e mapeá-los em código Java, por exemplo. A partir de tais códigos mapeados, o desenvolvedor pode iniciar o seu

trabalho, para criar a aplicação de valor agregado. A empresa SUN disponibiliza ferramentas para este propósito. Nesta dissertação, será visto um exemplo de como utilizar tais ferramentas, para a implementação de uma nova capacidade a ser incluída num emulador de *gateway* Parlay X da empresa Ericsson. Esta inclusão de uma capacidade num emulador de *gateway* Parlay X é uma das contribuições desta dissertação, como ainda será demonstrado neste documento, no capítulo 5. Com esta contribuição, pode-se ver então como lidar com Parlay X, observando os conceitos relacionados com as tecnologias envolvidas em tal trabalho.

Conforme estas informações e o evento 3 descrito na **Figura 24**, o código XML, abaixo, representa a mensagem SOAP enviada sobre o HTTP:

```
<?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>userA</ns1:callingParty>
            <ns1:calledParty>userB</ns1:calledParty>
        </ns1:makeCall>
    </soapenv:Body>
</soapenv:Envelope>
```

O código XML, abaixo, representa a mensagem SOAP enviada sobre o HTTP, mas, desta vez, do *gateway* para a aplicação, o que não está demonstrado na **Figura 24**.

```
<?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:makeCallResponse>
            <ns1:result>0</ns1:result>
        </ns1:makeCallResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Na verdade, sempre que o método *makeCall* é invocado, ele retorna um identificador da requisição feita, como comentado antes. Este identificador, segundo o XML acima, vale 0 (zero). Nas próximas requisições feitas ao método *makeCall* , durante

uma mesma sessão de comunicação com o *web service ThirdPartyCall,* os identificadores retornados poderão ser 1,2,3,4 e assim por diante. Mas, a *string* a ser retornada é dependente da implementação do *gateway*. Não há regras para definir o valor da mesma.

Aqui, um dado entendimento importante deve estar fixado:
Um arquivo WSDL, como o *parlayx_third_party_call_interface_2_3.wsdl,* especifica uma *interface*, o que já foi dito. Isto significa que uma aplicação, cliente de *gateway*, utiliza esta especificação WSDL para ter condição de gerar requisições corretas a serem feitas no objeto, com esta *interface,* encontrado no *gateway*. Isto é, a aplicação cliente precisa saber como se comunicar com o objeto alvo. Por outro lado, o mesmo arquivo WSDL, quando convertido em código fonte, como será comentado mais adiante, gera uma *interface*, por exemplo em Java, que depois é implementada, para a formação proposital do objeto servidor (alvo da aplicação cliente). Portanto, a aplicação cliente e o *gateway* com o objeto servidor compartilham os mesmos arquivos WSDL.

3.4.6 Instâncias de G*ateways*

A arquitetura de um *gateway parlay* pode ser construída com qualquer combinação de Parlay X ou Parlay/OSA. Ou seja, pode-se ter um *gateway* Parlay X como cliente de um *gateway* Parlay/OSA. Ou pode-se ter um *gateway* Parlay X que interage diretamente com os recursos de rede, ou pode-se ter um conjunto de *web services* definidos com *interfaces* de Parlay/OSA e Parlay X. A decisão de como a arquitetura de um *gateway* Parlay deve ser implementada não depende de qualquer regra padronizada na indústria de telecomunicações e varia entre os *gateways* disponíveis atualmente. *Gateways* Parlay X e Parlay/OSA também podem ser combinados, juntamente com outras tecnologias, oferecendo um ambiente de estudos e testes. Ou seja, alguns *gateways* podem estar disponíveis para que sejam usados

respondendo requisições de protótipos de aplicações Parlay implementadas por profissionais da indústria de telecomunicações/TI, ou aplicações de estudantes, etc.
O Instituto Fraunhofer para Sistemas de Comunicação Abertos (FOKUS), fundado há mais de 20 anos na Alemanha, criou um centro de competência em desenvolvimento de aplicações para NGN. Mais precisamente, é um centro de competência em desenvolvimento de serviços móveis e aplicações com tecnologia 3G. Naquele centro, há um ambiente de desenvolvimento chamado *The FOKUS Open OSA/Parlay Playground*. De acordo com o artigo (MAGEDANZ, et al., 2004), o *OSA/Parlay playground* traz diferentes *gateways parlay*, simuladores de *gateways*, kits de ferramentas de criação de serviços e também aplicações de demonstração, tudo num único ambiente, disponível em http://www.fokus.fraunhofer.de/research/ngni. O objetivo daquele *playground* é estabelecer um centro aberto de educação e amostras em OSA/Parlay, para tornar esta tecnologia de APIs disponível para todos os *players* do mercado aberto de telecomunicações futuro. Por exemplo, desenvolvedores de aplicações podem obter experiência no uso da API, com o propósito de investigar os benefícios disto para seus sistemas de *software*. Institutos acadêmicos também podem interagir com estas tecnologias e, então, obter conhecimento base sobre o assunto. Operadoras de rede podem ver a tecnologia em funcionamento e os vendedores de produtos podem obter retorno dos desenvolvedores etc.

Conforme o artigo (MAGEDANZ, et al., 2004), o "coração" daquele *playgroud* forma o *FOKUS Open Communication Server (OCS)*, mostrado na **Figura 25** retirada de (MAGEDANZ, et al., 2004), o qual possibilita a implementação de aplicações de valor agregado sobre redes integradas de circuitos comutados e de pacotes, trazendo consigo APIs Parlay e Parlay X.

Como ilustrado na **Figura 25**, uma aplicação Parlay, feita em Java ou C++, terá as opções de envio de requisições a um *gateway* Parlay X, via SOAP, ou envio de requisições diretamente a um *gateway* Parlay/OSA.

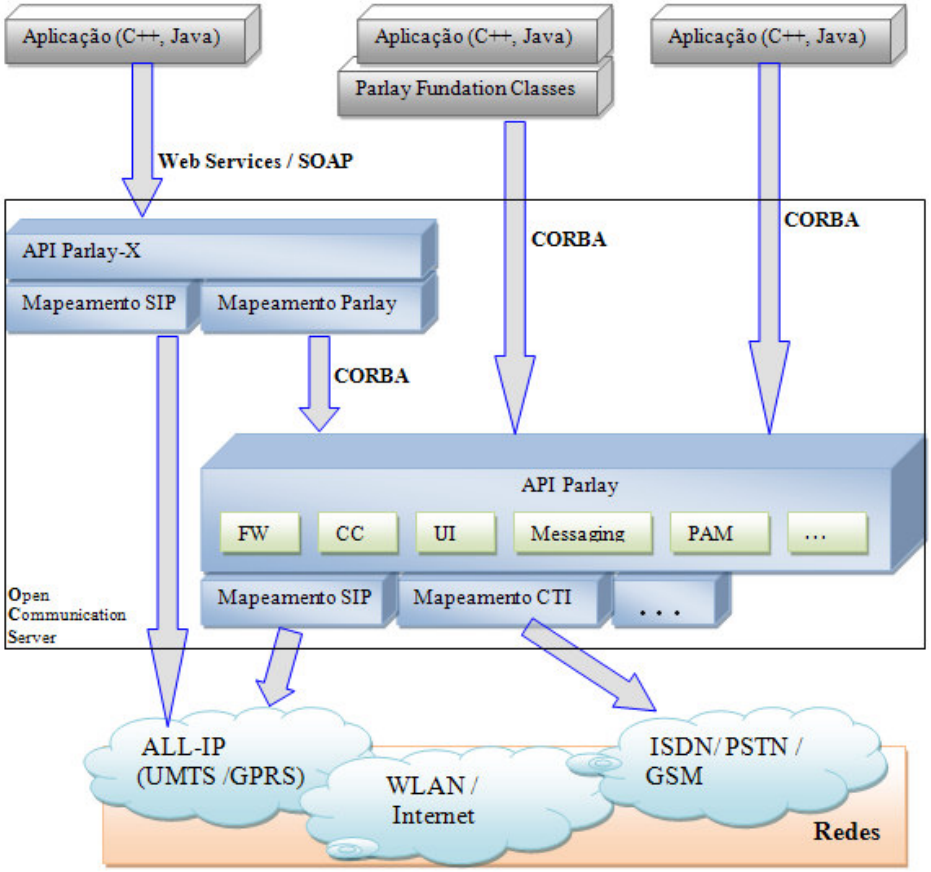

Figura 25 – *Open Communication Server @ FOKUS.*

A implementação de cada *gateway* pode prover um mapeamento às mensagens do protocolo SIP. Por exemplo, um serviço Parlay de controle de chamadas poderá utilizar as capacidades do protocolo SIP, como comentado em (TAKATAMA & TANI, 2001), quando da necessidade de estabelecer uma ligação entre terminais utilizando a sinalização implementada por este protocolo. Para tal, uma pilha SIP deve ser usada e, neste caso, por exemplo, pode-se usar uma pilha que implemente as *interfaces* da API JAIN-SIP. Portanto, mais uma vez, a pilha SIP do NIST pode ser considerada. Mas, estas tecnologias de telecomunicações que dão suporte à implementação dos *gateways* Parlay estão fora do escopo desta dissertação e, conseqüentemente, não foram alvo de estudo. O que deve ser notado é que já existem esforços para a divulgação das tecnologias de *interfaces* para a abertura das redes de telecomunicações, numa forma didática. Neste intuito, existem instâncias de *gateways* disponíveis para aqueles que estão iniciando seus estudos sobre como oferecer aplicações de valor agregado à indústria de telecomunicações.

Como exemplo de instâncias de *gateways parlay*, pode-se ter também emuladores de *gateways*. Um exemplo de emulador é fornecido pela empresa Ericsson, o qual ainda encontra-se em fase de implementação e se chama **Telecom Web Services Network Emulator**. Esse é um emulador de *gateway* Parlay X.

Além do ambiente didático do Fraunhofer e do emulador da Ericsson, podem ser citados também outros artefatos relacionados com Parlay, para utilização em estudos, implementações e testes, disponíveis pelo *International Institute of Technology* (IIT) em Montreal/Canadá.

Se a indústria de telecomunicações precisa que os profissionais de TI desenvolvam aplicações com valor agregado, usando recursos de redes das operadoras, então é de se esperar que as operadoras criem facilidades para os testes e implantações de tais aplicações. Uma facilidade para os testes das aplicações de terceiros é o fornecimento de acesso aos *gateways* reais, que usam recursos reais de uma operadora. Por exemplo, a Brasil Telecom também irá tornar disponível em sua rede um *gateway* Parlay X, para que desenvolvedores externos tenham condições de testarem suas aplicações *Parlay*. Esta afirmação pode ser comprovada com a mensagem eletrônica enviada pelo Atendimento Virtual daquela empresa, respondendo à pergunta sobre a intenção de realmente tornar este tipo de *gateway* disponível, como pode ser visto no Apêndice 1. Portanto, o conhecimento sobre Parlay X e sobre como criar aplicações clientes para um *gateway* deste tipo, poderá se tornar útil aos desenvolvedores, externos ou internos, que criarão aplicações de valor agregado para aquela empresa.

3.5 Sobreposição de Funcionalidades

Considerando todas as tecnologias comentadas até aqui e o relacionamento entre elas, percebe-se que a execução de uma aplicação, cliente de um *gateway* Parlay e/ou Parlay X, envolve a sobreposição destas tecnologias. Isto significa que uma tecnologia T1 depende de outra T2, sendo que T2 é de mais baixo nível, se comparada com T1. Para exemplificar, a requisição de envio de mensagem de texto curto, a partir de uma aplicação cliente, pode envolver as seguintes funcionalidades de tecnologias sobrepostas:

a) O sistema da aplicação cliente, possivelmente feito com a tecnologia Java, organiza uma mensagem, com o texto curto a ser enviada para um terminal móvel. Em tal aplicação, talvez haja uma função intitulada *enviaSMS*. O máximo que esta função poderá fazer é organizar os dados considerados, como o número do telefone do destinatário e o texto da mensagem, além de usar alguma tecnologia que suportará o envio de SMS. O algoritmo de tal função representa a funcionalidade de envio de SMS na forma mais abstrata possível, de mais alto nível, em termos de complexidade computacional. Este nível mais alto de programação é adequado aos desenvolvedores de *software* acostumados com, por exemplo, Programação Orientada por Objetos (RUMBAUGH et al., 1997). Esta função, *enviaSMS*, pode ser cliente, por exemplo, da tecnologia Web Services.

b) Se a tecnologia Web Services é utilizada para o envio de uma mensagem de texto curto, através de um *gateway* Parlay, isto não implica que haja um algoritmo, no respectivo *web service*, que repita as instruções da função *enviaSMS* presente um nível mais acima na aplicação cliente. Na verdade, as funcionalidades do *web service* estarão provendo os recursos indisponíveis no nível da aplicação cliente, como, por exemplo, acesso direto a algum recurso de uma rede de telecomunicações. Então, podem existir funções de SMS na aplicação cliente e também no *gateway*, mas com sentido de complemento, não de repetição de tarefas. O *web service*, contido num *gateway* Parlay X, que necessita enviar uma mensagem de texto curto, poderá utilizar um

serviço de SMS implementado num *gateway* Parlay/OSA. Isto também não significa que a funcionalidade de SMS na tecnologia Parlay X esteja repetindo a funcionalidade de SMS na tecnologia Parlay/OSA. Neste caso, o *gateway* Parlay X estará representando uma ligação entre a aplicação cliente e o *gateway* Parlay/OSA. Assim, a aplicação cliente não precisará lidar com a complexidade de um *gateway* Parlay/OSA. Então, as funcionalidades da tecnologia de SMS, em cada uma das entidades (aplicação cliente e os *gateways* Parlay X e Parlay/OSA) desenvolverão papéis distintos. O *gateway* Parlay/OSA, por sua vez, deverá interagir com elementos de rede específicos para a execução de SMS.

Tanto a API Parlay X, quanto a API Parlay/OSA declaram capacidades para SMS e outros serviços. Como apenas declaram, não definem os algoritmos para os serviços. Então, só haverá uma sobreposição de funcionalidades, pelas tecnologias sobrepostas, no sentido de repetição de tarefas, caso haja um planejamento errôneo dos algoritmos que implementarão as respectivas *interfaces*. Por exemplo, se tanto o algoritmo para a função *enviaSMS*, da aplicação cliente, quanto os algoritmos das funções de envio de SMS, nos *gateways* Parlay X e Parlay/OSA, verificarem a quantidade de caracteres presentes no texto a ser enviado pela rede, isso caracterizar-se-á como uma indesejada sobreposição de funcionalidades repetidas. Portanto, é importante notar que a sobreposição das funcionalidades das tecnologias implica em complementação de tarefas computacionais, o que é uma conseqüência da arquitetura de organização de tecnologias sobrepostas, como visto na **Figura 1**.

Esta sobreposição de tecnologias, dependendo do tipo de aplicação cliente envolvida, pode se tornar uma desvantagem. Porque, quanto mais tecnologias estão sobrepostas, maior é a quantidade de diferentes sistemas envolvidos na execução de uma requisição à rede de telecomunicações. Envolver vários sistemas durante a execução de uma requisição pode implicar num tempo demandado maior que o aceitável pela aplicação cliente. Por exemplo, o uso de SMS pode envolver execução de Máquina Virtual Java, servidor *web*, serviço na *web*, transmissão de mensagens XML pela rede, interpretador de mensagem SOAP, autenticação com *Framework* Parlay, etc.

Se a aplicação cliente tem a função de acionar uma máquina qualquer, em tempo real, enviando mensagem (SMS) para a mesma, quando ocorre algum evento específico, poderá ser apropriado que haja um acesso direto entre a aplicação cliente e a capacidade da rede de enviar mensagem, sem, então, a necessidade de executar outros sistemas, como servidores *web*, etc. Mas estas questões relacionadas com a *performance* resultante da sobreposição das tecnologias, comentadas aqui, estão além do escopo deste trabalho.

3.6 Análise em Resumo

Durante algum tempo, o IETS e o 3GPP estiveram trabalhando, independentemente um do outro, para a criação da padronização das *interfaces* a serem usadas na abertura das redes de próxima geração. Contudo, estes dois trabalhos isolados foram alinhavados juntamente com o esforço do consórcio Parlay, para a criação de uma única API padrão, conhecida como API Parlay. A API Parlay contem *interfaces* que descrevem várias capacidades das NGNs, inclusive a *interface ThirdPartyCall*, a qual foi a escolhida para ser implementada com o propósito de contribuição a um *gateway* específico, como explicado no penúltimo capítulo deste documento. Portanto, este capítulo detalha o que são e para que servem as *interfaces* Parlay, objetivando deixar claro quais são as funções da *interface ThirdPartyCall*. Na verdade, a *interface ThirdPartyCall* faz parte da API Parlay X, que é o conjunto mais simples das APIs Parlay. Como as *interfaces* da API Parlay X descrevem serviços disponívies via *web* ou *web services*, faz-se necessário o entendimento sobre o que são *web services*, como a comunicação entre eles pode ser realizada e quais são as tecnologias disponívies para tal. Então, vários detalhes sobre isso foram comentados neste capítulo. É necessário o bom entendimento destas tecnologias e como o uso delas deve ser organizado, para que os serviços de telecomunicações, via *web*, sejam compreendidos corretamente. Uma das entidades resultantes desta organização é o *gateway* Parlay X, que deve ser implantando numa rede de uma operadora, como foi dito. Portanto, este capítulo dá as informações necessárias para que seja bem

entendida a função de um *gateway* deste tipo e para que seja bem entendido, posteriormente, como uma inclusão de uma nova funcionalidade pode ser feita no mesmo. Um bom exemplo de gateway Parlay X disponível no mercado de telecomunicações é provido pela empresa Ericsson, o qual será explicado no próximo capítulo. E após a leitura do próximo capítulo, serão notáveis as formas de contribuir com a implementação de tal *gateway*. Mas, antes disso, torna-se realmente necessário o conhecimento dos detalhes sobre as tecnologias explicadas aqui, como o protoloco SOAP para a comunicação com XML entre serviços na *web*. Somente vendo os detalhes neste capítulo é que se pode perceber a complexidade envolvida na construção de um *gateway* Parlay X e o que deve ser esperado, em termos de funcionamento correto, ao planejar a inclução de alguma nova funcionalidade nessa entidade computacional. Sabendo sobre a existência de todos estes detalhes comentados, pode-se ater atenção nos pontos mais importantes descritos no próximo capítulo, que está focado no emulador de *gateway* Parlay X, alvo da contribuição explicada neste trabalho.

3.7 Trabalhos Relacionados

Em março de 2008, Sedlar et. al. (SEDLAR et. al., 2008) propuseram o uso de Parlay X e *Third Party Call*, dentre outras tecnologias, para a construção de uma nova aplicação para IP *Television* (IPTV[6]). Basicamente, tal aplicação possibilita uma ligação telefônica entre o terminal (móvel ou fixo) de quem vê um programa de televisão via IPTV e o terminal (fixo ou móvel) da instituição que esteja fazendo propaganda durante o mesmo programa. Ou ainda, permite que o telespectador ligue para algum telefone, por exemplo, sem que seja necessário discar qualquer número a partir de um terminal real. O que ocorre é que a aplicação proposta interage com um *gateway* Parlay X e invoca uma ação da i*nterface ThirdPartyCall*, de tal forma que o *gateway* se comunica com os recursos de uma operadora de telefonia e, então, a ligação é estabelecida. Este *gateway* foi implementado com Java. E a aplicação pode

[6] Maiores informações sobre IPTV podem ser encontradas em http://www.itu.int/ITU-T/IPTV/ .

enviar requisições a ele através da tecnologia Web Services, a qual está explicada na Seção 3.3. A tecnologia Web Services foi escolhida devido a sua facilidade de implementação e compatibilidade com diferentes plataformas, segundo Sedlar (SEDLAR et. al., 2008). O trabalho feito nesta dissertação também utiliza as tecnologias Web Services, Parlay X e programação de *software* em Java, mas para outra aplicação. Contudo, ambas as aplicações são semelhantes a tal ponto que usaram a mesma *interface* da API Parlay X (*ThirdPartyCall*). Isto demonstra a versatilidade do uso destas tecnologias para a criação de serviços de valor agregado em Telecomunicações.

Vale salientar que, se Sedlar não tivesse decido usar Parlay e *Third Party Call,* o seu sistema teria que se comunicar diretamente com entidades que implementam tecnologias como SIP ou INAP, para o controle de chamadas, o que seria mais complexo.

Em (AJAM, 2008), Nabil Najam propõe acrescentar a um *gateway* Parlay X um novo *web service*[7] voltado para a privacidade dos usuários. Para avaliar a solução, uma implementação foi feita em Java utilizando o Servidor de Aplicações da Sun, versão 9.0, o qual é um *web server*. Tal trabalho trata-se de uma entidade real composta por *web services,* chamada *Location Platform Emulator* (LPE), que é um emulador de *gateway* Parlay X acrescido do novo *web service* proposto. Esta implementação simula o uso de um *gateway* Parlay X no contexto de serviços de localização e contém um conjunto de bases de dados distribuídos pela rede, cada qual simulando um nó real. Ou seja, dados de terminais simulados, com suas localizações específicas, foram armazenados em banco de dados. O trabalho apresentado nesta dissertação também usa o mesmo Servidor de Aplicações da Sun para suportar o emulador de *gateway* da Ericsson que foi modificado, como mostrado no Capítulo 5. Ao contrário de (AJAM, 2008) não foi desenvolvido um novo emulador de *gateway*

---

[7] Um *web service* é um serviço computacional disponível, por exemplo, na Internet, através do suporte da tecnologia Web Services.

Parlay X, para testar a nova *interface* implementada, já que seria mais proveitoso contribuir com o trabalho da Ericsson. Além disso, não foi usado banco de dados para representar dados de terminais simulados. Os terminais simulados, com o trabalho desta dissetação, só existem enquanto o emulador em questão está em execução no servidor *web* da Sun. Como será visto, não houve necessidade para se usar um banco de dados.

No Brasil também existem trabalhos relacionados com as APIs Parlay. As instituições IBM, Ericsson, Brasil Telecom e CPqD detêm experiências com este assunto. Por exemplo, a Ericsson já trabalhou no desenvolvimento de uma aplicação de localização de restaurantes, como comentado na mensagem vista no Apêndice 4, e O CPqD já desenvolveu aplicações Parlay para operadoras nacionais.

Levando em consideração todo o trabalho realizado durante esta dissertação de mestrado e considerando também as referências levantadas, não foram encontrados outros exemplos de trabalhos relacionados. Ou seja, o trabalho de implementação da *interface ThirdPartyCall* não pode ser visto em outros trabalhos até então. Isto é verdade devido ao fato que as tecnologias *web services* e Parlay X ainda são novas na área de Telecomunicações. Mesmo a API Parlay X ainda vem sendo desenvolvida e melhorada a cada *release*. Outros trabalhos envolvendo a *interface ThirdPartyCall* ainda estão por vir.

4. Emulador de *Gateway* Parlay X da Ericsson

4.1 Programa *Ericsson Mobility World* para Desenvolvedores

A empresa sueca Ericsson criou um programa abrangendo ações, eventos, documentos e ferramentas, com o propósito de fornecer aos profissionais de TI o suporte necessário ao desenvolvimento de aplicações inovadoras para a indústria de telecomunicações, que serão bem aceitas pelos usuários e que trarão mais lucros às empresas relacionadas com tal indústria. Este programa se chama *Ericsson Mobility World*. Os desenvolvedores de TI que se tornam membros deste programa, e isto é possível a qualquer pessoa mediante inscrição gratuita no *web site* daquela empresa, recebem notificações sobre locais e datas de eventos que reúnem grandes empresas da indústria de telecomunicações interessadas em novas aplicações com valor agregado, por exemplo. Os documentos e ferramentas constituem *kits* para a implementação das aplicações inovadoras que demandarão recursos de telecomunicações. O programa *Ericsson Mobility World* também oferece suporte adequado para testes das novas aplicações criadas, divulgação das mesmas e facilidades para instalação na indústria para, finalmente, gerar lucros, como esperado. Com a ajuda do programa *Ericsson Mobility World* um desenvolvedor de aplicação poderá iniciar o seu trabalho usando os seus conhecimentos prévios em Java, ou seja, não será necessário desenvolver destreza em telecomunicações, por exemplo. Os membros do programa *Ericsson Mobility World* também podem participar de fóruns de discussões, via Internet, sobre assuntos de desenvolvimentos

das aplicações por meio das ferramentas oferecidas. Estas ferramentas facilitam a criação das aplicações que dependerão de tecnologias tais como: IMS, Parlay/OSA, ***Telecom Web Services***, etc. Este suporte oferecido conta principalmente com o uso da infra-estrutura, real ou emulada, residente na rede de uma operadora de telefonia móvel, o que provê um enriquecimento das capacidades das aplicações inovadoras. As capacidades que um desenvolvedor pode usar, neste ambiente, são: *Mobile Positioning,* cobrança, MMS, localização, SMS e outras.

4.2 Web Services para Telecomunicações

Os *web services* na área de telecomunicações (*telecom web services*) são usados pelas aplicações enriquecidas com capacidades de telecomunicações, num cenário de integração *Business-to-business* (B2B).

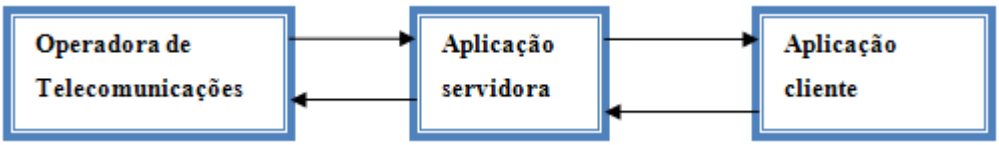

Figura 26 – *Aplicações envolvidas.*

As aplicações enriquecidas com capacidades de telecomunicações são aquelas que podem interagir com, por exemplo, uma operadora, para enviar requisições ou receber dados da mesma. Este tipo de aplicação, que foi citada neste documento várias vezes, foi chamada até aqui de aplicação cliente. Porque ela é realmente cliente dos serviços da operadora. Este tipo de aplicação está representado, por exemplo, nas figuras: **Figura 12, Figura 19, Figura 23, Figura 24** e **Figura 25**. Contudo, uma aplicação enriquecida com capacidades de telecomunicações, por exemplo, uma aplicação de envio de SMS, pode atender às requisições de várias outras aplicações que necessitam executar envios de SMS. Neste caso, a aplicação referenciada como cliente até aqui é, na verdade, citada como aplicação servidora, em materiais (guias, manuais, etc) que acompanham emuladores de *gateways*, por exemplo. Esta aplicação servidora está representada na **Figura 26**, no retângulo

central. Este tipo de aplicação não precisa estar, necessariamente, hospedada em alguma máquina do domínio da operadora. Pode estar em outra máquina, mas com acesso à Internet e capaz de se conectar a um provedor de *telecom web service* tal como uma operadora de telefonia móvel.

Na **Figura 23**, a aplicação responsável por coletar informações de meteorologia e enviá-las aos terminais móveis se comporta como uma aplicação servidora, do ponto de vista dos usuários com seus terminais ou em relação ao *software* cliente existente em tais terminais. Estes usuários são os clientes da aplicação comentada aqui e devem, provavelmente através da mesma, cadastrarem seus números de telefones móveis, para que possam receber as informações. Neste cenário, esta aplicação pode ser um sistema localizado num *web server,* contendo um conjunto de páginas *web* e, através do protocolo SOAP, pode manter relacionamentos B2B com o servidor de dados de meteorologia e também com um *gateway* Parlay da operadora de telefonia móvel, para uso de serviços de comunicações, como o SMS. Neste exemplo, a aplicação servidora necessita de um *gateway* Parlay, que pode ser um *gateway* Parlay X constituído por um grupo de *telecom web services*. Dentre os *web services*, a capacidade de usar SMS deve estar disponível, como notado. Portanto, para o desenvolvimento da aplicação sugerida na **Figura 23,** no mínimo durante a fase de testes, o desenvolvedor poderá utilizar uma das ferramentas disponíveis no programa *Ericsson Mobility World,* para verificar se a sua aplicação irá funcionar como esperado ao ser instalada num provedor de aplicações efetivamente usado no mercado. Tal ferramenta da Ericsson é o *Telecom Web Services Network Emulator*, como citado anteriormente.

Com este emulador pode-se construir testes para a aplicação da **Figura 23**, mas não para aquela demonstrada na **Figura 24**. O fato é que, até o momento em que este documento foi escrito, tal emulador ainda não apresentava um serviço como definido pela API *Third Party Call* do Parlay X. É ai que está uma das contribuições deste trabalho: uma proposta de implementação de tal serviço no emulador de *gateway* da

Ericsson. Assim, como será visto, este emulador também poderá prover a capacidade de aceitar requisições de terceiros para estabelecimento de chamadas entre terminais móveis.

4.3 Características Funcionais do Emulador Atual

O emulador de *gateway* Parlay X da Ericsson, o *Telecom Web Services Network Emulator*, é um sistema de *software* que pode ser obtido diretamente no *web site* da Ericsson, na página *web* (ERICSSON, 2008a), via *download*, para ser compilado, instalado e executado num computador pessoal. Este emulador foi construído usando a edição *Entreprise* de Java - *Java Platform, Enterprise Edition* (Java EE) e até este momento encontra-se na versão 3.0, que foi liberada à comunidade de desenvolvedores membros do programa *Ericsson Mobility World* e interessados, no dia 31 de março de 2008.

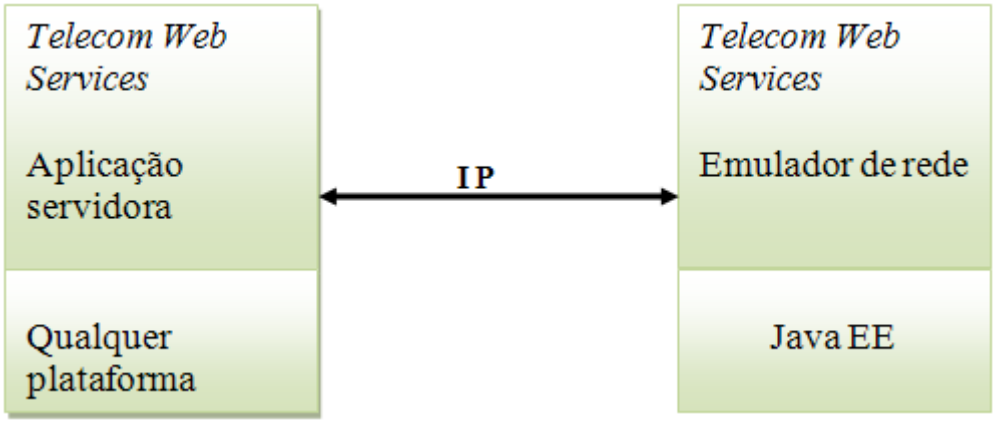

Figura 27 - *Emulador interagindo com uma aplicação.*

A **Figura 27**, copiada do Guia do Desenvolvedor (ERICSSON, 2008b), mostra o emulador interagindo com uma aplicação servidora, que pode ter sido desenvolvida em qualquer plataforma. Esta aplicação pode se comunicar com o emulador, desde que, por exemplo, utilize o SOAP sobre HTTP, o que pode ser feito numa rede com protocolo IP mais *Web Services*.

Como este emulador é constituído de um conjunto de *web services*, o mesmo deve ser executado sobre um servidor *web (web server)*. Dois *web servers* válidos para a execução do emulador são:

- Apache Tomcat 6;
- *Sun Java System Application Server* (SJSAS).

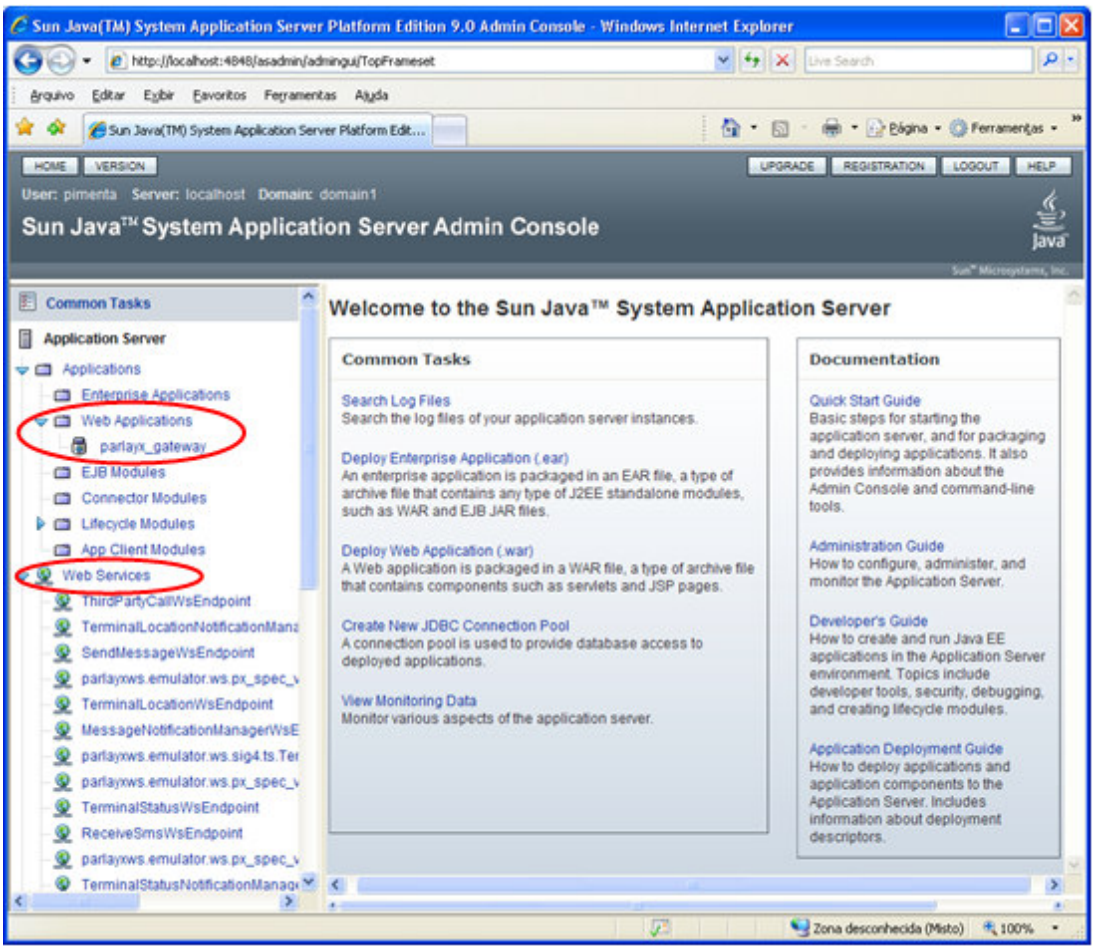

Figura 28 - *Console de administração do SJSAS.*

O Tomcat 6 é o servidor *web* mais atual da Fundação Apache (*Apache Software Foundation*) e suporta tecnologias baseadas em Java, como a *JavaServer Pages* (JSP). O SJSAS é um servidor de aplicações, construído pela empresa SUN e permite a execução e gerenciamento de aplicações feitas com a tecnologia Java EE. Por exemplo, o SJSAS tem funções para monitorar a execução das aplicações, bem como torná-las ativas no servidor.

A **Figura 28** representa o console de administração do SJSAS, o qual pode ser acessado via um navegador como o Internet Explorer ou Fire Fox, no endereço http://localhost:4848/. Pode ser visto que o console mostra quais aplicações *web* estão ativas, bem como os *web services*. O SJSAS "escuta" as requisições do HTTP na porta 8080.

Depois que o emulador do *gateway* é compilado, o resultado da compilação gera um arquivo chamado *parlayx_gateway.war*, o qual contém os *web services* programados e pode ser empregado diretamente no SJSAS. Um arquivo chamado *build.xml* acompanha o código do emulador e este arquivo contém o *script* necessário para a compilação e geração do arquivo *war*. O *script* deste arquivo pode ser executado pelo *software* Apache *ANT*[8].

Quando um emulador de *gateway* Parlay X já está empregado no SJSAS (aplicação *do arquivo war*) e ativado para receber requisições de serviços de telecomunicações, os seguintes eventos podem ocorrer:

a. Por exemplo, uma requisição de estabelecimento de chamada entre terminais é enviada ao *gateway* emulado, a partir de uma aplicação em fase de testes. Os números dos terminais devem ser passados em tal requisição.
b. O SJSAS recebe esta requisição via HTTP, na porta 8080, contendo uma mensagem do protocolo SOAP. Neste caso, a aplicação sendo testada pode enviar a sua requisição ao seguinte endereço: **http://localhost:8080/parlayx_*gateway*/ThirdPartyCallService**. Deve-se usar o endereço de IP ou domínio da máquina onde estiver instalado o SJSAS, nesta URL. O nome *ThirdPartyCallService* poderia ser qualquer outro nome. Na verdade, tal nome deve estar definido num arquivo XML

[8] O software Apache Ant é uma ferramenta construída pela Fundação Apache e baseada em Java. A sua função é parecida com, por exemplo, a função da ferramenta *make* do Unix. Neste caso, usa-se a ferramenta ANT para a interpretação de *scripts* em XML, os quais podem descrever ações de compilações de códigos fontes. A vantagem da ferramenta ANT é o fato de ser extensível sem depender de detalhes de sistema operacional, já que pode ser entendida a partir de novas classes em Java. Mais detalhes: http://ant.apache.org/ .

como será explicado mais adiante. Este nome é apenas uma convenção de identificação do serviço relacionado com *Third Party Call*.

c. O SJSAS entrega a requisição a certos objetos apropriados, que podem lidar com SOAP.
d. Através desta requisição os objetos determinam que um serviço é requisitado ao objeto que implementa a *interface ThirdPartyCall* da API Parlay X e, pelo conteúdo da mensagem SOAP, pode-se determinar que o método a ser invocado em tal objeto é o *makeCall(...)*.
e. Ocorre uma chamada ao método *makeCall(...)* passando os parâmetros obrigatórios obtidos da mensagem SOAP.
f. O objeto *ThirdPartyCall* executa o método *makeCall(...)* seguindo o algoritmo definido pelo desenvolvedor do *web service* respectivo no emulador. No caso de um *gateway* Parlay X real, o algoritmo deste método necessita enviar requisições reais a uma rede, por exemplo, de uma operadora de telefonia móvel, com o propósito de estabelecer a ligação real entre os terminais móveis reais.
g. Os terminais emulados, que fazem parte do emulador, simulam uma ligação entre si, num *status* de ligação conectada.
h. Certos objetos no SJSAS preparam uma mensagem SOAP contendo uma *string*, como um identificador da chamada requisitada, e enviam-na à aplicação. É o método *makeCall()* que formula o valor desta *string* a ser devolvida à aplicação.

Vale lembrar que o emulador de *gateway* Parlay X da Ericsson, versão 3.0, não contém a capacidade de *Third Party Call* e esta foi inserida como contribuição original deste trabalho.

4.3.1 Funções do Emulador

O emulador de *gateway* Parlay X da Ericsson também contém uma *interface* gráfica vista através do navegador Internet Explorer ou Firefox. Esta *interface* gráfica permite a criação e gerenciamento de terminais móveis também emulados. Assim, se um terminal é criado com o número 99112633, por exemplo, então uma aplicação que interage com o emulador poderá, se necessário, enviar uma mensagem, via SMS, para o número 99112633. Uma *interface* gráfica que representa o terminal emulado irá mostrar o recebimento de tal mensagem, confirmando que a aplicação está implementada corretamente e consegue, realmente, interagir com um *gateway* Parlay X para usar a capacidade de SMS.

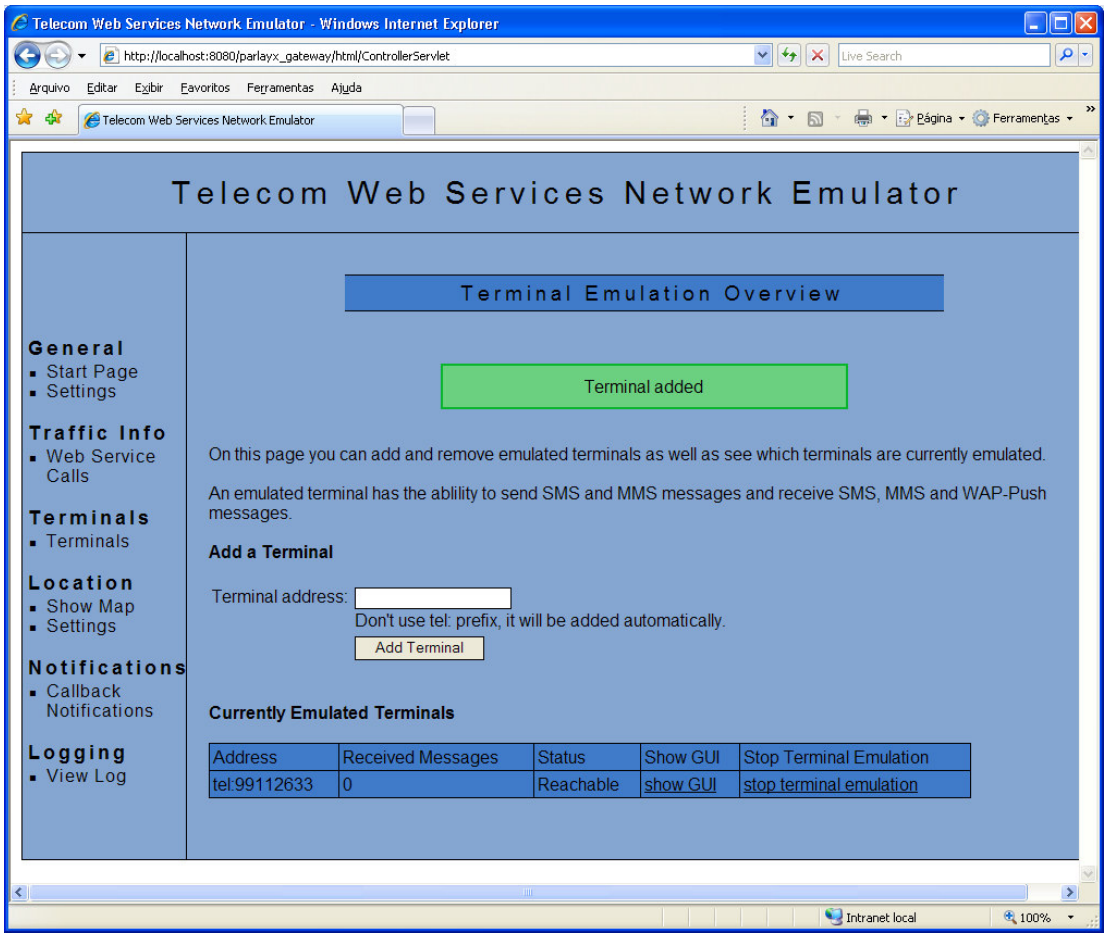

Figura 29 - *Interface gráfica para controle do emulador.*

A **Figura 29** mostra um instantâneo da *interface* gráfica para controle do emulador, quando um terminal móvel, cujo número é igual a 99112633, foi adicionado. Neste caso, o emulador passa a considerar que existe um terminal, com tal número. Para um *gateway* real, esta situação é como a existência de um terminal 99112633 real, alcançável pela rede de telecomunicações sob o mesmo *gateway*.

O *link* <u>*show GUI*</u> mostrado na *interface* representada pela **Figura 29**, quando seguido, mostra a *interface* gráfica do terminal emulado e se tal terminal recebeu uma mensagem via SMS, por exemplo, de um terminal com número 8312285. A **Figura 30** mostra a *interface* deste caso. O número 8312285 pode estar representando o telefone da pessoa que requisitou, à aplicação servidora, o envio da mensagem. Neste caso, a aplicação servidora requisita ao *gateway* o serviço SMS, passando os número 99112633 e 8312285 como parâmetros.

O botão *Off*, demonstrado pela **Figura 30**, quando acionado, simula um desligamento do terminal, após o que, se a aplicação servidora tentar enviar uma mensagem via SMS para o mesmo terminal, ocorrerá uma exceção. O botão *Make a call,* quando acionado, faz o terminal emulado simular a situação de ocupado, tanto que se a aplicação servidora requisitar o *status* deste mesmo terminal, ela receberá o valor *busy*. A **Figura 31** representa o terminal com *status Busy*.

Além das funções para emular terminais e seus *status,* o emulador de *gateway* Parlay X da Ericsson também apresenta outras utilidades, dentre elas:

- função para configuração do tipo de exceção (*exception*) que deve ser retornada e o momento de retorno, para uma requisição de serviço.
- função para demonstrar dados sobre as requisições recebidas, na *interface* gráfica vista no navegador, como na **Figura 32**.

- função para configurar as coordenadas geográficas simuladas para os terminais emulados, sobre a imagem de um mapa também visível na *interface* do navegador. Útil para testes de aplicações requisitando localização de terminais.

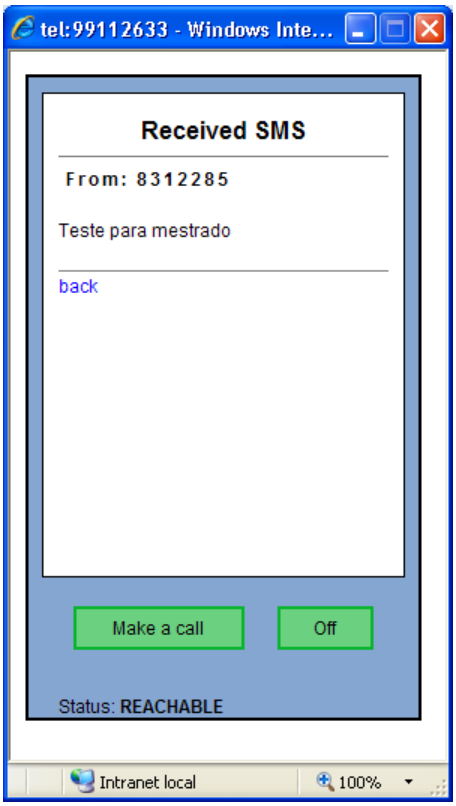

Figura 30 - *Terminal emulado recebeu SMS.*

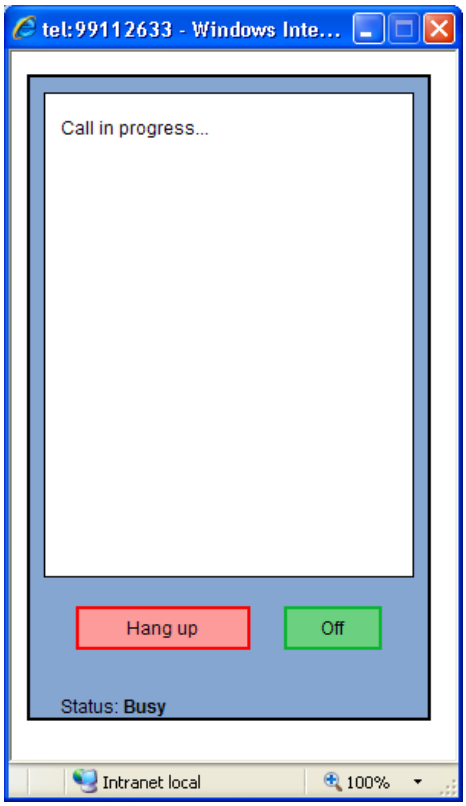

Figura 31 - *Terminal emulado ocupado.*

Service	Source	Timestamp	Parameters
SendSms	Sent from Emulator	2008-07-13 11:09:06	**Request** To addresses: [tel:99112633] From: 8312285 Message: Teste para mestrado **Response - OK** Request ID: 0

Figura 32 - *Dados da requisição de envio de SMS.*

Até o momento em que este documento foi escrito, o emulador podia ser manipulado apenas para 4 tipos de casos de usos: Envio de mensagem com SMS, com MMS, Requisição de status de terminal e Requisição de localização de terminal. O capítulo seguinte comenta sobre a adição de uma nova funcionalidade ao emulador, para trabalhar também com o caso de uso de chamadas estabelecidas entre terminais

requisitadas pelas aplicações servidoras de terceiros o que, em termos de funcionalidades macros do sistema, corresponderá a uma em cinco, ou seja, um quinto em relação ao todo.

4.4 Análise em Resumo

Já era sabido, conforme os capítulos ateriores, que um *gateway* Parlay real, implementando *interfaces* padronizadas, poderia ser usado para responder às requisições a serviços de telecomunicações, feitas por aplicações de *software* com valor agregado. Mas, agora, através deste capítulo, sabe-se também que um emulador de *gateway* pode ser perfeitamente utilizado para fins didáticos, o que consiste ajuda àqueles que desejam projetar, implementar e testar as aplicações de valor agregado, sem necessidade de interações com *gateways* reais. De fato, a própria empresa Ericsson promove ações para a divulgação e utilização de um emulador de *gateway* Parlay X, de sua autoria, como visto aqui. Como foi este o *gateway* escolhido, para receber uma nova funcionalide, este capítulo é fundamental neste trabalho, para introduzir os detalhes deste elemento de rede emulado, já que o mesmo é mostrado com suas funcionalidades, além de uma explicação sobre que ações ocorreriam em tal *gateway*, ao atender uma requisição ao serviço *Third Party Call*, se este estivesse já implementado no *gateway*. Finalmente, com a leitura destes quatro capítulos anteriores, pode-se entender as explicações do capítulo seguinte, onde se concentra o principal assunto deste trabalho.

5. Contribuição para o Emulador de *Gateway* Parlay X da Ericsson

Após a análise e experimentação do emulador citado no capítulo anterior, foi decidido contribuir com este projeto da Ericsson, através da adição de mais uma utilidade ao sistema emulado. A utilidade adicionada é um serviço *web* que implementa a *interface ThirdPartyCall* da API Parlay X.

5.1 Motivação

Quando uma pessoa precisa obter um orçamento, para depois comprar um produto, ela pode telefonar para o vendedor e indagá-lo sobre os preços. Este é um exemplo de uma situação comum, quando a pessoa que origina a chamada decide sobre o momento de telefonar, porque ela sabe que precisa falar com alguém e sabe qual é o momento de buscar a informação. Ou seja, a chamada telefônica só ocorre se o chamador estiver inclinado a realizá-la. Por exemplo, se uma pessoa precisa construir uma casa, então ela certamente irá telefonar para várias lojas de materiais de construções e pesquisar os preços praticados sobre produtos necessários. Alternativamente, pode-se construir um sistema computacional responsável em manipular os dados de vendedores (nomes, endereços e preços) e os dados de clientes em potencial (nomes, produtos demandados, limite de orçamento), de tal forma que, ao existir um vendedor com produtos viáveis a um cliente, esse sistema seja capaz de criar uma ligação telefônica entre as duas partes interessadas. Dessa forma, a ligação pode ocorrer sem mesmo alguém ter que decidir o momento para

realizá-la. Isto também evita a necessidade de uma pessoa ligar em várias lojas, para pesquisar preços, neste exemplo. Esta funcionalidade computacional seria um conforto a mais para as pessoas cadastradas em tal sistema. É claro que este é apenas um único exemplo, mas pertinente para despertar o entendimento da grande utilidade provida pela capacidade de uma operadora aceitar requisições terceiras pra estabelecimento de ligações. Sendo assim, tornar-se-á viável a existência de um emulador de *gateway* Parlay X com a capacidade *Third Party Call*.

Além disso, decidiu-se por esta *interface* porque as suas funções são de fácil entendimento e tal *interface* era a menor, dentre aquelas ainda não implementadas neste emulador, o que tornaria esta contribuição a menor possível, em termos de tamanho, mas não de menor importância. Assumir a contribuição de mais fácil compreensão foi uma estratégia para se aprender a lidar com o assunto Parlay X na prática, a partir de uma complexidade plausível para quem deseja iniciar seus entendimentos neste campo.

5.2 Visão geral do trabalho de implementação da *interface* alvo

A especificação completa da *interface ThirdPartyCall* encontra-se no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a) e a nova contribuição para o emulador foi baseada neste documento, além do código fonte que estava disponível dentre os arquivos desse sistema, obedecendo à mesma arquitetura do *software*. Este trabalho também foi baseado no guia (ERICSSON, 2008b), e em algumas dicas obtidas no fórum de discussões disponível através do programa *Ericsson Mobility World.* Algumas dúvidas foram esclarecidas sobre como organizar o ambiente computacional necessário à implementação da nova capacidade para o emulador, com a ajuda de um grupo de discussões moderado pela Ericsson da Suécia e presente no *web site* daquela empresa.

Para a implementação do *web service ThirdPartyCall,* no emulador, foi necessária a obtenção e compilação dos arquivos WSDL correspondentes, em primeiro lugar. Com a compilação dos arquivos WSDL, novos arquivos em código Java foram gerados, automaticamente, contendo a *interface* descrita na API Parlay X *ThirdPartyCall*. Em seguida, foi criada uma classe em Java que implementa todos os métodos desta *interface,* contendo então os algoritmos necessários à execução das lógicas dos métodos *makeCall(...), endCall(...), getCallInformation(...)* e *cancelCall(...).* Várias outras classes Java também foram implementadas, para dar suporte às necessidades do serviço emulado, mas nem todas serão discutidas neste documento, para o mesmo não ficar demasiadamente extenso. Com todas estas classes prontas, o controle do emulador, em relação ao novo serviço *ThirdPartyCall,* ficou pronto e, depois disso, alguns arquivos já existentes, relacionados com as *interfaces* gráficas dos terminais emulados, foram modificados para a simulação dos terminais, permitindo a execução de arquivos de sons que imitam campainhas de telefones celulares e/ou fixos, além de outros detalhes. Assim, agora, os terminais também simulam situações onde são chamados e há um novo botão nos mesmos, para o aceite de uma requisição de chamada. O trabalho prático final foi a modificação de alguns arquivos XML, com *script* de compilação do código do emulador e com descrições necessárias ao servidor *web* que irá portar o novo conjunto de *web services* Parlay X.

Com a conclusão deste trabalho, um novo arquivo *parlayx_gateway.war* foi gerado e enviado à Ericsson da Suécia, para os responsáveis do emulador, que analisaram a nova funcionalidade *ThirdPartyCall* e comentaram que a implementação feita está adequada (ver Apêndice 3).

5.3 Compilação dos Arquivos WSDL

Os arquivos WSDL relacionados com o serviço *ThirdPartyCall* podem ser obtidos no *web site* do Parlay, juntamente com a API respectiva. São 2 arquivos:

1. *parlayx_third_party_call_interface_2_3.wsdl e*
2. *parlayx_third_party_call_service_2_3.wsdl.*

O primeiro arquivo define a *interface ThirdPartyCall*, conforme a especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a). É a partir deste arquivo que uma *interface* Java chamada *ThirdPartyCall* pode ser criada. Portanto, este arquivo WSDL define como deve ser a assinatura das funções da *interface* em questão, quando codificadas numa linguagem de programação de *software*. Tanto o emulador, quanto a aplicação servidora, devem conhecer este mesmo arquivo WSDL, para que ambos considerem a mesma *interface ThirdPartyCall,* ao tratar uma requisição a um dos seus métodos. Ou seja, a aplicação servidora saberá como invocar um método desta *interface*, utilizando o nome correto do mesmo, bem como seus parâmetros necessários, na ordem correta em que devem ser fornecidos. Assim, a *interface* se passa como um contrato entre a aplicação e o emulador. Já o segundo arquivo descreve informações relacionadas com o protocolo SOAP e a *interface* definida no primeiro arquivo. Este segundo arquivo define que, no *web server,* haverá um serviço chamado *third_party_call_service* e que o mesmo conterá, por exemplo, uma operação chamada *makeCall*. Este segundo arquivo é necessário ao emulador, que deve esperar por requisições a este serviço, mas não é necessário à aplicação cliente. Conforme o arquivo *parlayx_third_party_call_service_2_3.wsdl*, quando uma ação for requisitada ao serviço, o objeto implementando a *interface* definida no primeiro arquivo será o responsável pelo trabalho solicitado. Isto se passa como uma regra a ser obedecida pelo novo *web service* a ser construído no *web server*. Além destes 2 arquivos, existe também o arquivo:

parlayx_third_party_call_types_2_3.xsd.

Este último arquivo define alguns tipos abstratos de dados, usados pelo serviço *ThirdPartyCall*, como : *CallInformation, CallStatus e CallTerminationCause*.

Para compilar os arquivos WSDL e *XML Schema* (XSD), visando a geração automática dos arquivos Java necessários, foi utilizado um computador pessoal com *Java Platform, Enterprise Edition*, versão 5 com *Java Development Kit* (JDK) *update* 2, instalado. Esta versão do Java EE, com JDK, contém a *Java API for XML Web Services* (JAX-WS). A JAX-WS é uma API para a construção de *web services* e clientes destes serviços, contendo, dentre outras coisas, a ferramenta *wsimport*. A ferramenta *wsimport* lê arquivos WSDL e XSD e gera todos os artefatos requeridos para o desenvolvimento, emprego e uso de um *web service.* Portanto, esta ferramenta *wsimport* foi utilizada para mapear os arquivos WSDL e XSD no código Java necessário. A API JAX-WS foi criada pela empresa SUN.

Para o uso da ferramenta *wsimport,* foi necessário um *script* XML, como segue:

```
<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>
```

Este código XML foi atribuído ao código original, no arquivo *build.xml*, que faz parte do emulador da Ericsson. O arquivo *2.* nomeado anteriormente faz referência ao arquivo *1.* e esse último ao arquivo *3.* Portanto, os 3 arquivos são considerados pelo *wsimport*. O arquivo *build.xml* pode ser executado pela ferramenta ANT. Quando este XML é interpretado e executado pela ferramenta ANT, ocorre a

compilação desejada dos arquivos WSDL e XSD, gerando, automaticamente, os arquivos Java.

Todo o desenvolvimento da nova parte do emulador, o serviço *ThirdPartyCall* (TPC), foi feito inteiramente com o uso do *software* Eclipse SDK[9], versão 3.3.2. O Eclipse contém uma *interface* gráfica provendo acesso a várias funções úteis à codificação do serviço TPC, compilação, testes, etc. Através do Eclipse, a ferramenta ANT pode ser invocada facilmente, para o uso do arquivo *build.xml*.

Ao mapear os arquivos WSDL e XSD, relacionados com TPC, 19 entidades Java (*classes*, *interfaces* e estrutura de dados) foram geradas automaticamente, distribuídas em 3 pacotes Java. Pela *interface* gráfica do Eclipse podem ser vistos os 19 arquivos respectivos, como na **Figura 33**.

Estes arquivos estão indicados com um símbolo contendo a letra "J". A *interface* Java que foi implementada, para a criação de uma classe que dá funcionalidade aos métodos da API TPC, está no arquivo *ThirdPartyCall.java*. No pacote *org.csapi.schema.parlayx.third_party_call.v2_3. local* estão as classes que definem os objetos que contêm valores utilizados como atributos para a execução de métodos da *interface* TPC, ou valores retornados pela execução de alguns destes métodos. Por exemplo, uma instância da classe *MakeCall* pode ser usada para manter os valores dos parâmetros a serem passados no método *makeCall* da *interface* definida no arquivo *ThirdPartyCall.java*. Entretanto, mesmo que exista a classe *MakeCall* e as outras deste pacote, isto não significa que elas tenham que ser, obrigatoriamente, utilizadas no código a ser criado ainda.

---

[9] No final deste capítulo há um breve comentário sobre as ferramentas escolhidas para a formação do ambinente de desenvolvimento do software.

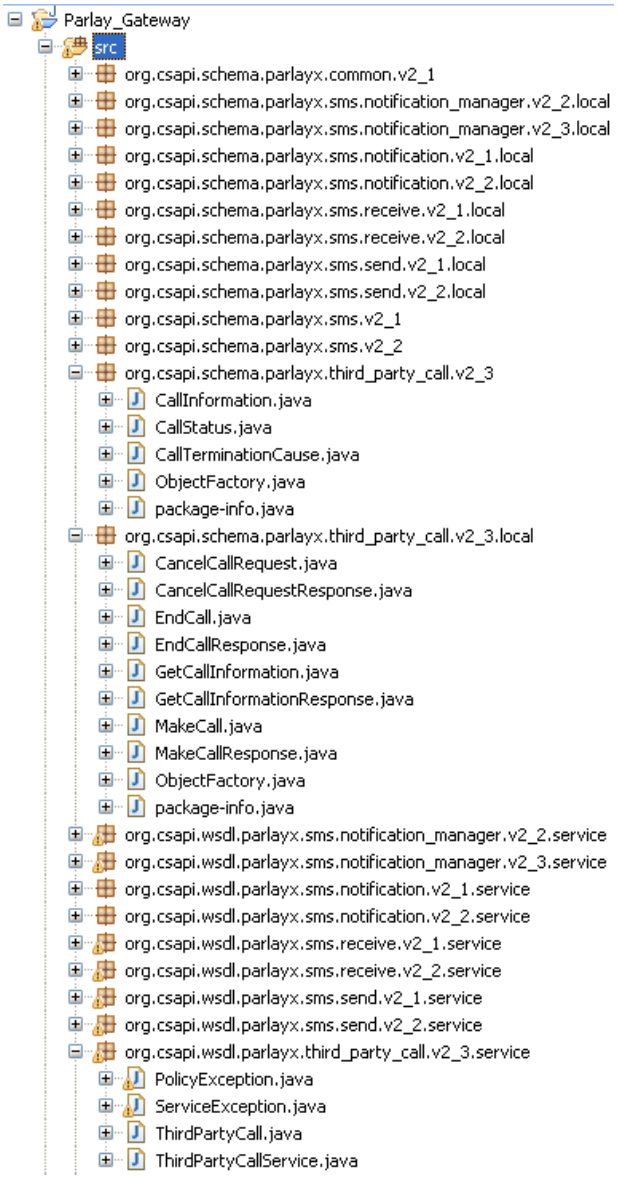

Figura 33- *Classes e interfaces geradas automaticamente.*

De fato, tais classes foram utilizadas somente entre algumas outras classes geradas automaticamente. Os arquivos *CallInformation.java, CallStatus.java* e *CallTerminationCause.java* do pacote *org.csapi.schema.parlayx.third_party_call .v2_3* definem as estruturas de dados para manter informações sobre o *status* de ligações requisitadas, sobre a causa do término de ligações, etc. Estas estruturas, por exemplo, são a tradução em Java do conteúdo XML do arquivo *parlayx_third_party_call_types_2_3.xsd.* Mais precisamente, como um exemplo, o conteúdo do arquivo *CallStatus.java* é o mapeamento em Java do seguinte código retirado do arquivo *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>
```

O citado fragmento XML é a definição da enumeração vista na **Tabela 3**, que está definida no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a).

Tabela 3 - *Enumeração de status de chamadas.*

Valor Enumerado	Descrição
CallInitial	A ligação está sendo estabelecida
CallConnected	A ligação está estabelecida (Ativa)
CallTerminated	A ligação foi encerrada

5.4 Implementação da *Interface ThirdPartyCall*

O diagrama UML de classes da **Figura 34** mostra algumas classes e *interfaces* que foram criadas relacionadas com a implementação da *interface ThirdPartyCall*. É a classe Java *ThirdPartyCallWsEndPoint* a responsável por, finalmente, definir os algoritmos das funcionalidades dos métodos desta *interface*. Portanto, neste emulador, a classe *ThirdPartyCallWsEndPoint* determina como o *gateway* irá trabalhar para funcionar conforme a especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a). Evidentemente, a classe *ThirdPartyCallWsEndPoint,* por si só, não implementa todo o código para o trabalho necessário a ser executado. Esta classe conta com várias outras classes provedoras de funções úteis ao propósito do serviço de TPC, o que deixa o sistema arquitetado em módulos. Mas, nem todas estas outras classes estão demonstradas e comentadas neste documento. Para um conhecimento detalhado do sistema, faz-se necessária a análise cuidadosa do código completo, implementado em Java, XML e Javascript.

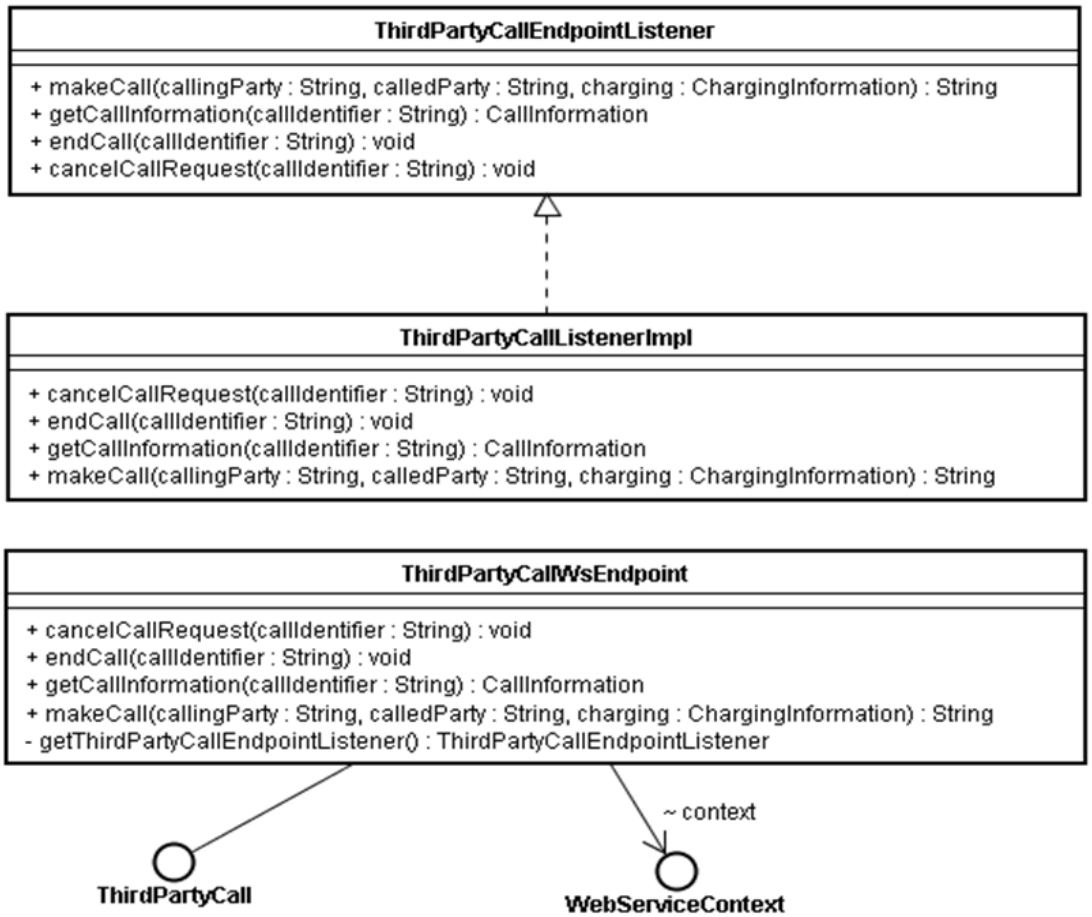

Figura 34 - *Diagrama de algumas classes para Third Party Call.*

Quando o SJSAS é executado, o mesmo, automaticamente, inicia a execução do emulador. Ou seja, ao executar o sistema SJSAS, o emulador passa a ser disponível num endereço IP, pronto para aceitar as requisições aos *web services*. Para isto, o SJSAS executa a classe *InitServlet,* que fica no arquivo *InitServlet.java*. Tal classe foi criada pela Ericsson, para a implementação do emulador e é a primeira classe do emulador a ser executada quando este inicializa. A classe *InitServlet* é responsável por inicializar outras classes necessárias à execução perfeita da classe *ThirdPartyCallWsEndPoint*.

Quando o emulador recebe uma requisição, por exemplo, "*makeCall*", o método *makeCall* da classe *ThirdPartyCallWsEndPoint* é executado e este, por sua vez,

necessita de um objeto que implemente a *interface ThirdPartyCallEndpointListener,* vista na **Figura 34**. Na verdade, sempre que o objeto *ThirdPartyCallWsEndPoint* tem um de seus métodos invocados, faz-se necessário o uso de uma nova instância de um objeto com a *interface ThirdPartyCallEndpointListener.* Portanto, uma nova instância com a *interface ThirdPartyCallEndpointListener* sempre será a responsável em atender às necessidades do objeto *ThirdPartyCallWsEndPoint.* O resultado disto é que sempre haverá uma única instância da classe *ThirdPartyCallWsEndPoint,* durante a execução do emulador, mas várias instâncias com a *interface ThirdPartyCallEndpointListener.* Assim, uma requisição de um serviço não afetará a execução de outra requisição, já que as mesmas podem ser servidas por instâncias independentes. Mas essas instâncias compartilham algumas estruturas em comum, como um *vector* Java para manter informações sobre as requisições realizadas. As informações podem ser, por exemplo, o *status* de uma ligação telefônica, motivo de término da ligação, etc. O objeto com a *interface ThirdPartyCallEndpointListener* é sempre uma instância da classe *ThirdPartyCallListenerImpl*, como visto na **Figura 34**. A classe *InitServlet* registra o nome "*ThirdPartyCallListenerImpl",* conforme a arquitetura do emulador, de tal forma que o objeto *ThirdPartyCallWsEndPoint* tem condição de obter uma instância da classe *ThirdPartyCallListenerImpl,* através do mesmo serviço de registro, mas este e outros detalhes de implementação não são interessantes neste documento.

Um objeto da classe *WebServiceContext*, que é usado por um objeto *ThirdPartyCallWsEndPoint,* tem a utilidade de prover acesso ao objeto *HttpServletRequest.* O objeto *HttpServletRequest* é criado pelo próprio *web server* e contém informações como o endereço IP de onde partiu a requisição a um *web service*. Essas informações podem ser demonstradas na *interface* gráfica do gerenciador do emulador, mostrada na **Figura 29**.

5.5 Inclusão de Novos Comportamentos aos Terminais Emulados

Com a contribuição realizada por esse trabalho, os terminais emulados no emulador de *gateway* Parlay X da Ericsson passaram a simular o recebimento de uma chamada telefônica, até mesmo tocando uma campainha real, sendo possível também o aceite ou a recusa das ligações. Como tal, os seguintes eventos ocorrem nesta ordem:

a. Uma aplicação requisita o estabelecimento de chamada entre dois terminais da rede de telecomunicações. Por exemplo, terminais X e Y.
b. O emulador recebe a requisição e, utilizando de seus objetos (alguns comentados na seção anterior), armazena as informações da requisição, como os números dos terminais relacionados neste caso.
c. Os terminais emulados consultam dados no emulador e descobrem sobre a requisição de chamada. Neste caso, se a requisição considera X como terminal chamador e Y como terminal chamado (parâmetros *CallingParty*=X e *CalledParty*=Y do método *makeCall()*, o terminal X irá tocar sua campainha, soando como um telefone celular, enquanto o terminal Y permanecerá inerte.
d. Quando um terminal soa a sua campainha, a sua *interface* gráfica se mostra ligeiramente diferente da original criada pela Ericsson, agora com um novo botão amarelo intitulado "*Accept call*". Veja a **Figura 35**.
e. Neste momento, um usuário poderá efetuar um *click* sobre o botão *Accept call*. Isso simulará um terminal real aceitando uma chamada de ligação, numa rede real. Com isto, a *interface,* vista na **Figura 35,** passa a ser demonstrada como aquela da **Figura 31**.
f. Em seguida o emulador registra a informação de que o terminal Y deve tocar sua campainha. Como no item c acima, o terminal Y descobre este fato e toca sua campainha, mas desta vez soando como um terminal fixo. As diferenças de sons de campainhas facilitam o acompanhamento, por parte do usuário, sobre qual terminal é o chamado e qual é o chamador. Mais uma vez, a *interface* gráfica, representando o terminal Y, se mostrará semelhante àquela da **Figura 35**.

g. O usuário decide se aceita a chamada para Y, ou não, através do botão *Accept call.* Se a ligação é aceita, os 2 terminais X e Y terão suas *interfaces* permanentemente mostradas como a **Figura 31**, a menos que o botão *Hung Up* receba um *click*. Estes dois terminais então simularão uma ligação telefônica estabelecida entre eles, como usuários destes terminais dialogando através dos mesmos.

Quando o botão *Accept call* recebe o *click* a campainha pára de soar, imediatamente, e o *status* do terminal emulado passa a ser *busy*.

Exemplo de mensagem SOAP enviada pela aplicação, no caso de terminal chamador = 99112633 e terminal chamado = 8312285:

```
<?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:99112633</ns1:callingParty>
            <ns1:calledParty>tel:8312285</ns1:calledParty>
        </ns1:makeCall>
    </soapenv:Body>
</soapenv:Envelope>
```

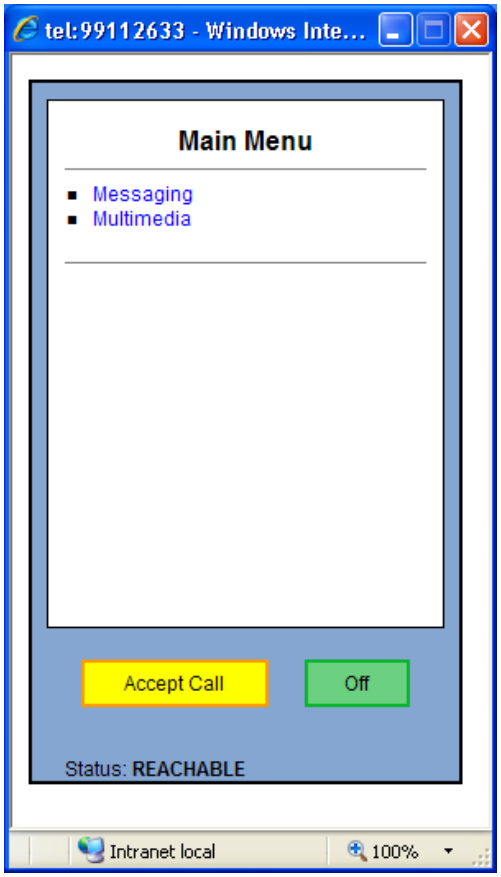

Figura 35 - *Terminal emulado simulando chamada.*

Se a opção de menu *web service calls,* mostrada na **Figura 29**, receber um *click,* a *interface* gráfica do gerenciador do emulador, visível pelo Internet Explorer ou Firefox, mostrará a informação vista na **Figura 36**.

Service	Source	Timestamp	Parameters
Make Call - Make Call - Informações no projeto de mestrado.	127.0.0.1:1542	2008-07-19 14:43:23	**Request** Calling Party: tel:99112633 Called Party: tel:8312285 Charging Information: null **Response - OK** Message ID: 0

Figura 36 - *Web service call information para makeCall.*

Além do método *makeCall()*, da *interface ThirdPartyCall*, o emulador também está apto a receber requisições para os outros métodos desta *interface* e se comporta exatamente como especificado em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a). Por exemplo,

depois que uma ligação é conectada, quando os terminais X e Y aceitam as chamadas, se uma aplicação enviar uma requisição ao método *getCallInformation()*, do *web service ThirdPartyCall*, ocorre o seguinte:

a. O *web server* recebe uma mensagem HTTP, contendo a seguinte mensagem SOAP:

```
<?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>
```

O identificador **0** é passado como parâmetro ao método *getCallInformation()*, o que identifica a requisição feita anteriormente considerando os terminais X e Y. A aplicação que invoca um *getCallInformation()* conhece o valor 0 porque o recebeu como um retorno do método *makeCall()*.

b. Após uma análise feita da mensagem SOAP, por parte de objetos capazes de interpretar este protocolo e disponíveis no *web server* pela JAX-WS, o método *getCallInformation()* do objeto *ThirdPartyCallWsEndPoint* é invocado.

c. O emulador pesquisa as informações relacionadas com a requisição *makeCall(X,Y,...)* e retorna o *status* correto, contido num objeto da classe *CallInformation.* O objeto da classe *CallInformation* contém um objeto da classe *CallStatus*, que contém a informação sobre o *status* na forma de uma *string*.

d. O objeto *CallInformation* retornado é interpretado por outros objetos Java, no *web* s*erver*, disponíveis pela JAX-WS e capazes de formular uma resposta SOAP com as informações obtidas de um *CallInformation.* Assim, o emulador envia uma resposta à aplicação, como visto abaixo:

```
<?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>
```

e. A aplicação que requisitou a informação interpreta a mensagem SOAP e obtém o *status Call Connected*.

Service	Source	Timestamp	Parameters
Get Call Information - Get Information - Informações no projeto de mestrado.	127.0.0.1:1779	2008-07-19 15:32:54	**Request** Call Identifier: 0 **Response - OK** Call Information - Duration: null Call Information - CallStatus: CALL_CONNECTED Call Information - TerminationCause: null Call Information - StartTime: null
Make Call - Make Call - Informações no projeto de mestrado.	127.0.0.1:1542	2008-07-19 14:43:23	**Request** Calling Party: tel:99112633 Called Party: tel:8312285 Charging Information: null **Response - OK** Message ID: 0

Figura 37 - *Web service call information.*

Após isso, o emulador pode mostrar as informações vistas na **Figura 37**, confirmando a requisição da aplicação e o status da ligação telefônica.

Segundo a especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a), os atributos *Duration, TerminationCause e StartTime* não são obrigatórios, i.e., o objeto da classe *CallInformation* não precisa conter valores para estes atributos, de forma obrigatória. Neste trabalho, optou-se por não atribuir valores a tais atributos.

5.5.1 Estados de Chamadas Conforme Comportamento dos Terminais

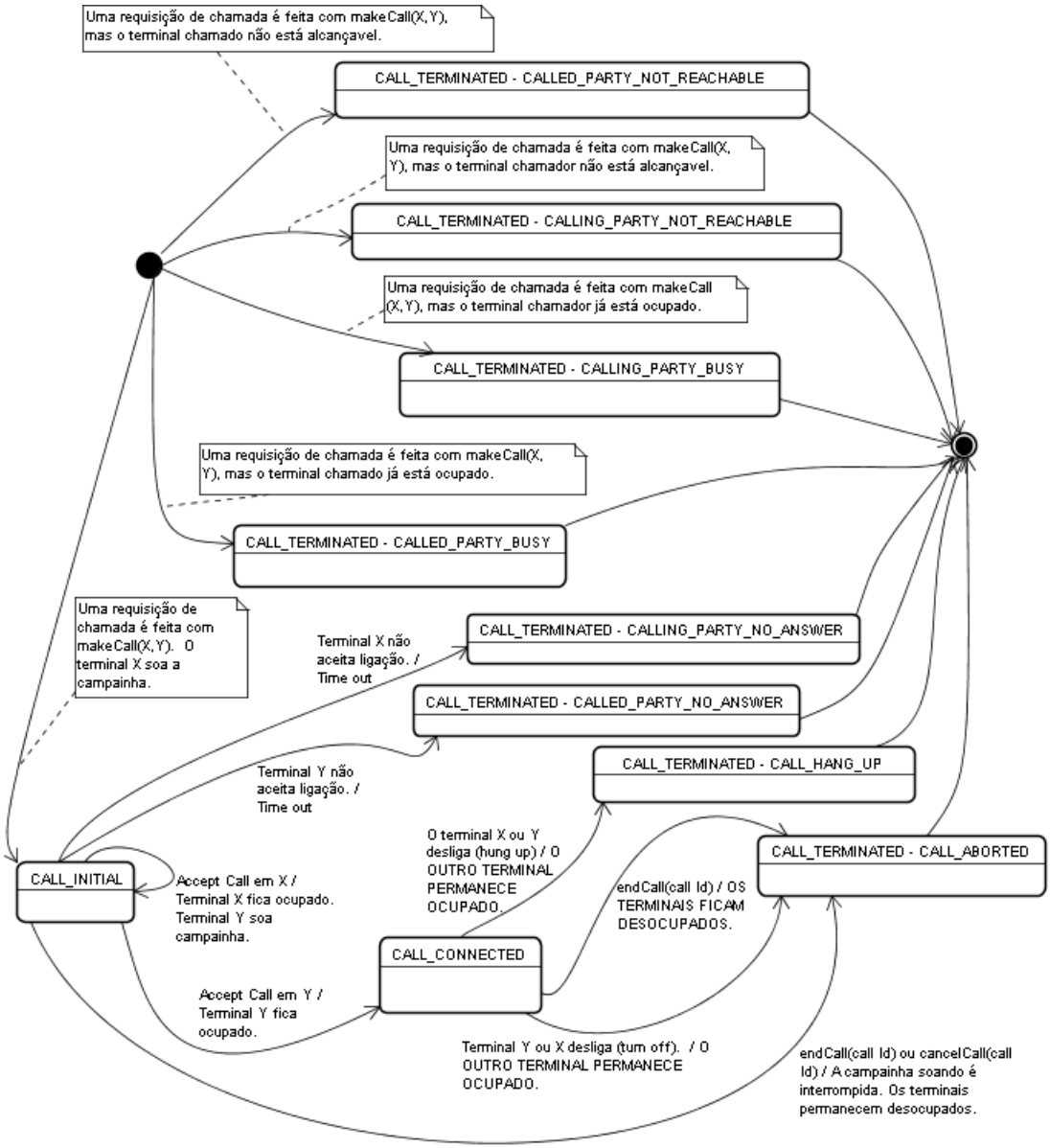

Figura 38 - *Diagrama de estados de pedido de ligação entre terminais.*

De acordo com todos os novos comportamentos criados para os terminais emulados, de acordo com requisições feitas ao *web service ThirdPartyCall* e conforme as reações do usuário interagindo com o emulador, uma chamada requisitada entre os terminais X e Y, chamador e chamado, respectivamente, pode assumir alguns estados definidos em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS

INSTITUTE & THE PARLAY GROUP, 2006a). O diagrama de estados da **Figura 38,** o qual não existe no documento (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a), mostra quais são os estados que uma ligação telefônica pode assumir e como isso pode ocorrer. Cada um destes estados pode ser obtido através da execução do método *getCallInformation(),* do *web service ThirdPartyCall*.

Todos os estados relacionados com uma ligação telefônica, como visto na **Figura 38**, exceto os estados *Call_Initial* e *Call_Connecte*d, transitam para o estado final, depois de um determinado tempo. O que significa que o tempo para reter a informação de *status* da ligação é limitado no emulador ou num *gateway* real. Tal informação deve expirar depois de certo tempo, segundo a especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a). Para este comportamento, foi escolhido o tempo de 40 segundos. Já o estado *Call_Initial* transita automaticamente para o estado *Call_Terminated* (se X ou Y não responde à chamada), após 20 segundos. A especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a) não impõe um valor definido para estes tempos.

Este diagrama de estados é uma conseqüência imediata das regras especificadas em (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a). Entretanto, alguns detalhes não estão especificados; portanto, a seguinte decisão foi tomada:

- Quando um dos terminais, que participa de uma chamada já conectada, decide desligar (*turn off* ou *hung up*), o outro terminal não volta para o estado de desocupado. Ele permanece ocupado até que seu botão *Hung up* receba um *click*. Para o bom entendimento desta decisão, os terminais podem ser considerados fixos. Neste caso, o segundo terminal, realmente, só estará desocupado se o seu usuário colocar o fone no gancho. Portanto, com esta decisão, o *status* do telefone passa a ser dependente do seu usuário, apropriadamente, e um *gateway* Parlay X pode,

realmente, ser um *gateway* para uma rede com terminais fixos. A especificação (THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE & THE PARLAY GROUP, 2006a) não tem esta regra definida.

Durante o estado final, representado na **Figura 38**, se uma requisição for feita ao método *getCalInformation(),* então a aplicação que requisita a informação receberá o seguinte dado:

"*Response Error. Information retention time out*".

5.5.2 Código Fonte Editado para o Comportamento dos Terminais

Para a inclusão dos novos comportamentos dos terminais, foi necessária a edição e/ou criação dos seguintes arquivos indicados na **Figura 39**:

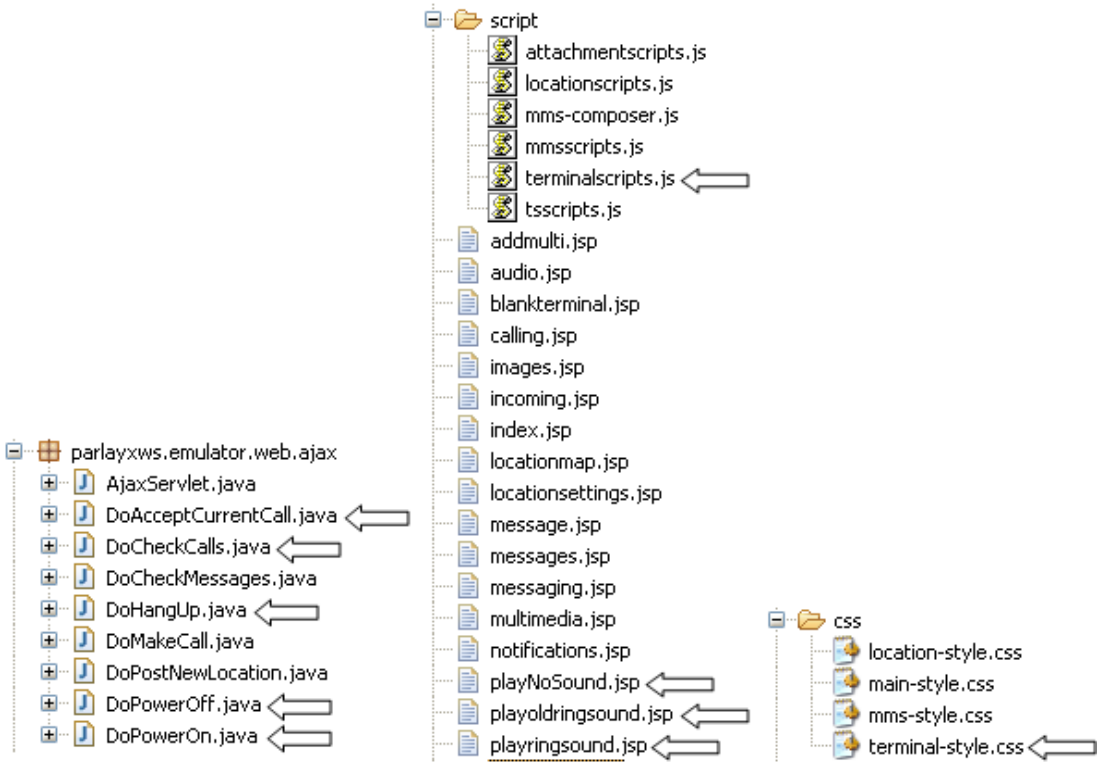

Figura 39 - *Arquivos relacionados com um terminal emulado.*

O arquivo *terminalscripts.js* contém código Javascript, que são *scripts* determinando alguns comportamentos dos terminais emulados. Por exemplo, neste arquivo, encontra-se o código que controla o tipo de informação que o terminal mostra (mensagem SMS, MMS, etc), dependendo da interação do usuário com a sua *interface* gráfica. Também há controle para o tipo de botão que será mostrado, se de *Make a call*, *Hung up* ou *Accept call*. Uma das modificações feitas neste arquivo foi a inclusão do seguinte código:

```
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();
}
```

Este código é executado quando o usuário efetua um *click* sobre o botão *Accept Call*. Neste ponto, precisamos da tecnologia *Asynchronous JavaScript* and XML (AJAX) (AJAX, 2008). Ela permite que este código use a classe *DoAcceptCurrentCall* de forma assíncrona. Ou seja, novos métodos incluídos no *script terminalscripts.js* utilizam classes do pacote *parlayxws.emulator.web.ajax*. A classe *DoAcceptCurrentCall,* por exemplo, tem a responsabilidade de configurar algumas informações sobre o respectivo terminal, como atribuir o *status busy* a ele, configurar seu atributo campainha como "desligado" e manter a informação de *Call Connected* junto às informações da chamada respectiva, se for o caso.

As classes do pacote *parlayxws.emulator.web.ajax* são classes de ações para o controle dos comportamentos dos terminais, cujos objetos recebem requisições diretamente do Javascript e podem interagir com outros objetos de outras classes do emulador. Estas classes são as servidoras das requisições com AJAX, vindas do Javascript e, mesmo que seus algoritmos sejam complexos, isto não afetará a *performance* da *interface* gráfica dos terminais emulados, devido à característica

assíncrona das requisições, graças ao AJAX. Os outros arquivos do pacote *parlayxws.emulator.web.ajax* cuidam de outros comportamentos dos terminais.

Alguns dos métodos do *script terminalscripts.js* utilizam os arquivos *playNoSound.jsp, playoldringsound.jsp* e *playringsound.jsp*. Estes arquivos, escritos em JSP são responsáveis pelo soar das campainhas dos terminais chamados ou chamadores. O arquivo *playNoSound.jsp* é o responsável em interromper a campainha de um terminal, quando este aceita uma chamada. O código deste arquivo é invocado pela função *playNoSound()* presente no arquivo *terminalscripts.js.*

O arquivo *terminal-style.css* contém código que define estilos visuais para componentes da *interface* gráfica dos terminais. Este arquivo foi modificado para conter o estilo do novo botão de aceite de chamada, botão *Accept call,* visto na **Figura 35**.

5.6 Edição dos Arquivos XML, para Aplicação do Emulador

Após a conclusão do código fonte necessário às novas funcionalidades do emulador, fez-se necessária a edição dos arquivos *build.xml, sun-jaxws.xml* e *web.xml*.

Os arquivos *web.xml* e *sun-jaxws.xml* contêm descrições necessárias à execução de uma aplicação para a *web*. Por exemplo, estes arquivos são utilizados para expor cada *web service* do emulador. Expor significa declarar quais são as classes que implementam *interfaces* de serviços, de tal forma que, se o *web server* receber uma requisição direcionada a um serviço definido por uma *interface* X, haverá uma chamada a um método no objeto da classe que implementa X. Tal chamada será

providenciada por algum outro objeto residente no *web server*, possivelmente definido por uma API como a JAX-WS.

No arquivo *sun-jaxws.xml* foi acrescentado o seguinte código:

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

Este código XML expõe a classe *ThirdPartyCallWsEndpoint*. Com esta codificação, o *web service ThirdPartyCall*, agora representado pelo objeto *ThirdPartyCallWsEndpoint,* pode ser visto também na **Figura 28**. Este objeto é o ponto final onde chega uma requisição de uma aplicação, por exemplo. Depois disto, a requisição é, finalmente, tratada pelo emulador. Daí vem o sufixo *Endpoint* usado no nome da classe. O desenvolvedor do emulador deve preocupar-se com a codificação desta classe que implementa a *interface ThirdPartyCall*, mas não deve preocupar-se com as outras classes responsáveis pela interpretação das mensagens SOAP, porque isto é responsabilidade da JAX-WS. Os objetos das classes do JAX-WS executam os métodos da classe *ThirdPartyCallWsEndpoint,* por exemplo. E isto faz parte da arquitetura de *Web Services,* quando se usa uma API Java como neste trabalho para TPC. A **Figura 40**, retirada do guia (ERICSSON, 2008b), dá uma visão geral desta arquitetura. O processamento, indicado em tal figura, dentro do *Telecom Web Services Network Emulator* é, exatamente, o comportamento programado no emulador. Nesta imagem, a *Web application* representa a aplicação servidora, vista à esquerda na **Figura 27**. As setas circulares, apontadas pela seta 2, representam, por exemplo, o objeto da classe *ThirdPartyCallWsEndpoint* e outros objetos necessários. O evento descrito pelas setas 3,6, 7 e 8 não ocorre durante o uso do serviço *ThirdPartyCall*.

Existe uma classe na JAX-WS, chamada *WSServlet,* que é a responsável em receber a requisição SOAP, vista pela seta 1 na **Figura 40**.

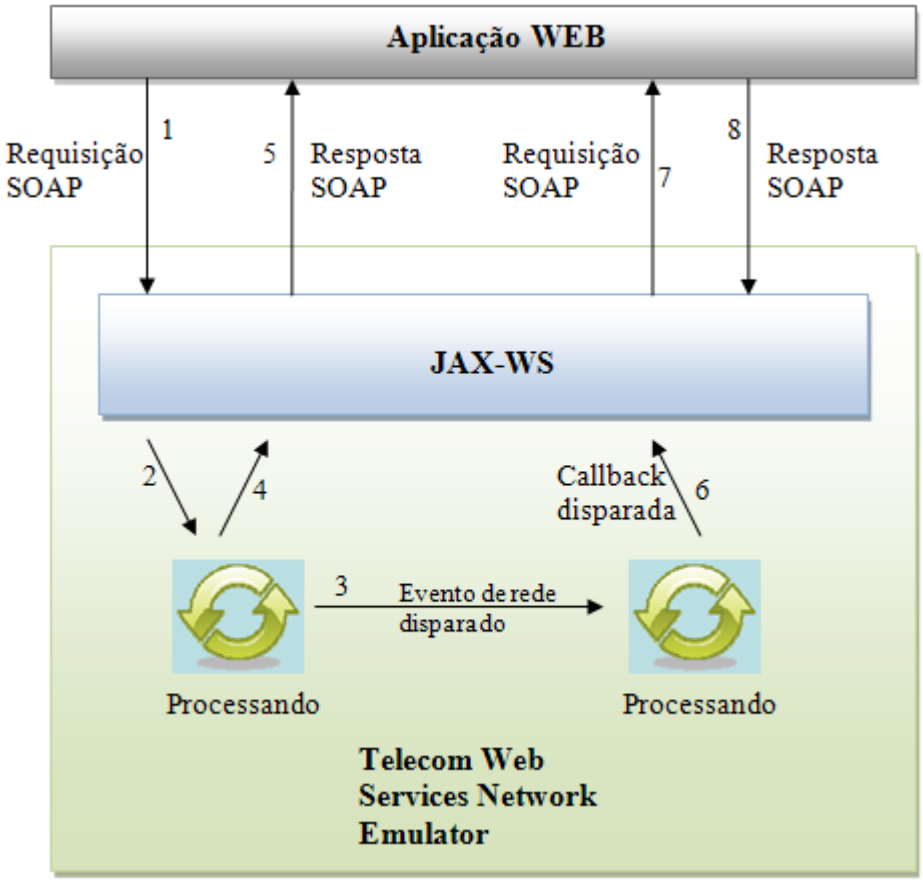

Figura 40 - *Arquitetura Web Services.*

Além disto, deve existir também um mapeamento entre tal classe e todos os *web services* disponíveis via o emulador. Ou seja, um mapeamento deve indicar que o objeto *WSServlet* irá usar um objeto representando o ponto final onde entregar a requisição recebida da aplicação servidora. Este objeto pode ser um *ThirdPartyCallWsEndpoint* ou qualquer outro *EndPoint.* Por exemplo, se o emulador receber uma requisição de informação sobre *status* de terminal, esta requisição será entregue ao objeto da classe *TerminalStatusWsEndpoint*, que já foi codificada pela Ericsson e está presente na versão 3.0 do emulador. Então, tal mapeamento define a conexão entre objetos da JAX-WS e objetos que provêem o comportamento desejado no emulador.

As seguintes passagens de código XML mostram conteúdos do arquivo *web.xml*, resumindo esta explicação:

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

O código acima indica qual é a classe da JAX-WS que constitui um *WSServlet*.

```
B)
<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>
```

O código acima indica que, se o *web server* receber uma requisição HTTP cuja URL contém o nome `ThirdPartyCallService,` então a classe *WSServlet* irá atender a tal requisição.

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

O código acima indica que o objeto da classe *WSServlet* está mapeado para qualquer classe relacionada com o serviço de *ThirdPartyCall* que, no caso, é somente a classe *ThirdPartyCallWsEndpoint*, pela definição constante no arquivo *sun-jaxws.xm,* já comentada nesta seção.

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

O código acima indica que uma URL contendo "*/ThirdPartyCallService*", está mapeada ao *servlet name ThirdPartyCallService*, o que torna válido o mapeamento definido em B.

Resumidamente, quando uma requisição, cuja URL contem "*/ThirdPartyCallService",* chega ao *web server*, pela definição em D o servidor

deve utilizar o servlet *ThirdPartyCallService.* Pela definição em B, isso implica em usar a classe *com.sun.xml.ws.transport.http.servlet.WSServlet.* Esta classe é referenciada pelo *servlet name WSServlet,* conforme A. Pela definição em C, este último servlet está mapeado para qualquer serviço definido no *web server*. Finalmente, pela definição no arquivo *sun-jaxws.xml,* tem-se a classe correta para executar o serviço demandado pela aplicação cliente.

Quanto ao arquivo *build.xml,* apenas mais uma instrução XML fez-se necessária, para que os arquivos Java criados neste projeto fossem compilados juntamente com os outros arquivos Java gerados, automaticamente, mais os arquivos JSP e arquivos .js criados. O código abaixo mostra esta última modificação do arquivo *build.xml*:

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

Todo o projeto do emulador de *gateway* Parlay X da Ericsson, agora incluindo a nova capacidade para *Thrid Party Call*, bem como o código fonte em Java e documentações, podem ser obtidos, via download, diretamente no endereço http://www.biosoftware.com.br/_downloads/Parlay_Gateway_SOURCE_CODE.rar .

Além disso, como visto no Apêndice 3, segundo **Eric Eriksson**, o novo código fonte, desenvolvido durante esta dissertação de mestrado, muito provavelmente, fará parte do código fonte oficial do projeto *Telecom Web Services Network Emulator* da Ericsson e, desta forma, em breve, estará acessível a todos os desenvolvedores cadastrados no programa *Ericsson Mobility World.*

5.7 Arquitetura Final Para o Serviço de TPC.

O diagrama de blocos da **Figura 41** dá uma visão geral dos vários assuntos discutidos nesta dissertação, o relacionamento entre eles e a arquitetura resultante para o emulador.

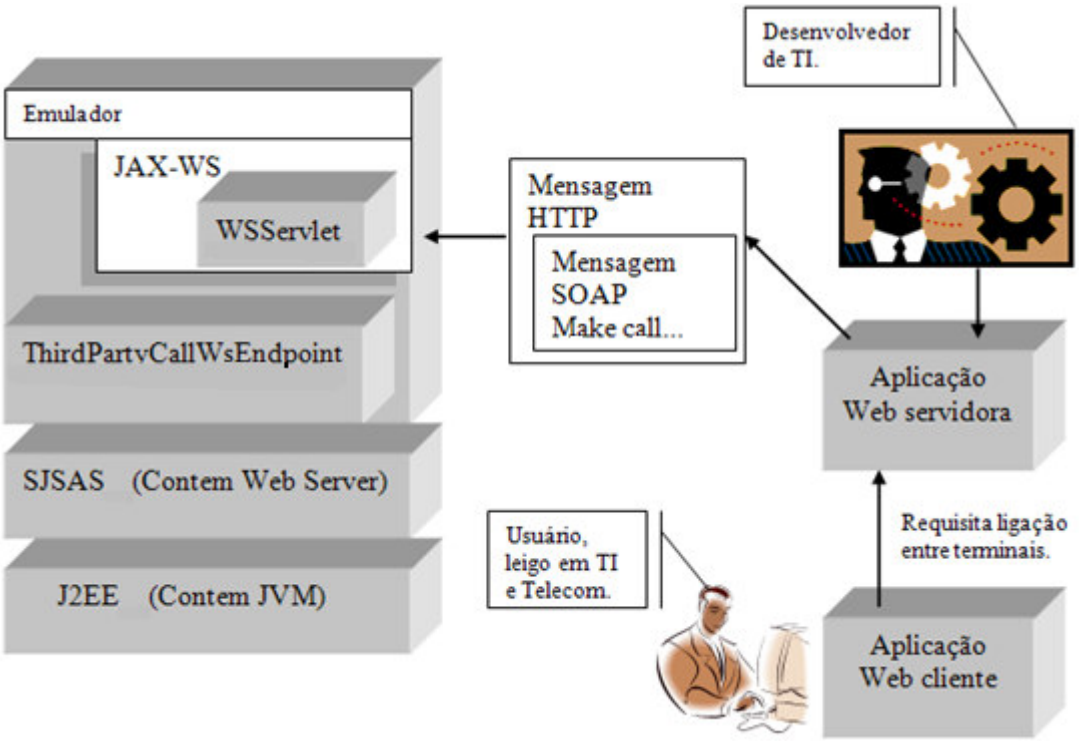

Figura 41 - *Visão geral da arquitetura do sistema com emulador.*

Como visto, agora, um desenvolvedor de TI pode criar a sua aplicação servidora, utilizando *Web Services*, a qual poderá usufruir da capacidade TPC de uma rede de telecomunicações através de um *gateway* Parlay X. Para testá-la, poderá ser usado o emulador da Ericsson modificado. A aplicação servidora poderá receber requisições de uma aplicação cliente, por exemplo, quando se quiser estabelecer uma chamada telefônica entre 2 terminais. Neste caso, um usuário leigo em TI e telecomunicações poderá usar a aplicação cliente, para requisitar o que necessita da rede por trás do *gateway*. O emulador já modificado para a Ericsson está representado na **Figura 41**, com os blocos à esquerda.

5.8 Análise em Resumo e Contribuição

Este capítulo, finalmente, mostrou detalhes de computação relacionados com o trabalho de acréscimo de uma nova funcionalidade no emulador de *gateway*. A partir do entendimento do que foi explicado neste capítulo, fica possível iniciar outros trabalhos relacionados com Parlay X, envolvendo desenvolvimento de *software,* já que muitos detalhes relacionados estão agora esclarecidos. Ficou claro que toda a teoria comentada nos quatro primeiros capítulos pôde ser empregada, quando o trabalho prático descrito foi realizado. Mas, a maior importância deste capítulo é a descrição da organização, ou metodologia, usada para a manipulação correta das ferramentas e tecnologias relacionadas com o desenvolvimento da contribuição proposta. E agora, como fica possível depois de todo o assunto já explicado neste documento, a seção seguite faz um apanhado geral das explicações utilizadas na metodologia seguida, "diluída" dentre as várias páginas anteriores, e forma uma lista resumida concentrando as instruções macros de tal metodologia. Por fim, o resultado das explicações neste capítulo comprova a veracidade das teorias e intenções propostas para o software concluído.

Com o trabalho concluído, pode-se afirmar que foi possível aprender como projetar e implementar uma contribuição para o *software* da Ericsson, nos moldes daquela empresa. Ou seja, foi possível perceber a organização do projeto e a sua arquitetura, de tal forma que novas contribuições não irão violar estas características particulares já criadas. Dificilmente será necessário contatar alguém responsável por este projeto, ao planejar uma nova contribuição, devido ao fato que muitos detalhes pesquisados estão documentados aqui.

5.9 Metodologia Seguida

Resumidamente, as seguintes instruções podem ser seguidas, para compor uma metodologia de desenvolvimento de *software* aplicável ao *gateway* Parlay X da Ericsson, com intuito de atribuir um serviço de telecomunicações ao mesmo. Esta metodologia é útil aos desenvolvedores de *software* responsáveis em criar/contribuir com *gateways* deste tipo.

I) Instalar todas as ferramentas necessárias para o desenvolvimento. Os detalhes sobre isto estão descritos na próxima seção. As ferramentas são os *softwares* de licenças gratuitas que foram utilizados no desenvolvimento do trabalho descrito neste documento.

II) Obter o projeto do emulador de *gateway* Parlay X da Ericsson da Suécia, disponível diretamente no site daquela empresa. Este projeto contém o código fonte Java necessário, que receberá as constribuições, também em Java, do novo trabalho a ser desenvolvido.

III) Estudar as *interfaces* Parlay X relacionadas com a capacidade que se quer incluir no *gateway*. Para um bom conhecimento sobre as *interfaces* a serem tratadas, basta estudar a documentação da API respectiva. Nenhum outro documento adcional faz-se necessário, já que a documentação produzida pelo grupo Parlay é de boa qualidade em termos de didática e quantidade de informações necessárias.

IV) Obter os arquivos WSDL que descrevem, em XML, as *interfaces* necessárias do Parlay X.

V) Em seguida, deve-se compilar os arquivos WSDL. Tal compilação gera, por exemplo, os arquivos Java necessários para iniciar a codificação da nova capacidade a ser incluída no *gateway*. Esta compilação deve ser feita através de um compilador que seja capaz de interpretar a sintaxe dos WSDLs destas *interfaces*, para gerar os arquivos corretamente. Para tal, pode-se usar a função chamada *WsImport* presente na API JAX-WS, como foi mostrado. Esta API está disponível no J2EE. A função

WsImport pode ser chamada através de um *script* XML. Um exemplo de tal chamada é a que está no arquivo *build.xml* disponível juntamente com o código fonte do emulador.

VI) Codificar os algoritmos necessários às funções das *interfaces* em questão. Ou seja, definir o comportamento dos métodos já declarados nas *interfaces* dos novos serviços a serem incluídos no *gateway*. É nesta fase do trabalho que o desenvolvedor de *software* decide como o *gateway* emulado irá se comportar ao receber uma requisição que demandará a sua nova capacidade. Aqui, devido às características da arquitetura do projeto original da Ericsson, algumas classes de tipos determinados deverão ser criadas. Exatamente o que deve ser feito pode ser baseado no que já está pronto, facilmente. É importante reparar que as *interfaces* definidas na API Parlay X resultam em *interfaces* Java consideradas como "*endPoint*s" do ponto de vista das aplicações clientes do *gateway*. Cada *interface* "*endPoint*" terá uma classe que a implementa, obviamente.

VII) Relacionar as classes, que implementam as *interfaces* "*endPoint*", aos *web services* que estarão acessíveis no *gateway*. Este relacionamento é feito com a alteração de algums arquivos XML contidos no projeto do emulador. O primeiro arquivo que pode ser editado é o XML***-jaxws.xml*** como já foi mostrado neste documento. Após isso, editar o arquivo ***web.xml***. Os detalhes sobre estas edições já foram mostrados.

VIII) Editar o arquivo *javascritp* relacionado com a aparência de alguma *interface* gráfica usada no emulador, que ser quer alterar. Por exemplo, no caso das *interfaces* dos telefones emulados, foi necessária a edição do arquivo ***terminalscript.js***. De acordo com cada nova funcionalidade a ser incluída no emulador, um determinado arquivo *.js* poderá ser editado.

IX) Editar o arquivo relacionado com a ação de alguma *interface* gráfica usada no emulador, que se quer alterar. Por exemplo, no caso das *interfaces* dos telefones emulados, foi necessária a edição do arquivo ***DoAcceptCurrentCall.java***. De acordo com cada nova funcionalidade a ser incluída no emulador, um determinado arquivo *.java* poderá ser editado. Cada evento ocorrido na *interface* gráfica do emulador, como um

botão que pode ser pressionado pelo usuário, deve-se gerar uma ação a ser tratada pelo emulador. Estes arquivos de ações programadas têm nomes que inciam com o prefixo *Do*.

X) Editar os arquivos JSP. Estes são os arquivos que executam na máquina onde estiver hospedado o emulador. Por exemplo, pode-se editar um arquivo responsável em acessar e usar outro arquivo local na máquina do *gateway*, como um arquivo de som ou acesso a um banco de dados, se necessário. Por exemplo, no trabalho corrente foi editado o arquivo ***playNoSound.jsp***, para silenciar a campainha do telefone emulado, quando uma chamada é aceita.

XI) Gerar o arquivo *.war*. Quando o projeto de contribuição para o emulador da Ericsson está pronto, pode-se utilizar o arquivo *build.xml* novamente, para compilar todo o projeto e gerar um único arquivo *.war* que pode ser empregado no *web server*. O próprio aquivo *build.xml* tem esta função de inclusão no servidor *web*.

XII) Finalmente, pode-se enviar requisições ao *web server*, porque um *web service*, contido neste servidor, irá responder às requisições, conforme documentado na API da *interface* implementada.

5.10 Ambiente Utilizado de Desenvolvimento de *Software*

Todo o ambiente de desenvolvimento de *software* utilizado neste trabalho é baseado nas recomendações constantes do documento (ERICSSON, 2008b). Ou seja, foram utilizadas as mesmas ferramentas sugeridas em tal documento, exceto o Eclipse, já que o documento (ERICSSON, 2008b) sugere utilizar o *software* NetBeans da SUN para a codificação em Java. As ferramentas de *software* utilizadas aqui, bem como as suas versões e a ordem que as mesmas devem ser instaladas no computador, onde é feito o desenvolvimento de qualquer trabalho semelhante ao corrente, estão descritas abaixo:

5.10.1 Java EE

Utilizar a verão 5, com JDK *update* 2. (inclui *Sun Java System Application Server*). Contém máquina virtual Java e o *kit* de desenvolvimento. Contem também a API JAX-XML. Se no computador já existe Java instalado (Java SE 6 update 2), então basta utilizar o Java EE 5 sem o JDK.

5.10.2 Apache ANT

Software capaz de interpreter e executar arquivos XML.

5.10.3 Jakarta Commons FileUpload

Pacote que manipula *upload* de arquivos, sobre o HTTP, no lado do servidor.

5.10.4 Emulador do *gateway* Parlay X

O emulador pode ser obtido no *website* do programa *Ericsson Mobility World* para desenvolvedores. Durante esta dissertação foi usada a versão 3.0, que é de maio de 2008.

5.10.5 Eclipse

Foi utilizado o Eclipse Europa, da IBM. O código original da Ericsson foi feito na ferramenta NetBeans, da SUN. Tanto o Eclipse, quando o NetBeans são igualmente úteis para o desenvolvimento do trabalho descrito nesta dissertação. E estas duas ferramentas são usadas mundialmente. O próprio documento (ERICSSON, 2008b) (*user guide*) do emulador da Ericsson, versão 3, não sugere usar uma ou outra. Mas, documentos de versões mais antigas comentavam somente sobre o NetBeans. Estes fatos podem levar o desenvolvedor a imaginar que usar o NetBeans é a melhor escolha, para evitar problemas com compiladores, compatibilidade do projeto com a

ferramenta de desenvolvimento, etc. Contudo, como as duas ferramentas de desenvolvimento são igualmente úteis neste caso, durante este trabalho, foi usada a ferramenta Eclipse, o que serve para mostrar que realmente não importa qual delas é a escolhida. Particularmente no caso desta dissertação, a ferramenta Eclipse já era a mais usada durante a fase de análise do código da Ericsson, sendo que continuar com esta ferramenta, na fase de desenvolvimento, tornou-se mais evidente. Para ver uma lista de desvantagens e vantagens de cada uma destas ferramentas, como um comparativo, deve-se procurar por este assunto na Internet, já que isto não faz parte do escopo do trabalho corrente.

6. Conclusão

O desejo pelo crescimento de novos negócios com as redes de telecomunicações tem sido uma força impulsionadora do desenvolvimento de APIs para redes abertas, tais como as APIs Parlay/OSA e Parlay X. Através destas APIs espera-se uma adesão significativa de desenvolvedores de *software* à indústria de telecomunicações, trazendo para tal indústria novas aplicações criativas e úteis ao cotidiano das pessoas, de tal forma que as operadoras possam oferecer novos serviços com valor agregado, afim de que haja mais retorno financeiro sobre o investimento neste mercado. De fato, os profissionais experientes em Tecnologia da Informação, com o uso de tecnologias de desenvolvimento de *software*, como Java, XML, JSP e *Web Services*, completam o ambiente de prestação de serviços inovadores das operadoras, porque trazem consigo os conhecimentos que não são de domínio dos profissionais de Telecomunicações. Por outro lado, os profissionais de TI, ávidos por entrar no mercado de telecomunicações, mas sem destreza com as tecnologias deste mercado, necessitam de *interfaces* comuns as quais possibilitem requisições às capacidades das operadoras, como requisição de envio de mensagens, requisição de localização de terminais, requisição de estabelecimentos de chamadas entre terminais, etc. Portanto, as APIs que descrevem as capacidades das operadoras e como usá-las são a 'ponte' entre o mundo de TI e de Telecomunicações. Para que esta idéia se concretize, deve haver trabalhos com o intuito de implementar as APIs e torná-las disponíveis ao público interessado.

As APIs Parlay são descrições formais, o que pode ser feito em UML, das capacidades que estarão disponíveis (´visíveis´) nas redes de telecomunicações, para

profissionais fora do domínio seguro das operadoras. Portanto, é de interesse de várias empresas que estas especificações atendam aos interesses delas próprias. Neste caso, grandes empresas estão trabalhando na definição de APIs, como aquelas vistas na **Figura 11**. Devido a isto e à utilidade das próprias APIs, surge uma situação que indica que os trabalhos do Parlay não serão abandonados facilmente e o conhecimento deste assunto poderá abrir novas possibilidades de projetos para os profissionais de Tecnologia da Informação. Este tipo de trabalho encaixa-se no objetivo das NGNs de organizar as redes com uma camada de prestação de serviços isolada de uma camada de recursos de transmissão de dados.

As entidades que implementarão as *interfaces* destas APIs serão os *gateways*, que aparecerão nas redes como um *firewall* entre o domínio das operadoras e o domínio externo onde se encontrarão as aplicações de terceiros. Do lado das operadoras, ainda haverá trabalho de desenvolvimento de *software* com tecnologias direcionadas às telecomunicações, como SIP, IMS, etc. Ou seja, as tecnologias de telecomunicações serão as 'engrenagens' dos *gateways* e tais tecnologias também poderão ser alvos de objetivos das NGNs, como a arquitetura de IMS. No lado externo das operadoras, ou seja, onde estarão as aplicações que requisitarão serviços aos *gateways*, o trabalho de desenvolvimento de aplicações será suportado por tecnologias de TI, geralmente, como banco de dados, *web servers*, HTTP, Java e XML. Contudo, criatividade para o uso de recursos de telecomunicações, provendo serviços inovadores, será fundamental aos criadores das novas aplicações. Em caso contrário, dificilmente haverá valor agregado nas mesmas.

Esta dissertação mostrou, então, algumas tecnologias disponíveis para a criação de serviços com valor agregado para as telecomunicações. Contribuiu também mostrando como utilizá-las, numa ordem correta. Isto é, devido ao grande número de tecnologias necessárias, é preciso saber o momento de usar cada uma e como encaixar entre si os artefatos resultantes de tal uso, para compor um produto final

(*software*). Por exemplo, foi visto que, para criar uma aplicação capaz de se comunicar com um *gateway* com *interfaces* Parlay X, é necessário, nesta ordem:

1. Estudar as *interfaces* que descrevem o recurso necessário na rede. Por exemplo, serviço de MMS. Assim, fica possível saber como usar tais recursos, invocando corretamente os métodos disponíveis nos mesmos.
2. Obter os arquivos WSDL, descrições em XML das mesmas *interfaces*.
3. Usar uma ferramenta, também em software, capaz de mapear o código WSDL em uma linguagem de programação, como Java.
4. Criar novas classes Java, por exemplo, para a implementação da lógica da aplicação com valor agregado que se quer construir. As novas classes devem usar as classes criadas automaticamente e/ou implementar *interfaces* definidas a partir do mapeamento de WSDL para a linguagem de programação.
5. Compilar todas as classes e gerar a aplicação final.

Além disso, foi mostrado também como implementar novas funcionalidades para um *gateway* Parlay X.

Com a apresentação de um emulador de *gateway* Parlay X da Ericsson, com suas funcionalidades, foi mostrado que a capacidade de emular o estabelecimento de chamada telefônica entre 2 terminais ainda não era parte daquele emulador. Mas, a partir do entendimento de Parlay X, manipulações de tecnologias como Java, XML, *Web* Services e WSDL, foi possível planejar e explicar sobre a inclusão desta nova capacidade no *gateway* emulado. Para comprovar toda a teoria comentada, a contribuição sugerida foi, de fato, implementada, com sucesso. Portanto, baseado nas referências bibliográficas desta dissertação e nas explicações contidas na mesma, percebe-se que existe um roteiro a ser seguido, para a criação de *gateways* Parlay X ou Parlay/OSA e que as definições das APIs, realmente, suprem as necessidades dos

desenvolvedores de TI de conhecer meios de acesso aos recursos das redes de telecomunicações.

Conclui-se que as APIs para as redes abertas promovem uma forma excitante e simplificada de uso dos recursos das operadoras, se comparado ao que é necessário aprender para lidar com tecnologias de redes sem *interfaces* deste tipo, e não é, obrigatoriamente, necessário ter acesso a um *gateway* real, para testar uma aplicação que dependerá deste tipo de elemento de rede. Pode-se utilizar emuladores de *gateways* e, se estes estiverem implementando *interfaces* Parlay X, todo o conhecimento em *Web Services* torna-se aplicável neste contexto, de tal forma que, do ponto de vista de uma aplicação cliente de *web service,* o trabalho de acessar um recurso de rede consiste em um trabalho computacional.

Em suma, as principais contribuições deste trabalho são:

- Citações de ferramentas úteis ao aprendizado das tecnologias relacionadas com Parlay.
- Explicações de como utilizar tais ferramentas numa ordem correta.
- Demonstração de um emulador de *gateway* Parlay X.
- Explicações de como contribuir para o progresso de tal emulador e efetivação de uma contribuição real para o mesmo.

Outras contribuições que também podem ser mencionadas são:

- Demonstração da existência da demanda por aplicações inovadoras, de valor agregado, por parte das empresas de telecomunicações, o que poderá gerar oportunidade para novos projetos dos profissionais de TI.
- Explicações sobre como portar, tecnologicamente, no mundo de telecomunicações, as aplicações desenvolvidas no mundo de TI.
- Explicações e exemplos sobre *interfaces* para a abertura das redes das operadoras de telecomunicações.

- Demonstração da existência das *interface*s Parlay, além de seus detalhes, bem como o relacionamento delas com outras tecnologias.

Por fim, caso um profissional de TI, leigo em telecomunicações e *interfaces* de redes abertas, interessado em criar uma aplicação de valor agregado para uma operadora, como a Brasil Telecom, decida iniciar seus estudos neste campo, ele poderá primeiramente fazer a leitura deste documento e, então, estudar as *interfaces* Parlay, através das respectivas especificações do ETSI. Além disto, existem os materiais fornecidos pelo programa *Ericsson Mobility World* que, como o (ERICSSON, 2008a), guiam os passos do desenvolvedor por um roteiro correto, para atingir o objetivo.

O emulador de *gateway* da Ericsson, fornecido no programa *Ericsson Mobility World,* ainda pode receber várias contribuições, o que irá acelerar o progresso do esforço daquela empresa em ajudar os profissionais de TI a testarem suas aplicações. Por exemplo, um trabalho futuro, figurando como uma continuação deste trabalho, poderia ser a inclusão de uma nova capacidade neste emulador, conforme a API *Payment*, do Parlay X. Agora, com as explicações presentes aqui, tal trabalho já não apresentará tantos desafios, como os notados para a inclusão de *ThirdPartyCall*.

Um possível trabalho futuro poderia ser a análise da utilidade da API Parlay X, quando comparada com a API Parlay/OSA ou outras tecnologias de mais baixo nível, como SIP. Ou seja, analisar se a quantidade de funções diferentes disponíveis através da API Parlay X é um fator restringente para o desenvolvimento de alguma aplicação de valor agregado, como se tecnologias de mais baixo nível tivessem que ser realmente utilizadas, por desenvolvedores de TI, para suprir alguma incapacidade funcional relacionada com Parlay X. Se uma aplicação de valor agregado não puder ser implementada somente sobre Parlay X, então este fato poderá ser usado para mostrar as desvantagens desta API. Isto seria o mesmo que mostrar que Parlay X

limita, negativamente, a quantidade de funções disponíveis às aplicações de valor agregado. Além disso, será conveniente analisar se o uso de Parlay X + Parlay/OSA causará alguma deficiência de *performance* para alguma aplicação específica, quando, talvez, poderá ser melhor não usar estes *gateways*, para ganhos de velocidade de interação com a rede.

A - Apêndice 1

De: atendimento@brasiltelecom.com.br
Enviado em: domingo, 13 de abril de 2008 17:09
Para: Rodrigo Pimenta Carvalho
Assunto: Resposta Padrão de APIs Parlay X
[T20080413019YS050Z4768225]

Prezado Rodrigo Pimenta Carvalho,

Obrigado por entrar em contato com o Atendimento Virtual da Brasil Telecom.

A Brasil Telecom já disponibiliza as APIs Parlay 3.1 para desenvolvedores externos poderem escrever aplicações para atender demandas
específicicas da compania.

A adoção de API Parlay X em um ambiente baseado em Web Services faz parte do escopo de evolução da infra-estrutura implantada.
Com a adoção de Parlay X será também disponibilizada interface externa em ambiente de testes controlado e com acesso limitado, de forma a permitir que desenvolvedores cadastrados e homologados possam testar suas aplicações dentro de condições reais de rede.

O prazo previsto para a completa migração para Parlay X ainda não pode ser anunciado publicamente,
mas será tornado público assim que todas as condições técnicas e comerciais pertimentes estiverem equacionadas.

Estamos sempre prontos para atender você, por e-mail ou pelas nossas Centrais de Relacionamento.
(Consultar em http://www.brasiltelecom.com.br)

Atendimento Virtual
Brasil Telecom
www.brasiltelecom.com.br/sac_on_line
RWC

B - Apêndice 2

Uma das pessoas que mais esclareceram dúvidas sobre como organizar o ambiente computacional necessário à implementação da nova capacidade para o emulador foi **Erik Eriksson**, que trabalha no time de desenvolvedores do citado sistema, na Ericsson da Suécia.

Com a conclusão deste trabalho, um novo arquivo *parlayx_gateway.war* foi gerado e enviado para o **Erik Eriksson**, que analisou a nova funcionalidade *ThirdPartyCall* no emulador, fez testes e comentou que a implementação está correta e tudo mais funcionando apropriadamente.

```
-----Mensagem original-----
De: Erik Eriksson XB [mailto:****@ericsson.com]
Enviada em: quinta-feira, 26 de junho de 2008 04:06
Para: Rodrigo Pimenta Carvalho
Assunto: SV: File WAR

Hi Rodrigo,

I finally got the file, thank you.

I tested the emulator and it looks good. All the tpcc services were
working and the phones were ringing.

So I'm now curios on taking a look at the code. Is the code license
free, that will say the code has no license? If that is the case
then I think it would be possible to add the feature in our next
release if the code looks alright.

BR
Erik
```

C - Apêndice 3

-----Original Message-----
From: Erik Eriksson XB
Sent: den 15 juli 2008 10:45
To: 'Rodrigo Pimenta Carvalho'
Subject: RE: Emulator TPC source code available!

Hi Rodrigo,

I implemented your source code in our emulator and it runs fine. I currently have one bug left (that I'm currently aware of) to solve and that's when I cancel a call request before any of the parts have answered.
The problem is that the status is not changed from Busy to Reachable in the terminal gui. But I'll have a look on it today.

We will most likely include this code in the next version of the emulator. Your code will then be under Ericsson's license (the same license we have today). Your name will still be in the source code and we would also like to have your email address (a permanent) in the code.
In that case people can also contact you if they got any code questions.

Best Regards
Erik Eriksson

D - Apêndice 4

De: MELISSA TABOADA (SP/EBS) [melissa.taboada@ericsson.com]

Enviado em: terça-feira, 10 de julho de 2007 10:05

Para: Rodrigo Pimenta Carvalho

Cc: RODRIGO LUGLIO (SP/EBS)

Assunto: RE: Dúvida sobre ferramenta da Ericsson, para desenvolvimento.

Olá Rodrigo,

Com relação a dúvidas técnicas, o Rodrigo Luglio em cópia é um contato, que apesar de não ter muito conhecimento em Parlay pode ajudá-lo com informações ou até mesmo direcioná-lo às pessoas responsáveis por cada área técnica.

Eu cuido da parte de parceria de empresas para oferta de soluções desenvolvidas aos nossos clientes, e estou a disposição para auxiliá-lo, quando possível.

No Brasil temos duas plataformas de Parlay-X vendidas a clientes Ericsson, uma das soluções que está em andamento é um Localizador de Restaurantes.

Espero ter-lhe esclarecido e auxiliado em suas dúvidas.

Atenciosamente

Melissa Taboada do Lago

Mobility World

Phone: +55 11 6224-8930

Mobile: +55 11 9496-0647

Mail to: melissa.taboada@ericsson.com

Rua Maria Prestes Maia, 300

Vila Guilherme, São Paulo, SP

CEP 02047-901

www.ericsson.com/mobilityworld

Referências

DEITEL, H. M.; DEITEL, P. J. **Java, Como Programar**. Tradução: Carlos Arthur Lang Lisbôa. 4. ed. Porto Alegre: Bookman, 2003. p. 59. ISBN 85-363-0123-6

INTERNATIONAL TELECOMMUNICATION UNION; ITU-T. **Recommendation Y.2001.** Dec, 2004. Disponível em: < http://www.itu.int/ITU-T/ngn/index.html >. Acesso em: 03 set. 2007.

KUROSE, J. R.; ROSS, K. W. **Redes de Computadores e a Internet:** Uma abordagem top-down**.** trad. Arlete Simille Marques. 3. ed. São Paulo: Pearson Addison Wesley, 2006. ISBN 85-88639-18-1

OLIVEIRA, E. T. O Horizonte das Redes de Próxima Geração. **World Telecom**, São Paulo, SP, ano V, n. 50, p. 32, set. 2002.

CARVALHO, R. P.; ALBERTI, A. M. Java Technologies for NGN Service Creation: Discussion and Architecture to Improve SIP Addresses Discovery. In: International Conference on Internet
and Multimedia Systems and Applications (EuroIMSA), 2007, **IASTED Paper**, Chamonix, France, IASTED, 14-16 Mar 2007.

TANENBAUM, A. S. **Redes de Computadores**. 7ª tiragem. Rio de Janeiro: Campus, 1997. ISBN 8-352-0157-2

WIKIPEDIA. **Convergência tecnológica**. Disponível em: < http://pt.wikipedia.org/wiki/Converg%C3%AAncia_tecnol%C3%B3gica >. Acesso em: 10 ago. 2008.

NET e Embratel começam a vender "telefone a cabo". Editor: Sérgio Ripardo. **FolhaOnline**, [São Paulo], 21 mar. 2006. Disponível em: <http://www1.folha.uol.com.br/folha/dinheiro/ult91u106173.shtml>. Acesso em: 05 ago. 2008.

THE PARLAY GROUP. Parlay and Next-Generation Networks**.** **05391r03PG_Marketing-WP-Parlay_and_NGN[1].doc.** May 2005. Disponível em: < http://www.parlay.org/en/resources/>. Acesso em: 05 ago. 2008.

INTERNATIONAL ENGINEERING CONSORTIUM. Next-Generation Communications Environments: Guiding Principles for Legacy Replacement . **glenayre_com_environ.pdf**. 25 Feb. 2005. Disponível em:< http://www.iec.org/online/tutorials/ >. Acesso em: 06 ago. 2008.

ROSA, C. **Parlay**: O que é?. 20 Mar. 2006. Disponível em: < http://www.teleco.com.br/tutoriais/tutorialparlay/pagina_1.asp>. Acesso em: 05 ago. 2008.

VoipDiscount. Software para transmissão de voz sobre IP. 1 arquivo executável. Disponivel em: < http://www.voipdiscount.com/en/index.html>. Acesso em: 03 set. 2008.

OSSE, J. S. Últimas Notícias: British Telecom tem queda de 11% no lucro do quarto trimestre. **UOL Economia,** São Paulo, 18 maio 2006. Disponível em: < http://noticias.uol.com.br/economia/ultnot/valor/2006/05/18/ult1913u50737.jhtm>. Acesso em: 06 ago. 2008.

MOREIRA, D. Receita da telefonia móvel cresce 20,5% em um ano e se aproxima da fixa. **IDG NOW!,** [S.I.], 20 nov. 2006. Disponível em: < http://idgnow.uol.com.br/telecom/2006/11/20/idgnoticia.2006-11-20.0384022511/>. Acesso em: 06 ago. 2008.

BRASIL TELECOM. **Setor de Telecomunicaçõe**. [S.I.], 15 set. 2007. Disponível em: <http://www.mzweb.com.br/brasiltelecom/web/conteudo_pt.asp?tipo=9544&id=15255&idioma=0&conta=28#>. Acesso em: 10 nov. 2007.

O NOVO Cenário das Telecomunicações. **Valor Econômico** [São Paulo], 2006. Valor Online, Valor Análise Setorial, Telecomunicações, v. 1, p. 10. Disponível em: <http://setorial.valor.com.br/>. Acesso em: 10 nov. 2007.

ARTIGAS, F. C. de O.; NUNES, G. H. C. **Redes NGN**: Introdução. 12 Nov. 2007. Disponível em: < http://www.teleco.com.br/tutoriais/tutorialngnconverg/pagina_1.asp>. Acesso em: 05 ago. 2008.

WIKIPEDIA. **Next Generation Networking.** 16 July 2008. Disponível em: <http://en.wikipedia.org/wiki/Next_Generation_Networking>. Acesso em: 05 ago. 2008.

CASTRO, A.; LOURENÇO, R. B. **Next Generation Networks.** Niterói: Universidade Federal Fluminense, Escola de Engenharia, [200-].

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. TISPAN. **Defining the Next Generation Network.** Apr. 2008. Disponível em: <http://www.etsi.org/tispan/>. Acesso em: 05 ago. 2008.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T SG 13 Work Programme (2005-2008).** Cronograma do ITU para a padronização das NGN. Disponível em: < http://www.itu.int/ITU-T/workprog/wp_search.aspx?isn_sp=1&isn_sg=116 >. Acesso em: 10 ago. 2008.

HEINISCH, A. M. C. **NGN III**: Considerações finais. 17 Apr. 2006. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialngnIII/pagina_6.asp >. Acesso em: 05 ago. 2008.

MAGEDANZ, T.; WITASZEK, D.; KNUTTEL, K. Service Delivery Platform Options for Next Generation Networks within the national German 3G Beyond Testbed **…**In: South African Telecommunication Networks Architectures Conference (SATNAC)**,** 2004, Stellenbosch, South Africa. **Paper**. Stellenbosch: [s.n.], 8 Sep. 2004. ISBN 0-620-32632-8.

VASQUES, E. À procura de um rumo. **B2B magazine**, São Paulo, SP, ano 6, n. 79, p. 33, set. 2007.

INTERNATIONAL ENGINEERING CONSORTIUM. Voice Portal Solutions: An Introduction to Next-Generation Network Services…**voice_portal.pdf**. [entre 2006 e 2007] Disponível em:< http://www.iec.org/online/tutorials/ >. Acesso em: 06 ago. 2008.

MOYER, S.; UMAR, A. A. The impact of network convergence on telecommunications software. **IEEE Communications Magazine,** [S.I.], v. 39, p. 78-84, Jan. 2001. ISSN: 0163-6804. Disponível em: < http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=894380>. Acesso 6 mar. 2007.

INTERNATIONAL TELECOMMUNICATION UNION. **Next Generation Networks Global Standards Initiative**. Disponível em: <http://www.itu.int/ITU-T/ngn/index.html>. Acesso em: 9 ago. 2007.

INTERNATIONAL TELECOMMUNICATION UNION. NGN Focus Group. [Estudos apresentados sobre NGN] In: NGN Industry Event, 2005, London. **Proceedings.** [Geneva, Switzerland], Nov. 2005b. p 109-144. Part II. Disponível em: <http://www.itu.int/ITU-T/ngn/release1.html>. Acesso em: 5 ago. 2008.

CARUGI, M. Service Requirements and Capabilities of NGN. In: ITU-T Worshop on “Next Generation Networks”, 2006, Hanoi, Vietnam. **Tópico temático**. Disponível em: < http://www.itu.int/ITU-T/worksem/ngn/200605/presentations/s1_carugi.pdf>. Acesso em: 05 ago. 2008.

INTERNATIONAL TELECOMMUNICATION UNION. **NGN 2004 Project description.** Version 3. 9_ww9.doc. 12 Feb. 2004. Disponível em: < http://www.itu.int/itudoc/itu-t/com13/ngn/9.html>. Acesso em: 05 ago. 2008. Word for Windows 2000.

FALCARIN, P.; LICCIARDI, C. A. Technologies and Guidelines for Service Creation in NGN. **exp,** [Italy], v. 3, n. 4, p. 46-53, Dec. 2003. Disponível em: <http://exp.telecomitalialab.com>. Acesso em: 2006.

INTERNATIONAL TELECOMMUNICATION UNION. [Estudos apresentados sobre NGN] In: NGN Industry Event, 2005a, London. **Proceedings**. [Geneva, Switzerland], Nov. 2005. p 89-107. Part I. Disponível em: <http://www.itu.int/ITU-T/ngn/release1.html>. Acesso em: 5 ago. 2008.

SUN MICROSYSTEMS. **API Specifications:** JAIN-SIP, versão 1.1. Disponível em: < http://jcp.org/en/jsr/detail?id=32>. Acesso em: 9 ago. 2008a.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGIES. Web site que contem o projeto da pilha NIST-SIP. Disponível em: < http://snad.ncsl.nist.gov/proj/iptel/>. Acesso em: 9 ago. 2008.

LAURETTI, S. R. **Evolução das Redes de Telecomunicação:** Arquitetura IMS. 06 dez. 2004. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialims/default.asp>. Acesso em: 3 set. 2008.

THE PARLAY GROUP; WEB SERVICES WORKING GROUP. Parlay Web Services: Overview. Version 1.0. **04308r00PG_Accelerator-Parlay-Web-Services---Overview-Version-1.0.pdf.** Oct. 31, 2002a. Disponível em: < http://www.parlay.org>. Acesso em: 12 nov. 2007.

SUN MICROSYSTEMS. Página online com perguntas e respostas sobre JAIN, mantida pela empresa SUN. [200-?]. Disponível em: < http://java.sun.com/products/jain/qa.html>. Acesso em: 9 ago. 2008.

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, Que., Canada: Concordia Univ., v. 18, p. 48-55, 1 June 2004. ISSN: 0890-8044.

WIKIPEDIA. **Java APIs for Integrated Networks.** 27 July 2007. Disponível em: < http://en.wikipedia.org/wiki/Java_APIs_for_Integrated_Networks >. Acesso em: 9 ago. 2008.

SUN MICROSYSTEMS. JAIN and Java in Communications. **Jain_and_Java_in_Communications-1_0.pdf.** Santa Clara, California, Mar. 2004. Disponível em: < http://java.sun.com/products/jain/reference/whitepapers/index.html>. Acesso em: 9 ago. 2008.

SUN MICROSYSTEMS. **API Specifications**. Disponível em: < http://java.sun.com/products/jain/api_specs.html>. Acesso em: 9 ago. 2008b.

DEBBABI, M. et al. **API JAIN Presence.** [S.I.]: Sun Microsystems, 30 Jan. 2006. Disponível em: < http://jcp.org/en/jsr/detail?id=186>. Acesso em: 6 mar. 2007.

THE PARLAY GROUP. Web site oficial do grupo Parlay. [200-?]. Disponível em: <http://www.parlay.org/en/index.asp>. Acesso em: 9 ago. 2008.

GUPTA, M. Parlay/OSA mature for the telecoms market. In: Eurescom Workshop 'OSA and Parlay @ Work', 13 Nov. 2002, Heidelberg. **Press Release**.

VENTERS, T. Open Standard Initiatives For Service Delivery Platforms. **TMCnet,** [S.I.], Mar. 18, 2004. Disponível em: < http://www.tmcnet.com/tmcnet/articles/2004/031804tv.htm>. Acesso em: 9 ago. 2008.

THE PARLAY GROUP; BRITISH TELECOMMUNICATIONS. Comparing OMA OSE and Parlay Architectures. **05382r00PG_Marketing-2005_04_Comparing-OMA-OSE-and-Parlay-Architectures[1].pdf.** Ipswich, Suffolk, UK: Parlay, Mar. 2005. Disponível em: < http://www.parlay.org/en/resources/>. Acesso em: 9 ago. 2008.

MAGEDANZ, T. **Parlay 101**:Getting Started with the Parlay /OSA APIs. Concept, Architecture and API Overview. Berlin: Fraunhofer Institute FOKUS, 17 June 2004. Disponível em: < magedanz@fokus.fraunhofer.de >. Acesso em: [2006?].

ODADZIC, B.; JANKOVIC, M. Open Service Access (OSA) Business Models and Service Level Agreement Aspects. In: Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003. 6th International Conference on, Serbia and Montenegro, **IEEE Paper,** [S.I.], IEEE, Oct. 2003, v.1, p. 22-25.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Application Programming Interface (API). **06740r01PG_Accelerator-Parlay_5.1_Part_3.pdf.** Sophia Antipolis, France: ETSI, Jan. 2007a. ETSI standard: ETSI ES 203 915-3, v. 1.2.1, Part 3: Framework. Disponível em: < http://www.parlay.org/en/specifications/ >. Acesso em: 15 jan. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Application Programming Interface (API). **06761r01PG_Accelerator-Parlay_5.1_Part_9.pdf.** Sophia Antipolis, France: ETSI, Jan. 2007b. ETSI standard: ETSI ES 203 915-9, v. 1.2.1, Part 9: Generic Messaging. Disponível em: < http://www.parlay.org/en/specifications/ >. Acesso em: 15 jan. 2008

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Application Programming Interface (API). **06757r01PG_Accelerator-Parlay_5.1_Part_7.pdf.** Sophia Antipolis, France: ETSI, Jan. 2007c. ETSI standard: ETSI ES 203 915-7, v. 1.2.1, Part 7: Terminal Capabilities. Disponível em: < http://www.parlay.org/en/specifications/ >. Acesso em: 15 jan. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Application Programming Interface (API). **06771r01PG_Accelerator-Parlay_5.1_Part_14.pdf.** Sophia Antipolis, France: ETSI, Jan. 2007d. ETSI standard: ETSI ES 203 915-14, v. 1.2.1, Part 14: Presence and Availability Management. Disponível em: < http://www.parlay.org/en/specifications/ >. Acesso em: 15 jan. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Application Programming Interface (API). **06755r01PG_Accelerator-Parlay_5.1_Part_6.pdf.** Sophia Antipolis, France: ETSI, Jan. 2007e. ETSI standard: ETSI ES 203 915-6, v. 1.2.1, Part 6: Mobility. Disponível em: < http://www.parlay.org/en/specifications/ >. Acesso em: 15 jan. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Application Programming Interface (API). **06767r01PG_Accelerator-Parlay_5.1_Part_12.pdf.** Sophia Antipolis, France: ETSI, Jan. 2007f. ETSI standard: ETSI ES 203 915-12, v. 1.2.1, Part 12: Charging. Disponível em: < http://www.parlay.org/en/specifications/ >. Acesso em: 15 jan. 2008.

LECLERC, M. . Network Resource Gateway: Benefits and Business Opportunities of Building Wireless Applications Using Parlay/OSA for Developers. **Part3.ppt.** Montreal (Québec), Canada: ERICSSON, 8 Dec. 2003. Part 3: Developing Parlay/OSA Applications. Disponível em: < http://netstorage.ericsson.com.edgesuite.net/marcleclerccd3/mainframe_high.htm>. Acesso em: 9 ago. 2008.

WIKIPEDIA. **Web service.** Disponível em: < http://en.wikipedia.org/wiki/Web_service>. Acesso em: 25 jan. 2008.

CERAMI, E**. Web Services Essentials**: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. 1. ed.. [S.I]: O'Reilly, Fev. 2002. ISBN: 0-596-00224-6.

THE APACHE SOFTWARE FUNDATION. **AXIS2**. Disponível em: < http://ws.apache.org/axis2/ >. Acesso em: 05 set. 2008.

RECKZIEGEL, M. **Gerenciamento de Infra-estruturas de Medição usando Web Services.** Santa Rosa, RS: Universidade Regional do Noroeste do Estado do Rio Grande do Sul, Departamento de Tecnologia, [200-?]. Provavelmente apresentado como trabalho de conclusão de curso de Sistemas de Informação.

THE PARLAY GROUP. Parlay Web Services Architecture Comparison. Version 1.0. **04306r00PG_Accelerator-Parlay-Web-Services---Architecture-Comparison-Version-1.0-.pdf. 31** Oct. 2002b. Disponível em: < http://www.parlay.org/en/resources/>. Acesso em: 05 ago. 2008.

YIM, JONG-CHOUL; CHOI, YOUNG-ILI; LEE, BYUNG-SUN. Third Party Call Control in IMS using Parlay Web Service Gateway. In: Advanced Communication Technology. ICACT 2006. The 8th International Conference, 2006, [S.I.], **IEEE Paper,** [S.I]: IEEE, 22 Fev. 2006. p. 221-224. ISBN 89-5519-129-4.

WEGSCHEIDER, F.; BESSLER, S.; GRUBER, G. Interworking of presence protocols and service interfaces. In: Wireless And Mobile Computing, Networking And Communications, 2005. (WiMobapos;2005), IEEE International Conference on, [S.I.], **IEEE Paper**, [S.I.], IEEE, 22-24 Aug. 2005, p. 45-52. v. 4.

THE PARLAY GROUP; WEB SERVICES WORKING GROUP. Parlay Web Services: Application Deployment Infrastructure. Version 1.0. **04305r00PG_Accelerator-Parlay-Web-Services---Application-Deployment-Infrastructure-.pdf.** Oct. 31, 2002c. Disponível em: < http://www.parlay.org>. Acesso em: 12 nov. 2007.

ERICSSON. Parlay X Web Services. **parlay_x_web_services.pdf**. June 22, 2006. Disponível em: < http://www.ericsson.com/mobilityworld/sub/open/technologies/parlayx/docs/parlay_x_web_services >. Acesso em 9 ago. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Parlay X Web Services. **es_20239102v010201p.pdf.** Sophia Antipolis, France: ETSI, Dec. 2006a. ETSI standard: ETSI ES 202 391-2, v. 1.2.1, Part 2: Third Party Call. Disponível em: < http://www.parlay.org/en/specifications/pxws.asp>. Acesso em: 9 ago. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Application Programming Interface (API). **06745r01PG_Accelerator-Parlay_5.1_Part_4-2.pdf.** Sophia Antipolis, France: ETSI, Jan. 2007. ETSI standard: ETSI ES 203 915-4-2, v. 1.2.1,

Part 4: Call Control, Sub-part 2: Generic Call Control. Disponível em: < http://www.parlay.org/en/specifications/ >. Acesso em: 15 jan. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS. **ETSI OSA Parlay X**: Parlay X 3.0 Specifications. Disponível em: < http://portal.etsi.org/docbox/TISPAN/Open/OSA/ParlayX30.html>. Acesso em: 9 ago. 2008.

IBM. **Introducion to IP Multimedia Subsystem (IMS), Part 1**: SOA Parlay X Web Services. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-soa-ipmultisub1/>. Acessado em: 15 set. 2008.

THE EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE; THE PARLAY GROUP. Open Service Access (OSA): Parlay X Web Services. **es_20239104v010201p.pdf.** Sophia Antipolis, France: ETSI, Dec. 2006b. ETSI standard: ETSI ES 202 391-4, v. 1.2.1, Part 4: Short Messaging. Disponível em: < http://www.parlay.org/en/specifications/pxws.asp>. Acesso em: 10 ago. 2007.

NEWMARCH, J. **Guide to Jini Technologies**. Versão 4.03. Disponível em: < http://jan.newmarch.name/java/jini/tutorial/Jini.xml >. Acesso em: 9 set. 2008.

TAKATAMA, H.; TANI, H. Intelligent SIP System for Mobile Internet. In: Intelligent Network Workshop, 2001. Boston, **IEEE Paper**, [S.I.], IEEE, 6-9 May 2001, p. 83-93. NEC Corporation.

RUMBAUGH. J. et al. **Modelagem e Projetos Baseados em Objetos**. Tradução: Dalton Conde de Alencar. Rio de Janeiro: Campus, 1997. ISBN 85-700-841-X

SEDLAR, U. et. al. Bringing Click-to-Dial Functionality to IPTV Users. **IEEE Communications Magazine**, Toronto, Ont., Canada, v. 46, p. 118-125, Mar. 2008. ISSN: 0163-6804.

AJAM, N. Privacy Based Accessto Parlay X Location Services. In: Networking and Services, 2008. Fourth International Conference on Networking and Services, 2008. IEEE Computer Society. **IEEE Paper,** IEEE, Mar. 2008. Gosier. p. 204-210, ISBN: 978-0-7695-3094-9

ERICSSON. Página web para download do Telecom Web Services Kit**.** Disponível em: < http://www.ericsson.com/mobilityworld/sub/open/technologies/parlayx/tools/telecom_web_services >. Acesso em: 1 abr. 2008a.

ERICSSON. **Telecom Web Services Network Emulator**: Developer's Guide. [Suécia]: Ericsson, 31 Mar. 2008. Disponível em: <

http://www.ericsson.com/mobilityworld/sub/open/technologies/open_development_tips/tools/telecom_network_emulator >. Acesso em: 1 abr. 2008b.

WIKIPEDIA. **AJAX (programming)**. Disponível em: < http://en.wikipedia.org/wiki/AJAX>. Acesso em: 15 set. 2008.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR6023:** Informação e documentação: Referências. Rio de Janeiro, 2002