SERVICE DELIVERY PLATFORMS: THE KEY TO SERVICE CONVERGENCE

DEVOTEAM WHITE PAPER

CONNECTING BUSINESS & TECHNOLOGY
SERVICE DELIVERY PLATFORMS: THE KEY TO SERVICE CONVERGENCE

DEVOTEAM WHITE PAPER

OCTOBER 2007
# Table of Contents

## 1. Introduction

### 1.1. Objectives

### 1.2. How to read this white paper

### 1.3. About Devoteam

### 1.4. About the SDP work group

## 2. SDP General Overview

### 2.1. Traditional service delivery models

### 2.2. Service deployment issues

### 2.3. SDP concepts

### 2.4. Layered architecture

### 2.5. SDP components

### 2.6. The standards behind the SDP

#### 2.6.1. OSA/Parlay

#### 2.6.2. OMA

#### 2.6.3. JAIN

### 2.7. Service creation and delivery

#### 2.7.1. Telco vs. Web service models

#### 2.7.2. Service Logic Execution Environment (SLEE)

#### 2.7.3. Performance concerns

#### 2.7.4. Service Creation Environment (SCE)

#### 2.7.5. Service testing

#### 2.7.6. Java in Communications Network Topology

#### 2.7.7. Java integrated development tools

## 3. The OSA Model Detailed

### 3.1. OSA/Parlay innovative services in different environments

#### 3.1.1. Mobile applications

#### 3.1.2. Fixed applications

#### 3.1.3. Convergent applications

### 3.2. Architecture

### 3.3. Description of the OSA/Parlay Framework

#### 3.3.1. Framework Access Session API

#### 3.3.2. Framework to Application API

#### 3.3.3. Framework to Enterprise Operator API

#### 3.3.4. Framework to Service API

#### 3.3.5. Minimum set of Framework interfaces

### 3.4. Description of the OSA/Parlay SCFs

#### 3.4.1. Call Control

#### 3.4.2. User Interaction

#### 3.4.3. Mobility

#### 3.4.4. Terminal Capabilities

#### 3.4.5. Data Session Control

#### 3.4.6. Generic Messaging

#### 3.4.7. Connectivity Manager

#### 3.4.8. Account Management

#### 3.4.9. Charging
3.4.10. Policy Management ................................................................................................................... 61
3.4.11. Presence and Availability Management ..................................................................................... 63
3.5. OSA and Web services 70
3.6. Web service concepts illustrated 71
3.7. Description of the Parlay X SCFs 75
  3.7.1. Third Party Call .......................................................................................................................... 75
  3.7.2. Call Notification ......................................................................................................................... 76
  3.7.3. Short Messaging ........................................................................................................................ 78
  3.7.4. Multimedia Messaging ............................................................................................................... 80
  3.7.5. Payment ..................................................................................................................................... 81
  3.7.6. Account Management ............................................................................................................... 82
  3.7.7. Terminal Status .......................................................................................................................... 82
  3.7.8. Terminal Location ...................................................................................................................... 83
  3.7.9. Call Handling ............................................................................................................................. 83
  3.7.10. Audio Call .................................................................................................................................. 84
  3.7.11. Multimedia Conference .......................................................................................................... 84
  3.7.12. Address List Management ..................................................................................................... 85
  3.7.13. Presence .................................................................................................................................... 86
3.8. Parlay promises and reality 88
3.9. Parlay's evolutions 89
4. THE JAIN MODEL DETAILED 90
  4.1. Architecture 90
  4.2. The JAIN APIs 92
    4.2.1. JAIN TCAP ................................................................................................................................. 92
    4.2.2. JAIN ISUP ................................................................................................................................. 92
    4.2.3. JAIN MAP .................................................................................................................................. 93
    4.2.4. JAIN Operations, Administration, and Maintenance (OAM) ...................................................... 93
    4.2.5. JAIN MGCP ............................................................................................................................... 93
    4.2.6. JAIN SIP .................................................................................................................................... 93
    4.2.7. JAIN INAP .................................................................................................................................. 94
    4.2.8. JAIN MEGACO .......................................................................................................................... 94
    4.2.9. JAIN H.323 ................................................................................................................................ 94
    4.2.10. JAIN JCC/JCAT ........................................................................................................................ 95
    4.2.11. JAIN Service Provider API for the Parlay Specification ............................................................. 95
    4.2.12. JAIN Service Creation and Service Logic Execution Environment ......................................... 95
5. THE OMA OSE MODEL DETAILED 96
  5.1. Open Service Environment principles 96
  5.2. OSE concepts, architecture and interfaces 96
  5.3. OSE controlled exposure of enablers and resources 98
  5.4. Using exposed resources in OSE 99
  5.5. Migrating from silo enabler architectures to OSE through Policy Enforcement 100
6. MIGRATION OF IN SERVICES WITH OSA 102
  6.1. IN-SDP evolution strategies 102
  6.2. IN call model evolutions 103
    6.2.1. Model mapping issues from traditional IN .................................................................................. 103
    6.2.2. Event triggering and notifications .............................................................................................. 104
© Devoteam 2007
6.2.3. Trigger priority handling ................................................................. 105
6.2.4. Call model management ................................................................. 106
6.3. SDP-IN message mapping at the OSA/Parlay gateway ................................. 107

7. IMS SERVICES AND THE OSA MODEL ................................................... 109
7.1. OSA in IMS reference architecture ....................................................... 109
7.2. OSA/Parlay gateway integration in IMS ............................................... 109
7.3. IMS service deployment strategies ....................................................... 110
7.4. IMS and the OSA Multi-Party Call Control model (MPCC) ..................... 112
  7.4.1. Trigger setting ................................................................................. 112
  7.4.2. Mapping OSA Call Session and Call Leg to SIP ............................... 113
  7.4.3. SIP Call-id &dialog vs. OSA Call & Call Leg Session ID ................. 113
  7.4.4. OSA Call and SIP Dialogue correlation table example .................... 114
7.5. SIP Server Roles in OSA SCS ............................................................. 115
  7.5.1. OSA SCS acting as a SIP Proxy server ........................................... 115
  7.5.2. OSA SCS acting as UA ................................................................. 116
  7.5.3. OSA SCS acting in redirect mode .................................................. 116
  7.5.4. OSA SCS acting as a B2BUA ......................................................... 117
  7.5.5. OSA SCS acting as a third Party Controller .................................... 117
7.6. Session description for application controlled calls ............................... 118
  7.6.1. Principles ..................................................................................... 118
  7.6.2. Example: OSA SCS Application initiated One-Party Call .......... 119

8. PERSPECTIVES: SDP CONCEPTS AND TELCO’S NEW BUSINESS MODELS 121
8.1. From voice transport to content delivery .............................................. 121
8.2. Innovating with fixed-mobile convergence ......................................... 122
8.3. Opening networks .................................................................................. 123

9. REFERENCES ......................................................................................... 125
10. SDP VENDORS ................................................................................... 126
11. CONTACTS ........................................................................................ 127

List of figures

Figure 1: Traditional deployment model .......................................................... 9
Figure 2: Benefits of an SDP architecture ....................................................... 11
Figure 3: SDP layered service delivery model .............................................. 12
Figure 4: SDP components .......................................................................... 13
Figure 5: OSA in 3GPP standardization ....................................................... 15
Figure 6: The OSA/Parlay Framework .......................................................... 17
Figure 7: The OSA/Parlay gateway in the network ....................................... 17
Figure 8: IN vs. IP-based triggers ............................................................... 20
Figure 9: A Java-based SCE example .......................................................... 25
Figure 10: Code sample with a Parlay SDK ................................................ 25
Figure 11: Example of SDP integrated test environment ............................. 26
Figure 12: Java in Communications Network Topology .............................. 27
Figure 13: Integrating JSLEE with J2EE ..................................................... 28
1. INTRODUCTION

1.1. Objectives

The Devoteam SDP workgroup has created this document to provide a thorough overview of SDP (Service Delivery Platforms) concepts.

The SDP concept brings together two worlds with different cultural backgrounds:

- Traditional telecom services, characterized by highly specialized ITU-T Intelligent Network protocols and their equivalents in the IP world (typically IMS), career-grade reliability, and high performance concerns. They are usually delivered in embedded real-time environments, and closed proprietary frameworks
- Operators’ information systems and more globally IT frameworks, driven by business services in open environments dominated by Java, Corba, and Windows.

As a result, there are few people who master both aspects.

However this need is growing as fast as the interpenetration of the Telecom and Internet worlds. It is felt by different actors in the SDP business with different profiles: decision makers needing to make sound strategic choices (business executives, consultants, marketers), as well as those more deeply involved technically (infrastructure architects, product line managers), and, of course, service developers, especially those who are accustomed to Web service designs, and who do not have a traditional service creation background.

This white paper then has the objective to answer questions about SDPs at every level, addressing issues ranging from business strategic considerations to in-depth technical reviews of the SDP standards. It also aims at being illustrative and concrete, and can be viewed as a tool box that provides many real and detailed service implementation examples based on SDP standards.

1.2. How to read this white paper

The document is designed to be adapted to different readers’ profiles. It is organized by sections, each with a different level of technical detail.

Section 2: SDP general overview, describes the general concepts behind the SDP and their applicability to operators’ business models. It can be read by anyone wishing to have an overview of the SDP concept and applicability in the field.

Sections 3, 4 and 5 provide a deeper technology overview of the standards behind the SDP concepts (OSA, JAIN, and OMA), including detailed descriptions of API, SCF, enforcement policies and associated message flows. These sections imply an understanding of the development processes involved.

Sections 6 (for IN) and 7 (for IMS) provide a broader technology overview of migration issues and twists in the role models when moving from a traditional service architecture to an SDP-based one, and could be attractive for network architects.

In conclusion, Section 8 proposes a longer-term perspective on the evolution of telecom operators’ business models and how SDP concepts can help meet these new challenges.

For those seeking further details, a list of major SDP vendors is provided, with their relevant SDP
product lines, as well as links to the main SDP reference sources, standards, and specifications. As is unavoidable when discussing this topic, a large number of abbreviations are used throughout the document. A complete glossary would list 150+ terms, and would provide limited value, because most of them would already be familiar to the reader. Because of this, only those that may not be immediately comprehensible to the reader are expanded and explained in the document, the first time they are used. All trademarks listed in the document are the property of their respective owners.

1.3. About Devoteam

12 years after its creation, Devoteam Group has become the number one consultancy and engineering company in information technologies in Europe, specializing in information system infrastructures. Combining consulting and technology solutions offers enables Devoteam to provide its customers with independent advice and effective solutions that meet their strategic objectives (IT performance and optimization) in complementary areas: networks, systems infrastructure, security and e-business applications.

Devoteam achieved a turnover of €267 million in 2006, growing by 34% over the period, with a €22 million operational margin, up 60%, amounting to 8.3% of turnover. After the acquisition of auSystems, the group counts 3,400 employees through 19 countries in Europe, Middle East and Africa. Listed on the Euronext Eurolist B compartment since October 1999, it is part of the NextEconomy, CAC SMALL 90, IT CAC 50, SBF 250 indexes of Euronext Paris.

In 2007, following the acquisition of the international divisions of auSystems formerly owned by Teleca, Devoteam’s telecom business unit counts 500 consultants in France alone, and over 1000 in Europe. Telecoms now represent 33% of the group’s activity.

1.4. About the SDP work group

Devoteam provides an efficient global knowledge management environment (knowledge communities, forums, document databases, workgroup collaboration tools, continuous education including e-learning courses, etc...), allowing consultants to share their experience.

The SDP group was formed by the Telecom knowledge community and led by François Mouillaud. Its purpose was to leverage the experience of Devoteam’s telecom community into a state-of-the-art study. The current version is a 2007 update of the initial version (released internally in May 2006). It has been reviewed, augmented and improved by several contributors in Devoteam group, including Devoteam Consulting, Devoteam SRIT, and the French division of auSystems, which has deep knowledge and current practice of service creation in OSA/Parlay environments.

As such, it gathers expertise from all companies in the group: market awareness, functional expertise through service definition, field practice with support and introduction in operational networks, and technological master through service development and integration in various environments (Intelligent Network, IMS).
François Mouillaud (francois.mouillaud@devoteam.com) has 17 years experience in the telecom industry and joined Devoteam Solutions in 2000. His assignments in the telecom services area include Sigtran M3UA introduction in a major service platform provider’s portfolio, support of SMS services in production and introduction of the Color Ring Back Tone IN service for a mobile operator, specification of a major evolution of the pre-paid IN service and MVNO offer for a leading manufacturer, and validation of Push To Talk services in IMS environment. An active community member, he created Devoteam University courses (SS7, IN service design training), and contributed to several others (NGN, UMTS). In 2006, he joined Devoteam SRIT, a Lannion-based division that bears telecom integration and development offers for operators and equipment providers. Since August 2007, he is co-leader of Devoteam’s Telecom knowledge community.

Patrice Crutel (patrice.crutel@ausystems.fr) was a pioneer in the introduction of OSA at auSystems, and an initial member of the group that launched OSA at Ericsson. As a technical expert on the subject, he created and delivered trainings on OSA/Parlay and was implicated in standardization groups. He is currently involved in the benchmark and development of major Parlay and Parlay X applications, such as fixed-mobile convergence or home zone, for local and global mobile operators.

Delphine Le Grand (delphine.legrand@ausystems.fr) has 8 years of experience in the telecom industry. She has been fully involved in all OSA activities at auSystems. She contributed to a course on the evolution from IN services to OSA/Parlay technology, and is currently involved in the benchmark, specification and development of major OSA applications, based on fixed-mobile convergence, for local and global, fixed or mobile operators.

Member of Devoteam Consulting, Hakim Bessila (hbessila@siticom.com) has 10 years of experience in the industry. He has been involved in network audits, fixed and mobile infrastructure cost reduction projects, and optical fiber deployment with several major industry accounts and network carriers. He also created a technical and market study on service platform evolutions for mobile operators.

Oliver Quaranta (olivier.quaranta@devoteam.com), project director at Devoteam Consulting, has 10 years of experience in Telecom services for operators. He has been working for Mobile operators in various value added services projects within SMS, voice and Wap portals products. As a consultant, he has deep experience in convergent offers and services for broadband and mobile access. He is currently involved in the Media field where he is creating multi device TV / VOD content services.
2. SDP GENERAL OVERVIEW

2.1. Traditional service delivery models

The following picture illustrates a simplified view of classic service delivery models in fixed and mobile telecom networks.

Services are represented by functional bricks, hosted by one or several platforms. If the platform has multiple network interfaces and is modular enough, the same service can be accessed by different equipments (e.g. SMSC and MMSC), and over different network types (fixed and mobile).

In the best reuse case, it could be shared by the telecom and IT infrastructures. Likewise, a service can be based on information available from both the network and IT domains. In the example below, Service 1 makes use of both the MPS (Mobile Positioning System), and CRM (Customer Relation Management) platforms. MPS provides the customer’s geographical location and CRM provides marketing details specific to the customer. Service 1 could be for example an instant messaging service that locates friends nearby.

![Figure 1: Traditional deployment model](image)

2.2. Service deployment issues

For most operators, when it comes to rapid service deployment, complying with the model depicted above is a real challenge because it requires overcoming many different issues:

- Traditional service creation follows a “vertical solution” for each vendor that limits interoperability
- Most of the time the network infrastructure has been engineered as the business was
growing, resulting in islands of heterogeneous architectures

- Each solution is customized to include network specifications specific to the operator. As a result, there is no standard toolset, architecture or hardware platform for the different systems. This situation makes it more complex to combine maintenance of existing services and development of new services
- The solutions deployed are strongly linked ("tightly coupled") with the network and the infrastructure, making it difficult to change one without the other
- Operators must deal with a wide variety of systems, each with its own unique access methods and behaviors. This makes it difficult to develop and deploy a new service in a standardized way, and can often make the potential introduction of a new service prohibitive from a cost and risk perspective
- Even if Intelligent Network fixed and mobile network standards are close, they are not truly compatible. Deploying a new service for both fixed and mobile networks usually requires duplicated architectures
- Where mergers and acquisitions have occurred, the result is often a multi-vendor network, with no standard method available to create new services. The problems above are multiplied by having several systems’ problems combined into one new organizational entity.

### 2.3. SDP concepts

Operators needing to address the issues highlighted above are looking for a solution through the introduction of a complete Framework that can meet all these needs. Because SDP is not a standardized concept, each provider has a specific solution, and there is no unique definition of the concept. The best way to describe it is to list what is expected from an SDP:

- Quick delivery of value-added services (reduced time to market)
- Reduced development costs (using off the shelf Integrated Development Environments)
- Simple service development path by using open standard interfaces that provide an abstraction for service creation (simplified APIs usable by a Web service developer unaware of the complex underlying protocols)
- Service portability over different vendors, networks, platforms ("write once, run anywhere" concept)
- Simplified network abstraction
- Integrated in network operator’s IT infrastructure or service provider, rather than being an element of core network infrastructure
- Interworks with other elements of the operator’s IT infrastructure such as service enablers (portals, SMSC, MMSC, IVRs, MPS, WAP servers), CRM, BSS/OSS elements and AAA systems, but also with core network elements such as switches, MSC, GGSN, etc...
- Provides a complete Framework for deployment, provisioning, execution, management and billing of value added services
• Supports multi-media types (voice, data, video, messages)
• Aggregates different network capabilities, services and content sources allowing application developers to access them in a uniform and standardized way
• Service exposure: open and secure access to service capabilities for use by external service providers and enterprises.

The benefit of using an SDP is illustrated in the following figure: the “middleware” introduced by the SDP Framework makes it more resilient to architecture or network element changes.

![Benefits of an SDP architecture](image)

Figure 2: Benefits of an SDP architecture

2.4. Layered architecture

The SDP concept is based on IT concepts such as layered architecture, levels of abstraction, and Service-Oriented Architectures (SOA). The layered service delivery model illustrates the separation of concerns, and can be depicted as follows:
The key principles behind layered service delivery are:

- **Support of business drivers**: the purpose of the SDP is to support the enterprise's business strategies and goals. IT systems can be rapidly pulled together and reconfigured to support business processes as they change in response to business drivers.

- **Separation of concerns**: introducing an additional interfacing layer presents technical systems as entities that business processes can use easily. The separation of concerns principle is met by the concept of service (as defined in Service-Oriented Architecture concept). Using services between layers permits better definition of deliverables, reuse increase and better consolidation.

- **Permit collaborative effort**: the layered service delivery model represents the need for service delivery to be a collaborative effort, allowing plugging in third-party content, enabling self-provisioning, and integrating a variety of technical OSS and BSS infrastructure parts.

### 2.5. SDP components

There is no fully agreed consensus over which elements compose an SDP. Some vendors use the term SDP for their application servers, whereas others include a whole portfolio of products and components. The following figure shows the typical SDP architecture:
In general, an SDP is seen as a commercial bundle of different products that can be offered by different vendors. It should be composed of the following elements:

- **Services Application servers** that provide deployment and execution environment for a broad range of voice and data applications. The Services Application servers are built on a standard J2EE or .NET. They can be also based on a standardized service execution environment such as JAIN SLEE or on a proprietary vendor-specific solution using J2SE or XML scripting. The Services Application servers are composed of Application servers that host the real-time part of service logic (call handling), of EJB or .NET containers that host the business part of the service logic such as provisioning and billing and of Web, WAP and VoiceXML servers hosting the service access (self provisioning and profile management) through WAP, Web, USSD, SMS or IVR access.

- **Network Abstraction Layer** that provides standardized interfaces to network service capabilities such as call management, multimedia and multiparty services, messaging, user location, presence and others. The Network Abstraction Layer provides standardized interfaces to core network services and ensures that this access is, as far as possible, network, technology and vendor independent. The Network Abstraction Layer usually utilizes emerging standards such as OSA/Parlay, JAIN, Parlay X, Web Services, OMA, IMS (SIP and SIP SIMPLE), as well as other standard protocols Internet such as LDAP, IMAP, SMTP, VoiceXML, etc.. The interfaces defined by these standards are usually exposed to the application developers as standard Java/J2EE components or other types of standardized
connectors like SOAP or .NET. The Network Abstraction Layer is usually implemented by a number of Gateways and connectors such as OSA/Parlay and Parlay X gateways or JAIN servers terminating various network protocols such as INAP, CAP, or SIP. Also included are Web Services gateways, SMS and MMS gateways and network connectors (such as Java Mail) providing APIs to native IP protocols such as IMAP, LDAP, LIF, MM7 and others. Today, the OSA/Parlay APIs provide a standardized abstraction layer to fixed, wireless, IP network types and is a mature standard that satisfies most of the requirements of a truly network independent Network Application Layer. OSA/Parlay is already supported by over thirty different vendors.

- **Web Service Exposure Layer** exposes service capabilities to third party service providers and enterprises. This layer allows the operator to "open" its network through a set of standardized and secure interfaces. Web Services technology can now be seen as the candidate for opening up the networks and exposing capabilities to external service providers and enterprises. The Parlay Group has created a set of Web Services-based interfaces known as Parlay X. The OMA is working on additional Web Services interfaces for selected telecom capabilities. Microsoft in cooperation with Vodafone has proposed its own set of Web Services interfaces.

2.6. The standards behind the SDP

The SDP concept is defined through the OSA/Parlay body, but also through the OMA and the JAIN bodies. Initially it might seem logical that these 3 bodies are in competition because they have the same objectives. However, this competition is now over and OSA/Parlay, OMA and JAIN are working in a cooperative manner. Nevertheless, the bodies have some differences such as:

- Parlay is independent in the API implementation through JAVA
- Parlay provides only services APIs. JAIN also provides APIs for network protocol communications
- Parlay provides an environment where 3rd parties could develop applications that run outside the trusted space, using security methods integrated within Parlay.

2.6.1. OSA/Parlay

OSA/Parlay is an application programming interface (API) that enables rapid telecommunications services creation by leveraging the high numbers of IT application developers to create telecom services. OSA/Parlay also enables enterprise applications such as field force automation (FFA), sales force automation (SFA) and bank branch automation to exploit capabilities in the existing wireless and AIN networks.

The OSA/Parlay APIs are defined by the Parlay Group (http://www.parlay.org/), which is a not-for-profit consortium of over 65 companies representing both the telecommunications and the IT industries. The Parlay Group was founded in 1999, and since then it has produced five versions of the Parlay Specifications, each one building on the last, and incorporating new requirements and feedback from implementers in the industry. Parlay Group member companies include Alcatel, British Telecom, Ericsson, Fujitsu, HP, IBM, Incomit, Lucent, NTT, Siemens, Sun, Telcordia Technologies, Telecom Italia, Teltier, and over 50 other companies from North America, Europe and Asia.
OSA or Open Services Architecture, refers to the architecture for mobile services developed by the 3rd Generation Partnership Program or 3GPP (http://www.3gpp.org/), and also adopted by 3GPP2. Parlay is the API portion of OSA. Note that the terms OSA, Parlay and OSA/Parlay are used on the document but refer to exactly the same specifications.

In the 3GPP standards, the three stages of OSA are specified by different working groups, as illustrated below:

Parlay’s history also has roots in ETSI standard bodies. In 2000, ETSI SPAN (Services and Protocols for Advanced Networks) was re-organized. The group ETSI SPAN12, Application Interfaces for service providers and network operators, was created. An activity in ETSI SPAN14, called Service Provider Access Requirements (SPAR), was also created. ETSI SPAN12, aware of the identical scope of the work in 3GPP CN5, agreed to work jointly and make all meetings joint meetings. Today ETSI SPAN12 has an OSA Project, is part of the Joint API Group, and is also working on OSA/Parlay Compliance.

ITU-T SG11 defined a Question 4 called API/Object interface and architecture for signalling, “covering the interface between network control and application layers”. ITU-T decided to write a reference document for this activity, and delegate the contents to other bodies. This way, ITU-T will adopt OSA specifications by ETSI (+3GPP +Parlay).

Today, the Parlay standardization can be seen as a remarkable joint effort by ETSI, 3GPP standard bodies, and a consortium of industry leaders. This is quite an achievement when looking back at the proliferation of Intelligent Network standards.

The OSA/Parlay APIs are technology independent. They were designed to be used for mobile networks, for fixed networks and for next-generation networks based on the Internet protocol. Equally, the OSA/Parlay APIs can be used by developers working in a number of programming languages, such as C, C++ and Java. OSA/Parlay is based on open standards such as CORBA, IDL, Java, UML, and Web Services (SOAP, XML and WSDL).
OSA/Parlay includes a comprehensive set of APIs for communications applications which include: Mobility, Location, Presence and Availability Management, Call Control, User Interaction, Messaging, Content Based Charging, and Policy Management. The capabilities offered by these APIs include:

- **Common Data Definitions**: the objects and types used throughout the Parlay specifications
- **Framework**: how applications authenticate themselves to the network. How applications discover what facilities are available from the network. Fault and load management
- **Mobility**: how applications find the location of a terminal. How applications request notifications when terminals change location
- **Terminal Capabilities**: how applications find out the terminal features
- **Data Session Control**: how applications manage data sessions initiated from terminals. This is typically used for GPRS (and other 2.5G) applications
- **Presence and Availability Management**: how applications manage presence “I am at my desk” / “I am away” and availability “I am in a meeting” / “I am available for contact”. Typically used in Instant Messaging types of applications and their extension to wireless networks
- **Account Management**: how applications query accounts and charge history
- **Charging**: how applications request payment for services (“content-based charging”). How applications reserve payment for a future service. This allows applications to work in a pre-paid network
- **Call Control**: how applications set up calls in the network. How applications set up multiparty (conference) calls in the network. How applications are used to route calls from the network, e.g. so that an enterprise application can manage call diversions for an employee. How applications can set up multi-media calls
- **Generic Messaging**: how applications interact with messaging systems, such as voice, FAX or email
- **Connectivity Manager**: how applications manage Quality of Service (QoS) and the configuration of Virtual Private Networks (VPNs)
- **Policy Management**: how applications interact with policy-enabled networks.

One of the key requirements from network operators and service providers since the inception of OSA/Parlay has been to ensure that opening up the network by defining an API did not expose the communications infrastructure to unauthorized use or other threats. This is the function of the OSA/Parlay Framework. The Framework is implemented in the OSA/Parlay Gateway, which is described in more detail in the following chapters. All applications and services that want to use the OSA/Parlay APIs first have to register with the OSA/Parlay Framework. The OSA/Parlay Framework is a software component that cryptographically authenticates the application, and returns object references to the application for those OSA/Parlay functions (or service capability features) it has been allowed to use by the service provider. The Parlay Framework solves a number of security and availability problems.

---

1 However, by using a formal method and a model-checking security protocol test developed by the University of Twente, Italian and Dutch researchers have shown that the OSA model has security flaws. The detailed discussion can be read at http://eprints.eemcs.utwente.nl/698/
The OSA/Parlay model adds a new network element - the OSA/Parlay Gateway - which is used to link applications using the OSA/Parlay APIs with the existing network elements. The OSA/Parlay Gateway is under the control of the network operator or service provider, and is a single point through which all OSA/Parlay interactions pass. This means that applications are isolated from the specific protocols used within the network by the operator, and networks can be evolved without affecting existing applications and services. The OSA/Parlay Gateway is the element that implements the Parlay Framework.

2.6.2. OMA

The Open Mobile Alliance (OMA) was set up in June 2002 about the same time as 3GPP was finishing UMTS R.5. Around the Millennium there had been an increasing feeling in the mobile industry that the process was becoming too fragmented for specifying advanced mobility applications & services, such as those required for mobile commerce and associated security techniques like PKI (Public Key Infrastruc-
turse). Greater interoperability would be required if these new market sectors for mobility were to take off and achieve critical mass for profitable deployment in the market place. The OMA was formed as a combination of the old WAP forum which had been merged with the Location Interoperability Forum (LIF), the SyncML Initiative, the MMS Interoperability Group (MMS-IOP), the Wireless Village Initiative, the MWIF and the Mobile Games Interoperability Forum MGIF). Another foundation stone for the OMA “raison d’être” was to bring together both the major cellular communities (3GPP and 3GPP2) with the IT Industry in the OMA through the participation of key companies and by formalizing a cooperation agreement.

The OMA's overall intention is to make use of existing Standards, whether they originate from the Telecom or IT industry, and to then build end-to-end interoperability solutions. These solutions have three phases:

- **Phase 1:** An approved set of open technical specifications forming an enabler that can be implemented in products and solutions
- **Phase 2:** Specifications must pass interoperability tests
- **Phase 3:** OMA Interoperability Release is approved with end-to-end test reports, verifying end-to-end interoperability.

The OMA OSE (Open Service Environment) architecture is composed of:

- **Applications** (on an in-house platform or a 3rd party application)
- **Policy enforcer.** This function applies policies to the interaction between the application and the enablers and between enablers wherever applicable
- **Enabler.** An OMA Enabler contains intrinsic functions which can interact with other functions, within the domain of the architecture and underlying network resources
- **Execution environment.** This deals with aspects such as service management, load balancing, O&M, infrastructure.

The interactions between these components are further detailed in the OMA OSE section.

### 2.6.3. JAIN

The JAIN (Java APIs for Integrated Networks) initiative represents a community of communications experts defining the necessary Java interfaces as an extension of the Core Java platform to migrate proprietary communications networks to open standardized based networks. Development is carried out under the terms of Sun's Java Specifications Participation Agreement (JSPA) and the Java Community Process (JCP™).

JAIN is composed of over 80 member companies specifying over 25 new APIs that target converged IP, fixed and mobile networks. JAIN includes APIs for high-level application development (e.g. Service Providers APIs...), Call Control, as well as protocol level APIs for signaling (SS7, SIP...).

The objective of the JAIN initiative is to create an open-value chain from network equipment providers, computer and device vendors to 3rd party service, facility-based service, and communications service providers to consumers. This open value chain reduces the proprietary and specialized interfaces down to a manageable set of standard interfaces. This also allows reduced cost and
development complexity, deployment and maintenance of the communications and services. The scope of the JAIN initiative is to unify complex fixed, mobile and IP communications interfaces into a set of industry defined Java standards. The JAIN program has 3 main objectives:

- **Service Portability:** JAIN provides open and standardized APIs that allow services to run on different JAIN compliant equipments
- **Network independence:** JAIN provides abstracted APIs that allow applications to run and behave in the same way no matter what the underlying technology is (fixed, mobile, SS7, IP…). Network independent Java technology also allows service migration from traditional telephony networks to Internet networks to be turned into a straightforward task
- **Open development:** the goal of the JAIN initiative is to change the telecommunication market from many proprietary closed systems to one open environment, unlocking value similar to the effect triggered by J2EE in the IT industry. By opening the network to Java, network operators can extend their service portfolio, and make next-generation communications application development faster, simpler and more cost-effective.

The JAIN initiative currently consists of 2 areas of Java API specification development:

- **The container interfaces** specify services’ execution environment APIs geared at, but not exclusive to, the communications domain
- **The application interfaces** specify service APIs that enable service development across fixed, mobile and IP data networks.

The JAIN APIs address the signaling, call processing, and service layers in the telecom scope. They are optional extension packages to J2ME, J2SE or J2EE. The “OSS through Java” APIs (OSS/J) address the operation support system interfaces (back office systems) and are optional extension packages to J2EE. JMX defines management interfaces that can be deployed in any industry wherever management is needed. JMX is an optional package additional to J2SE. JAIN and OSS/J take advantage of JMX wherever possible for underlying management capabilities. So, rather than replicating JMX functionality, JAIN and OSS/J use JMX and build from it.

Note that, unlike JMX, JAIN and OSS/J, JDMK is not an API or a set of APIs defined trough the JCP but it is a product by SUN. JDMK is the implementation of the JMX API specification.

### 2.7. Service creation and delivery

#### 2.7.1. Telco vs. Web service models

The “telco” world of the 1990’s was dominated by the Intelligent Network paradigm and protocols. Although it had a huge success by breaking the limits of switch-based services, it has inherent weaknesses that make it less operational in the landscape of the IP-centric, Internet dominated 2000+ years. Among these weaknesses are:

- The inability to deliver or control the delivery and charging of a specific content, since “telco” protocols only deal with signaling and call control. The exception of the SMS services
(160 characters of plain text), shows limits over age which the MMS only partly overcome.

- The rigid model of the BCSM (Basic Call State Model, the state machine that describes the progress of a call in an Intelligent Network) does not well fit in environments where the need arises to convey information outside of a call or session, or with more elaborate trigger detection criteria, for example content-based.

- The telco-oriented architecture was not designed to be integrated or interoperable with the enterprise software environment (Information Systems, ERP, etc.), the broad range of available Web services, or the large population of skilled Web designers.

- It relies on limited service profile features (such as those provided by an HLR), and can not benefit from the rich customer profile knowledge located in the Information System.

This opens the door to an open API environment that would bring together features such as:

- Aggregate the subscriber's static and dynamic configuration into a ‘user profile’. This profile would include a service list, a detailed status including location and device availability, QoS, security, privacy, charging and billing policies.

- Operate in an IP-centric environment (SIP, TCP-IP).

- Leverage the existing capacities of the legacy IN features (detailed call status, subscriber state, location) to provide value-added services, e.g. location-based routing.

- Deliver and activate services dynamically and provide service interworking functions.

- Provide an easily chargeable a set of rich multimedia services accessible in background.

In this type of environment, a completely enriched set of services could be introduced, as represented by the examples in the following table:

<table>
<thead>
<tr>
<th>Service</th>
<th>With IN triggers</th>
<th>With IP Service profile</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content delivery</td>
<td>Limited (SMS, MMS, audio file...)</td>
<td>Multimedia, interactive (gaming)</td>
<td>Interactivity with user, detailed content charging</td>
</tr>
<tr>
<td>Corporate features</td>
<td>PBX-like features: VPN, DID (Direct Inward Dialing)</td>
<td>Interaction with CRM environment, wireless-based PBX</td>
<td>PBX and VPN features extended to the IT and mobile environments</td>
</tr>
<tr>
<td>Group features</td>
<td>Multi-party calls (group call, conference, call waiting)</td>
<td>Push To Talk (half duplex conferences)</td>
<td>Multimedia messaging, charging based on usage not on duration, broadcast facilities</td>
</tr>
<tr>
<td>Calling identification</td>
<td>Calling Line ID. limited name presentation</td>
<td>Multimedia identification based on user profile</td>
<td>Increased content value based on rich user profile, filtering, multi-networks</td>
</tr>
</tbody>
</table>

Figure 8: IN vs. IP-based triggers
2.7.2. Service Logic Execution Environment (SLEE)

One can view the IN model (end user, switch, service control point) as a precursor of a 3-tier architecture that was later adopted by the software industry. This model extended the switch-centric black box view by deporting the service logic and making it available to other vendors, especially application providers independent from the network equipment manufacturers like Ericsson, Alcatel or Siemens. However, only a few independent providers had a significant market share, and those who did were industry leaders such as HP or Logica (now LogicaCMG), not the kind of start-ups we observe in the Web industry.

This fact is illustrated with the Service Control Point as it is composed of three distinct layers:

- The services, possibly built from common base elements, but still independent from one another
- The SLEE (Service Logic Execution Environment)
- The network layer, allowing the SCP to communicate with network elements with protocols such as INAP, ISUP, H.323, or with external systems (SNMP, TCP/IP, FTP,..).

The entry level in this model is the SLEE. It allows services to be deployed, activated, upgraded, managed, inter work. It also provides low level functions to services like message distribution, timers, access to physical resources, databases, counters, statistics, fault management, and carrier-grade services such as fault tolerance, redundancy, load sharing, replication, overload handling, etc…

In the Telco model, the SLEE is a monolithic environment closely tied to the manufacturer's technological options (OS, hardware, etc…). It is completely proprietary to each vendor: a service designed in HP Open Call environment will be unusable by an Alcatel OSP platform. What's more, the two (each running in its own SLEE) will not be able to interoperate.

In an open world, the challenge of interoperability is to turn the SLEE into a standardized execution environment able to host services from different vendors. An interoperable SLEE should have the following features:

- Service portability over different SLEE vendors supporting the standard, through standardized APIs, objects and methods
- Independence from OS, hardware, platform, network architecture
- Common Framework providing the generic services of a SLEE, as described above
- Modular architecture, allowing interoperability with legacy, state-of-the-art, and evolution towards still-to-come service environments (network interfaces, provisioning, OSS, billing and management systems).

As of today, one environment meets these requirements: the JAIN SLEE Standard, specified by the Java community through the JCP (Java Community Process¹). Besides the features listed above, it can smoothly integrate to legacy and modern environments. For example, it provides:

¹ http://jcp.org/en/jsr/detail?id=22
• Interface to OSA/Parlay gateway for connection to an operator’s secure environment
• Integration with Web services operating in a J2EE environment
• Management interfaces using JMX (Java for Management Extensions)
• Smooth migration between legacy (SS7) and NGN (GPRS, IMS) network environments.

The JAIN SLEE standard can be seen as a tool box for building a service execution Framework. But the implementations and architectural options will be vendor specific. These options can be classified in three main areas:

• Career-grade features (reliability through redundancy, failure recovery, scalability, load distribution, overload control, etc.)
• Support for various network protocols, databases, directory servers, and ability to integrate new ones quickly
• Integration within operators’ environment (management, alarms, statistics).

2.7.3. Performance concerns

SLEE performance has always been a key issue in traditional telecom services. There are several reasons behind this:

• A service platform is generally unique in the network and plays a central role in the architecture (unless, say, a MSC that plays only a geographically limited role)
• Some services like number translation, routing, and pre-paid are highly solicited because they can be triggered for each call
• Services listed above are vital to the simple call establishment: failure to complete a pre-paid platform interrogation results in call failure, and immediate loss of revenue
• A poor TPS (Transaction Per Second, the key measurement unit in this area) performance directly translates into more telecom signaling cards (SS7), a costly hardware which also has impacts in the network architecture
• For some services, especially those requiring interactive voice processing, and therefore establishment of switched voice circuits, the scalability issue can be even more expensive. Failure to deliver the required performance may require duplicating not only signaling cards, but also the E1 cards holding switched circuits. In addition, this has an impact on the corresponding MSC (more E1 cards), and on the transmission chain.

The question of how an SDP can meet these requirements can be addressed at different levels:

• The SDP Framework, i.e. the middleware that delivers environment services (including the OSA/Parlay gateways), is even more central than an SCP in an IN architecture, because all services need to access it. It is clearly the element that requires the most careful performance planning

---

2 The TPS measures the time interval between the reception of a TCAP primitive from the network and the emission of the TCAP response that marks the end of the processing by the service logic
The application servers that host new, Web-type of services typically require non real-time processing, at least not to the career-grade extent. With the decrease of processing power and decline of its cost, Web services, IT infrastructure and user interaction in asynchronous services like messaging services can be well achieved in a Java Web-based environment, possibly on Windows platforms. These services would be typically co-hosted on the same platform.

The application servers that will fulfill current mission-critical services but in an SDP environment should be hosted on mission-critical platforms with sufficient performance figures, and high availability architectures.

It appears that a clear distinction should be made between the application and the SDP Framework. The former is not designed for performance and delivers what the general IT Framework can offer in a non real-time environment (SOAP with Parlay X, CORBA with Parlay are the typical paradigms here). The latter, even if allowing communication with non real-time Web services, requires very high performance and high-availability design, even more so than a traditional SCE due to his unique position in the architecture, and, consequently, its numerous network interfaces. An efficient JSLEE allows these needs to be met.

Because network interfaces in the SDP model are all IP-based, except for the legacy interfaces, it is possible to consider a scalable approach that does not translate into corresponding cost increase in signaling cards.

The proliferation of the SDP concept in the network can also be combined with a transport of SS7 protocols over IP using the SIGTRAN family of protocols (such as M2UA: MTP2 User Association, M3UA: MTP3 User Association, and SUA: SCCP User Association), that permit backhauling the processing of SS7 levels from the node that has direct network access to the node that actually processes signaling information (typically the OSA/Parlay gateway and the AS in this scenario). SIGTRAN protocols are now well defined, have all become RFCs, passed interoperability tests, and are widely integrated in SDP frameworks.

### 2.7.4. Service Creation Environment (SCE)

The Service Creation Environment can be viewed as a toolbox where the service developer will find all that’s needed to design the service. In the “classic” telecommunication world, services could be based on SIBB (Service Independent Building Blocks) which are modular representation of service enablers, on a development language, an SDL representation of state machines (the traditional and most efficient way to program IN services), automatic delivery of continuous integration and testing tools based on SDL representation and simulators with the TTCN suite, a development tool with a GUI, etc.. But as it turns out, the key point is that a vendor’s SCE is necessarily tightly coupled with a SLEE, since the service compiled with the SCE will have to be executed by the SLEE.

---

1 For more details on SIGTRAN charter and RFCs: [http://www.ietf.org/html.charters/sigtran-charter.html](http://www.ietf.org/html.charters/sigtran-charter.html)
This monolithic landscape is breaking apart in the world of Web services and its integration in the Web and IT environments. The answer, it appears, will be based on Java, possibly by evolution of the existing Integrated Development Environment (IDE) supplied by the vendor.

For example, HP, one of the leading service development platforms suppliers with its Open Call portfolio, merged its Open Call Media Platform (OCMP) by integrating the extensible Eclipse platform with the OCMP software developer’s kit. This simplifies the development of voice-based telephone services in a single, integrated, familiar Java environment.

The power of Java in state-of-the-art service delivery comes from JCP-driven industry standards offering built-in portability over numerous OS and platforms, vendor independence, abstraction of lower layers, Web services integration, and powerful development environments.

Leveraging on the trend towards platform independence symbolized by the paradigm “Write once, run anywhere”, Java enables monolithic service creation architectures to evolve into a modular and reusable architecture that both reduces costs and simulates growth.

Java 2 Enterprise Edition (J2EE) provides Java network platforms with a complete suite of components that meets all the needs of an SDP:

- OSS/BSS Platform – J2EE – and Interfaces – OSS through Java –
- Web services Platform – J2EE –
- Network Services Platforms – JAIN SLEE – and Interfaces – JSIP, JCC, SS7 –
- Service Creation – Java Developer tools –
- Mobile Devices Platforms – J2ME –

J2EE is a standard for enterprise applications. It offers Java Database Connectivity (JDBC) API for database access, CORBA technology to interact with enterprise resources, Web services interoperability, Java beans components (EJB), Java servlet API, Java server pages, and XML technology. EJB is a technology that allows the modeling of the objects in the enterprise, in three types: session beans, entity beans, and message-driven beans. Session beans are associated with client sessions, entity beans represent a collection of data and encapsulate actions on these data, and message-driven beans allow applications to process messages asynchronously.

OSS through Java (OSS/J) provides a component-based approach to develop OSS Solutions. Its interfaces are similar to business abstractions available in a typical environment and are modeled through APIs based on J2EE, XML and Web services. In addition, OSS/J provides design patterns, best practices, certification processes, and solution patterns.

These technologies enable service abstraction and management abstraction, two key components of service delivery and integration into an existing infrastructure. The enterprise or network layer may evolve separately and transparently from a legacy architecture (supported by Java through the Resource Adaptor technology) to the state of the art.

In the legacy Telco world, Service Creation Environments (SCE) were often SDL-based graphical environments customized for the vendor that, from an SDL description of the service, generate the associated code. Although this was well suited for IN applications in a closed environment, it does
not fit at all in the open world of Web services. Besides, it requires deep knowledge of the service and protocol mechanics behind the network APIs (typically the TCAP API) delivered with the tool, a step that many IT developers are not willing to make.

In an SDP, they would be Java integrated environments like the following, allowing smooth integration with classes objects, libraries or packages coming from existing IT environments:

The next step is the usage of a dedicated Software Development Kit (SDK). A broad range of SDK for Parlay application development are now available. They can be plugged into popular IDE such as Lucent eSAE, Java Forte, Eclipse, BEA web logic, Oracle Jdeveloper, JBuilder, etc. They support developers’ work with class libraries, gateway and network simulators, wizards and templates, and provide a very significant reduction of the programming effort.

The following example shows how easy it becomes to create a service reference using an adapted Parlay SDK:

```
FrameworkAdapter fwAdapter;
UserLocationAdapter ulAdapter;
// Step 1: create a Framework adapter using valid authentication credentials
fwAdapter = FrameworkAdapterFactory.createFrameworkAdapter("AppID","KEY");
// Step 2: get a reference to the required service
ulAdapter = fwAdapter.selectUserLocationService();
```

Figure 9: A Java-based SCE example

The next step is the usage of a dedicated Software Development Kit (SDK). A broad range of SDK for Parlay application development are now available. They can be plugged into popular IDE such as Lucent eSAE, Java Forte, Eclipse, BEA web logic, Oracle Jdeveloper, JBuilder, etc. They support developers’ work with class libraries, gateway and network simulators, wizards and templates, and provide a very significant reduction of the programming effort.

The following example shows how easy it becomes to create a service reference using an adapted Parlay SDK:

```
FrameworkAdapter fwAdapter;
UserLocationAdapter ulAdapter;
// Step 1: create a Framework adapter using valid authentication credentials
fwAdapter = FrameworkAdapterFactory.createFrameworkAdapter("AppID","KEY");
// Step 2: get a reference to the required service
ulAdapter = fwAdapter.selectUserLocationService();
```

Figure 10: Code sample with a Parlay SDK
2.7.5. Service testing

Testing can represent a significant portion of a service design, especially when it requires buying specific hardware to deploy the service developed and test it, and simulate the network where the service is supposed to fit.

In Intelligent Network environments, vendors (such as Telelogic, Tekelec) have developed packaged test tools that can simulate network elements, create and execute test scenarios, using an SDL-based model where events from all network elements can be simulated, and automate test suite execution.

The same issues arise in SDP environments, with the added constraint that the network infrastructure (e.g. the OSA/Parlay gateway) may not even exist at the time the service is developed, because, at least initially, introducing an OSA-based service implies deploying OSA/Parlay network bricks. Companies like Open API solutions provide test environments that can, like in the IN world, simulate the presence of these bricks, as well as other core elements of an SDP architecture like an SDP Framework, and service enablers (charging, mobility, messaging, call control,…). The suite also delivers a complete test environment, allowing the designing, executing, and automating of test scenarios. The following screenshot gives an example of such an environment.

Figure 11: Example of SDP integrated test environment
2.7.6. Java in Communications Network Topology

Many Java technologies are required to provide a complete communications solution. Container technologies, such as JSLEE and J2EE, are the core part. However, other Java technologies are needed to deliver complete solutions. The following network topology brings out the essence of a typical and recommended Java in communications solution. It draws on mainstream Java Mobile, Desktop and Enterprise technologies, namely J2ME, J2SE and J2EE, as well as emerging Java Communications and OSS technologies, such as JSLEE, JSIP, SAMS and OSS/J. Such a network topology provides a way to visualize how these various technologies relate to each other within the communications domain. Network deployment and implementation options are not considered here.

As shown in the previous picture, central to the Java in communications network topology is the JSLEE. It is the JSLEE that forms the core of the Network Value-Add Service Provider (NVASP), permitting applications, or Service Building Blocks (SBBs) to access and manipulate network elements (network resources). Access to the network elements is only made possible through the Resource Adaptors (RAs). RAs provide a common architecture for the JSLEE to interface with the various different types of network elements (e.g. Signaling Switching Point (SSP), Multimedia Messaging Service Center (MMSC), and Gateway Mobile Location Center (GMLC)). Key RAs are those for JSIP, JCC and SAMS, where the RAs are derived from their respective service APIs. Other RAs may be derived from interfaces from other organizations (e.g. 3GPP/3GPP2 OSA) or companies.
The RAs (or network element enablers) communicate with the network elements via network protocols, such as Intelligent Network Application Part (INAP), Multimedia Messaging reference number 7 protocol (MM7) and Mobile Location Protocol (MLP). Many different types of network elements are expected to be deployed within a network offered by the Network Service Provider (NSP), and it is these network elements, the network sub-elements, the access networks and the operations support systems that make up the fundamental network capabilities offered by the NSP.

The SBBs executing within the JSLEE are software components that can be assembled to provide network services ranging in complexity from simple to elaborate. Orchestration of these services can be done entirely within the JSLEE or via an external interface. The benefits of providing orchestration within the JSLEE would include predictable service security, reliability and performance. The benefits of providing orchestration outside the JSLEE would include greater integration with IT systems and service accessibility.

The most suitable platform for providing greater service accessibility and integration would be J2EE, since it provides Web Services Interoperability (WS-I) Basic Profile 1.0 support and integrates easily with traditional IT systems, thus, a mixture of communications and IT services would be offered by the Web Service Provider (WSP). Within the WSP, the J2EE would permit applications, or Enterprise JavaBeans (EJBs), to orchestrate SBBs possibly into an Enterprise Service (service hosted within the Enterprise) or a Web Service (service exposed through the Internet), both of which would incorporate network capabilities. Access to the SBBs is done via the J2EE connectors, as shown in more detail in the following figure:

![Figure 13: Integrating JSLEE with J2EE](image)

If the WSP and NVASP were not co-located, it is possible that communication between the two entities would be done over the Remote Method Invocation over the Internet Inter-Orb Protocol (RMI-IIOP), however, if they were co-located, it is possible that a high-speed proprietary protocol might be used. The Web Services would be offered by the WSP through the Web Services “Publish” mechanism, and
accessed by the Web Service Requester (WSR) embedded within the Consumer’s device. The choice of Consumer device could vary widely; however, the two most popular device categories would be mobile devices and desktop devices, which would typically host the J2ME and J2SE platforms, respectively.

Applications executing in the Consumer device would access the Web Services through the Web Services “Find” and “Bind” mechanisms, using SOAP, for example. Basic network connectivity and “line” based services, such as sending an SMS, offered by the NSP would be accessed by the Consumer device’s Network Access capability via the Consumer device APIs, such as SIP for J2ME, and WMA, using an access protocol like SIP. Internet access to Web Portals for functionality such as customer self care would be done via classic Internet protocols, like HyperText Transfer Protocol (HTTP) or HTTP with security (HTTP(S)).

The operation of the various network elements, network value-add elements and Web elements contained within the NSP, NVASP and WSP, respectively, could all be supported by the OSS management applications, which also would run within a J2EE platform. This network topology describes only part of the Java in communications architecture.

Other Java APIs exist for network identity, data presentation, data formatting, service creation, content creation, content delivery, and data storage and access, etc. However, what has been exposed is how Java fulfills the roles of service delivery and operations support in next generation networks.

2.7.7. Java integrated development tools

When the development is based on Java environments, it benefits from the integration of the service creation and execution environments. The following figure illustrates the development chain in Parlay and Parlay X when a specific SDK is introduced (note the SDK can be itself integrated within an existing IDE).
3.1. OSA/Parlay innovative services in different environments

OSA/Parlay provides a range of network-agnostic APIs, making the standard not only applicable to fixed and mobile networks, but also to existing (and future) next generation network architectures. These APIs offer rich functionalities, enabling the creation of a wide range of services in a variety of technologies, with detailed access to the capabilities available in various network types. In addition, the Parlay X APIs provide a much simpler abstraction of communication capabilities targeted at Web Service Application development.

This rich feature set combined with Parlay’s ability to provide convergent applications between fixed, mobile and IT networks creates a vast market of opportunities where the only limit seems to be our imagination. Before going deeper in the detail of the APIs, this section provides a quick overview of the type of innovative services that can be designed with Parlay APIs.

3.1.1. Mobile applications

Reverse Charge Calling
Prepaid subscribers with no credit can use an OSA application to make reverse charge calls towards other users on the same network. The tariff of such calls will typically include a premium. Presence-sensitive Click-to-Conference MS Outlook based plug-ins for telecommunications services can be provided with presence information by OSA applications using Parlay-X capabilities. For example, if an attempt is made to conference a range of numbers into a call from the desktop, then pop-up windows can display information on which mobile users are currently active.

Virtual Mobile PBX
Many mobile operators have successfully launched Virtual Mobile PBX systems in order to replace their PBXs with a set of mobile handsets. Calls to the company number can be routed to a mobile handset for the receptionist and simultaneously, pop ups and click events on the desktop can allow the receptionist to transfer the call to employees as they would with a traditional PBX.

Inbound Call Management
There are now a range of OSA applications that allow mobile users to effectively manage incoming calls. For example, during meetings, business users can be pushed a GPRS or USSD menu with a set of options instead having of the handset ringing. These options may allow the user to automatically send a message or play an announcement to the caller requesting that they call back later.

Mobile Call Completion to Busy Subscriber
In the fixed line world, Call Completion to Busy Subscriber (CCBS) allows users to “camp on” a busy line until it becomes available. OSA applications provide similar functionality in mobile networks such that a call can be automatically created when a subscriber becomes available again in order to generate more minutes in the network.
Location-Based Services
There are a wide range of Location-Based Services available today. Several of these use the OSA standards to access location information in the network since the same OSA interface can be used to manage call control, messaging features, etc. OSA Location-Based Services are particularly compelling as the mobility functionality can be associated with several other network capabilities.

3.1.2. Fixed applications

Internet Call Waiting
Internet Call Waiting is well-understood functionality that was designed in the pre-ADSL low-rate Internet ages, when a choice had to be made between picking up the call and leaving the network, or staying connected and missing the call. Users connected to the Internet could be notified of an incoming call by their ISP, through a screen pop-up that provided options to deal with the call. Several OSA implementations of this application are commercially deployed today that use the network independence of the technology in order to terminate incoming calls either on alternative mobile numbers or on a PC-based application in voice over IP.

Wholesale Voice
Deregulation in many markets has seen former monopoly fixed-line operators open up their networks to Carrier select and pre-select competition. Rather than competitive service providers purchasing their own switch and a regulated interconnect agreement, wholesale voice OSA applications allow them to lease switching capacity as a commercial managed service.

Network Call Center
Call centers are a highly competitive market segment with many service providers that are new entrants to the market targeting these customers. OSA fixed-line applications are available that provide management screens for call center organisations to configure inbound call load distribution across multiple sites. Typical PBX and CTI solutions can also achieve this, but there are costs associated with forwarding calls in this manner. A network-centric solution allows operators to gain more market share in the call center segment by significantly reducing the operational costs associated with load management between sites.

3.1.3. Convergent applications

Video Color Ring Back Tone
There are now a large number of commercially successful Color Ring Back Tones applications that allow music to be played to callers instead of ring back tone. An OSA-based approach allows the technology to be seamlessly migrated to take advantage of more advanced capabilities as they become available. For example, IMS architectures in mobile networks can provide streamed video to callers instead of music.
Scheduled Conference Calls
Network operators typically charge premium rates for conference calls. OSA applications are available that provide plug-ins to enterprise systems (such as MS Outlook) and use Parlay-X interfaces in the network to provide a much more user-friendly mechanism to arrange conference calls. This encourages greater use of such services based on the convenience of having the network create the call compared with traditional systems that require users to navigate announcement menus and remember passwords. There is an added benefit of exposing the network operator’s brand on many desktops in the market.

Mass Market VPN
Traditional IN systems are good at offering large-scale VPN systems towards the corporate segment but these products are typically not suitable for the mass market. OSA-based VPN systems can be aimed at a family or a community and provide a number of features such as on-net SMS broadcast, cheaper on-net calls, call barring options for the VPN owner, etc. These systems can be charged on a subscription basis and allow prepaid users to be included.

Call-Related Data
OSA applications can generate data traffic, based on specific call events. For example, a call missed by a corporate user can result in a MMS push to the caller with the company logo and a link to the corporate address book. Alternatively, a wireless Yellow Pages solution can be provided so that when a service company is called (for example, pizza delivery, hairdresser, etc.) an advertisement can be pushed to the caller.

Multi-Channel Televoting
Televoting systems are well-understood products. There are OSA-based solutions that provide multi-channel support, based on the broad horizontal scope of the OSA interface. Consequently, a single consistent OSA application can count votes made by IVR, SMS, USSD, WAP, etc. The voting mechanism can also be location-sensitive with results available to media organisations in real time.

Message Broadcast
Many mobile handsets can currently be programmed to generate SMS broadcast events but this is not a user-friendly process. Available OSA applications allow users to access Web-based provisioning screens to create broadcast lists. The broad functional scope of the OSA standards then allows a single message to be delivered to multiple recipients that may have a range of different messaging capabilities on their handsets.

Virtual Contact Numbers
In many markets, there is a range of publications in which people can insert personal messages or advertise items for sale. Typically, this requires a telephone number to be included, which can lead
to nuisance phone calls. As part of a broad set of OSA applications that target niche markets, there are products available that allow a number range to be associated with a specific publication or other media outlets. Web-based management screens then allow these publications to assign a virtual contact number from their number range to personal advertisers, before recycling this number for later use once the advertisement has expired.

3.2. Architecture

Open Service Access (OSA) enables applications implementing services to make use of network functionalities, defined in terms of a set of Service Capability Features (SCFs). These SCFs provide network capabilities accessible to applications through the standardized OSA interface for service development. They have been described by the 3GPP using UML and come in three realizations (OMG Interface Description Language™, Java J2SE™ and Java J2EE™). They are open and accessible to application developers, who can design services in any programming language, while the underlying core network functions use their specific protocols. Popular examples of SCFs include Call Control and User Location.

Generally, the OSA architecture is composed of:

- Applications such as VPN, conferencing, location based services, etc…
- A Framework enabling applications to use the SCFs, through the Authentication, Registration and Discovery methods defined in the OSA interface
- Service Capability Servers providing the applications with SCFs, which are abstractions from underlying network functionality.

![Figure 15: The OSA/Parlay Overall Architecture](image)
3.3. Description of the OSA/Parlay Framework

The Framework API contains interfaces between the Application Server and the Framework, between the Network Service Capability Server (SCS) and the Framework, and between the Enterprise Operator and the Framework (these interfaces are represented by the yellow circles in the diagram below). The description of the Framework in the present document separates the interfaces into these three distinct sets: Framework to Application interfaces, Framework to Enterprise Operator interfaces and Framework to Service interfaces.

![Diagram of OSA/Parlay Framework Relationships](image)

Some of the mechanisms are applied only once (e.g. establishment of service agreement), others are applied each time a user subscription is made to an application (e.g. enabling the call attempt event for a new user).

Basic mechanisms between Application and Framework are:

- **Authentication**: This is done by using a peer-to-peer model but not necessarily mutual. Once an off-line service agreement exists, the application can access the authentication interface in order to use any OSA interface.
- **Authorization**: Authorization is distinguished from authentication in that authorization is the action of determining what a previously authenticated application is allowed to do. Authentication must precede authorization. Once authenticated, an application is authorized to access certain service capability features.
- **Discovery of Framework and network service capability features**: After successful authentication, applications can obtain available Framework interfaces and use the discovery interface to obtain information on authorized network service capability features. The Discovery
interface can be used at any time after successful authentication

- **Establishment of service agreement**: Before any application can interact with a network service capability feature, a service agreement must be established. It may consist of an off-line (e.g. by physically exchanging documents) and an on-line part. The application has to sign the service agreement on-line part before it is allowed to access any network service capability feature.

- **Access to network service capability features**: The Framework must provide access control functions to authorize the access to service capability features or service data for any API method from an application, with the specified security level, context, or domain.

The basic mechanism between Framework and Service Capability Server is:

- **Registering of network service capability features**: SCFs offered by a Service Capability Server can be registered at the Framework. In this way the Framework can inform the Applications upon request about available service capability features (Discovery). For example, this mechanism is applied when installing or upgrading a Service Capability Server.

The basic mechanism between Framework and Enterprise Operator is:

- **Service Subscription function**: This function represents a contractual agreement between the Enterprise Operator and the Framework. In this subscription business model, the enterprise operators act in the role of subscriber/customer of services and the client applications act in the role of users or service consumers. The Framework itself acts as service retailer.

### 3.3.1. Framework Access Session API

- **Initial Access**: Before being authorized to use the OSA SCFs, the client must first of all authenticate itself with the Framework. For this purpose, the client needs a reference to the Initial Contact interfaces for the Framework. It may be obtained through an URL, a Naming or Trading Service or an equivalent service, a stringified object reference, etc. To initiate the authentication process, the initiateAuthenticationWithVersion is used (and only supported). Once the client has been authenticated by the Framework, it can gain access to other Framework interfaces and SCFs. This is done by invoking the requestAccess method, by which the client requests a certain type of access SCF. Independently, the client could decide to authenticate the Framework, before deciding to continue using the interfaces provided by the Framework. Steps are:

  1. **Initiate Authentication**: The client invokes initiateAuthenticationWithVersion on the Framework’s «public» (initial contact) interface to initiate the authentication process.

  2. **Select Authentication Mechanism**: The client invokes selectAuthenticationMechanism on the Framework’s API Level Authentication interface, identifying the authentication algorithm it supports for use with CHAP (OSA authentication is based on CHAP).
3. The client authenticates the Framework, issuing a challenge in the `challenge()` method
4. The client provides an indication if authentication succeeded
5. The Framework authenticates the client. It can authenticate the client before the client authenticates the Framework, or afterwards, or the two authentication processes could be interleaved. However, the client shall respond immediately to any challenge issued by the Framework, as the Framework might not respond to any challenge issued by the client until the Framework has successfully authenticated the client.
6. The Framework provides an indication if authentication succeeded
7. Request Access. Upon successful authentication of the client by the Framework, the client is permitted to invoke `requestAccess` on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface.
8. The client and Framework negotiate the signing algorithm to be used for any signed exchanges.

**Figure 17: Initial access in OSA/Parlay Framework**

- **Framework Terminates Access:**
  1. Following successful authentication and service discovery, the client initiates the service agreement signing process and the application starts to use the new service manager interface.
  2. The Framework decides to terminate the application’s access session, and to terminate all its service agreements. This is an unusual and drastic step, but could be due for example to violation or expiry of the application’s service agreements, or some problem within the
Framework itself. The Framework will also destroy each of the service managers the application was using. The application is now no longer authenticated with the Framework, and all Framework and service interfaces it was using are destroyed.

![Diagram of OSA/Parlay Framework terminates access](image)

**Figure 18: OSA/Parlay Framework terminates access**

- **Application Terminates Access:**
  1. The application terminates its use of the service manager instances in a controlled manner
  2. The application decides to terminate its access session and all its service agreements in one go. The Framework will also destroy each of the service managers the application was using. The application is now no longer authenticated with the Framework, and all Framework and service interfaces it was using are destroyed. The application could have terminated its service agreements one by one which would have been a more controlled shutdown.

![Diagram of OSA/Parlay application terminates access](image)

**Figure 19: OSA/Parlay application terminates access**

- **Non-API level authentication:**
  1. The client calls `initiateAuthenticationWithVersion` on the OSA Framework Initial interface. This allows the client to specify the type of authentication process.
  2. The client invokes the `requestAccess` method on the Framework's Authentication interface
  3. If the authentication was successful, the client and the Framework can negotiate, on the Framework's Access interface, the signing algorithm to be used for any signed exchanges.
4. The client can now invoke obtainInterface on the Framework’s Access interface to obtain a reference to its service discovery interface.

![Diagram showing the OSA/Parlay authentication flow](image)

**Figure 20: OSA/Parlay Non-API level authentication**

- **API level authentication:**
  The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

1. The client calls initiateAuthenticationWithVersion on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. OSA defines a generic authentication interface (API Level Authentication), which can be used to perform the authentication process.

2. The client invokes the selectAuthenticationMechanism on the Framework’s API Level Authentication interface. This includes the authentication algorithms supported by the
client. The Framework then chooses a mechanism based on the capabilities of the client and the Framework

3. The application and Framework interact to authenticate each other by using the challenge method. Note that at any point during the access session, either side can request re-authentication of the other side.

![Sequence diagram of authentication operations]

**Figure 21: OSA/Parlay API level authentication**

**3.3.2. Framework to Application API**

This API is used for applications to discover and use SCFs. Several methods are used to:

- Event the application requests to be informed of the availability of new SCFs (it will become available after the invocation of registerService and announceServiceAvailability on the Framework)
- Suspend or resume notifications from the application based on the evaluation of the load balancing policy as a result of the detection of a change in load level of the Framework
• Request load statistics for an application
• Register itself and the application invokes load management function to inform the Framework of application load
• Report its load condition to the Framework load manager
• Enable an application requests load statistics for the Framework
• Enable an application registers itself and the Framework invokes load management function based on policy
• Monitor applications (heartbeat mechanisms)
• Inform the client application of failure
• Select the right SCF
• Discover a new Service Capability Feature in the network. Even applications that have already used the OSA API of a certain network know that the operator may upgrade it any time; this is why they use the Service Discovery interfaces. Before the discovery process can start, the Application needs a reference to the Framework’s Service Discovery interface; this is done via an invocation the method obtainInterface on the Framework’s Access interface. Discovery can be a three-step process. The first two steps have to be performed initially, but can subsequently be skipped (if the service type and its properties are already known, the application can invoke discoverService() without having to re-invoke the list/discoverServiceType methods). The following figure illustrates this method.

![Diagram](image.png)

**Figure 22: OSA/Parlay Service Discovery Sequence Diagram**

### 3.3.3. Framework to Enterprise Operator API

In some cases, the client applications (or the enterprise operators on behalf of these applications) must explicitly subscribe to the services before the client applications can access those services. To accomplish this, they use the service subscription function of the Framework for subscribing or un-subscribing to services. Subscription represents a contractual agreement between the enterprise operator and the Framework operator. In general, an entity acting in the role of a customer/sub-
scriber subscribes to the services provided by the Framework on behalf of the users/consumers of the service. In this model, the enterprise operators act in the role of subscriber/customer of services and the client applications act in the role of service users or consumers. The Framework itself acts as services retailer.

The following examples illustrate these roles:

- Service (to be subscribed): Call Centre Service, Mobility Service, etc...
- Framework Operator: France Telecom, British Telecom, etc...
- Enterprise Operator: A Financial institution such as a Bank or Insurance Company, or possibly an Application Service Provider (Such an enterprise has a conformant Subscription Application in its domain which «talks» to its peer in the Framework)
- User/Consumer: Client Applications (or their associated users) in the enterprise domain that use the Call Centre Service or the Mobility Service.

The Service Subscription interface is used by an enterprise operator to subscribe to new services and is required before a client application of the enterprise can use the new service. In general, the service subscription is performed after an off-line negotiation of a set of services and the associated price between the Framework operator and the enterprise operator. The service subscription is performed online by the enterprise operator in the frame of an existing off-line negotiated contract between the Framework operator and the enterprise. The on-line service subscription function is used for subscriber, client application, and service contract management. The following clause describes a service subscription model.

**Subscription Business Model**

The following figure shows the subscription business model with respect to the business roles involved in the service subscription process. The subscription process involves the enterprise operator (which acts in the role of service subscriber) and the Framework (which acts in the role of service provider or retailer). Services may be provided to the Enterprise Operator directly by a service provider or indirectly through a retailer, such as the Framework. An enterprise operator represents an organization or a company which will be hosting client applications.

Before a service can be used by the client applications in the enterprise operator’s domain, subscription to the service must take place. An enterprise operator subscribes to a service by (electronically) signing a contract about the service usage with the Framework, using an on-line subscription interface provided by the Framework. The Framework provides the service according to the service contract. The Enterprise Operator authorizes the client application in his/her domain for the service usage. Finally a subscribed service can be used by a particular client application.
Figure 23: OSA/Parlay Subscription Business Model

The interfaces between an enterprise operator and the client applications in its domain are outside the scope of this API. The enterprise operator provides the Framework with the data about the client applications in its domain and the type of services each of them should be allowed access to, using the subscription interfaces offered by the Framework. The Framework provides (to the enterprise operator) the subscription interfaces for subscriber, client application and service contract management. This gives the enterprise operators the capability to dynamically create, modify and delete, in the Framework domain, the client applications and service contracts belonging to its domain.

The enterprise operator is represented in the Framework domain as an EntOp object. The EntOp object is identified by a unique ID and contains the enterprise operator properties. The EntOp ID is a unique identifier of an enterprise operator in the Framework domain. It is created by the Framework Operator during the pre-subscription off-line negotiation of services (and their price, etc.) phase. The enterprise operator properties contain information such as the name and address of the enterprise operator (individual or organization), service charge payment information, etc. The enterprise operator domain has one or more client applications associated with it. The enterprise operator may group a sub-set of client applications in its domain in order to assign the same set of service features to the group. Such a group is called a Subscription Assignment Group (SAG). An enterprise operator may have multiple SAGs in its domain.

A SAG relates a client application to the features of a service. A client application may be a member of multiple SAGs, one for each service subscribed for the client application by its enterprise operator. The enterprise operator subscribes to a number of services by creating a service contract with the Framework for each service. Each service subscription is described by a service contract which defines service provision conditions. A service contract restricts service usage at subscription time. It contains one or more Service Profiles, one for each SAG in the enterprise operator domain. A
Service Profile contains the service parameters which further restrict the corresponding parameters in the service contract in order to adapt the service to the SAG’s needs. A service profile is therefore a restriction of the service contract that provides a SAG with restricted service features. It is identified by a unique ID (within the Framework domain) and contains a set of service properties, which defines the restricted usage of service allowed for that SAG (and its client applications).

Figure 24: Relationship between Client Applications/SAG, Service Contracts and Profiles

The client application is related to the enterprise operator for the usage of a service. It is represented in the Framework domain as a clientApp object. The clientApp object is identified by a unique ID and contains a set of client application properties describing the client application’s relevant information for subscription. Each client application is part of at least one SAG, which can contain one or more client applications. Each SAG has one service profile per service that defines the SAG members’ preferences for that service usage. A SAG can have multiple Service Profiles associated with it, one for each service subscribed by the enterprise operator on behalf of the SAG members. The figure above shows the relationship between client application objects, SAGs, service contracts and service profiles.

An enterprise operator may not want to grant all client applications in its domain the same service characteristics and usage permissions. In this case the enterprise operator can group them in a set of SAGs and assign a particular Service Profile to each group. A client application can be assigned
to more than one service profile for a given service, as long as the dates within those service profiles do not overlap. The figure below illustrates this. Here, the client is assigned to two SAGs. One of these SAGs uses ServiceProfile1 to control access to service 1. The other uses ServiceProfile3 to control access to service 1. If the dates in the two service profiles overlap, as is the case here, then it can not be determined when the client signs the service agreement which service profile should be used. For example, if the client application signed the service agreement on February 8th, then it could not be determined which of service profile 1 or service profile 3 would apply. If the dates are not overlapping, then there is no problem identifying which service profiles to use. A SAG may have multiple service profiles, one for each subscribed service, associated with it.

![Diagram](image)

**Overlapping date fields in service profiles**

**Figure 25: Multiple Enterprise Operators**

The figure above illustrates that the Framework can offer its services to applications in the domains of many enterprise operators. An enterprise operator could be an Application Service Pro-
provider, a corporation, or even the network operator (if the services offered through the Framework belong to a single network – it is even possible that the network operator will be the only enterprise operator). It is possible, however, that each service registered with the Framework could actually be in a different network. The client application IDs have to be unique within the Framework. The Framework operator could decide to allocate a block of application IDs to each enterprise operator, or even negotiate with the enterprise operators to provide a set of client application IDs that are meaningful to them. Service subscription and subscription management requires a careful delineation of subscription-related functions. The service subscription interfaces are classified in the following categories:

- Enterprise Operator Account Management
- Enterprise Operator Account Query
- Service Contract Management
- Service Contract Query
- Service Profile Management
- Service Profile Query
- Client Application Management
- Client Application Query.

Only the enterprise operator, which is registered and later on authenticated, is allowed to use these interfaces.

### 3.3.4. Framework to Service API

Several methods are used on this API in order to:

- Enable service register (the Framework invokes load management function based on policy)
- Invoke load management function to inform the Framework of service load
- Monitor the service
- Sign service agreement
- Register a new Service Capability Feature in the Framework.
### 3.3.5. Minimum set of Framework interfaces

The following table shows the minimum set of Framework interfaces:

<table>
<thead>
<tr>
<th>Category</th>
<th>Interface classes</th>
<th>Mandatory methods</th>
<th>System</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Access</td>
<td>IpInitial</td>
<td>initiateAuthentication()</td>
<td>OSA</td>
<td>Initial Access and Authenticate</td>
</tr>
<tr>
<td></td>
<td>IpAccess</td>
<td>obtainInterface()</td>
<td>GW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>obtainInterfaceWithCallBack()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>endAccess()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>requestAccess()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>selectEncryptionMethod()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authenticate()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>abortAuthentication()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>authenticationSucceeded()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>requestAccess()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>terminateAccess()</td>
<td>OSA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authenticate()</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>abortAuthentication()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>authenticationSucceeded()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Framework to Application API</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IpService</td>
<td>signServiceAgreement()</td>
<td>OSA</td>
<td>SLA management between AS and GW</td>
</tr>
<tr>
<td></td>
<td>Agreement Management</td>
<td>terminateServiceAgreement()</td>
<td>GW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>selectService()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>initiateSignServiceAgreement()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>signServiceAgreement()</td>
<td>OSA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>terminateServiceAgreement()</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>listServiceTypes()</td>
<td>OSA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>describeServiceType()</td>
<td>GW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>discoverService()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Framework to Enterprise Operator API</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td><strong>Framework to Service API</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IpFwService Registration</td>
<td>registerService()</td>
<td>OSA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>announcementServiceAvailability()</td>
<td>GW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unregisterService()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unannouncementService()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.4. Description of the OSA/Parlay SCFs

#### 3.4.1. Call Control

The call model used for the Call Control SCFs has the following objects:

- **A call object.** A call is a relation between a number of parties. The call object relates to the
entire call view from the application: the entire call will be released when a release method is called on the call object. Note that different applications can have different views on the same physical call, e.g. one application for the originating side and another application for the terminating side. The applications will not be aware of each other: all ‘communication’ between the applications will be by means of network signaling. The API currently does not specify any feature interaction mechanisms

- A **call leg object**. The leg object represents a logical association between a call and an address. The relationship includes at least the signaling relation with the party. The relation with the address is only made when the leg is routed. Before that, the leg object is idle and not yet associated with the address
- An **address**. The address logically represents a party in the call
- A **terminal**. A terminal is the end-point of the signaling and/or media for a party. This object type is currently not addressed.

The call object is used to establish a relation between a number of parties by creating a leg for each party within the call. Associated with the signaling relationship represented by the call leg, there may also be a bearer connection (e.g. in the traditional voice only networks) or a number (zero or more) of media channels (in multi-media networks).

A leg can be attached to the call or detached from the call. When the leg is attached, this means that media or bearer channels related to the legs are connected to the media or bearer channels of the other legs attached to the same call, i.e. only attached legs can ‘speak’ to each other. A leg can have a number of states, depending on the signaling received from or sent to the party associated with the leg. Usually there is a limit to the number of legs that are being routed (i.e. the connection is being established) or connected to the call (i.e. the connection is established). Also, there usually is a limit to the number of legs that can be simultaneously attached to the same call. Some networks distinguish between controlling and passive legs. By definition, the call will be released when the controlling leg is released. All other legs are called passive legs. There can be at most one controlling leg per call. However, there is currently no way the application can influence whether a Leg is controlling or not.

There are two ways for an application to get the control of a call. It can request to be notified of calls that meet certain criteria. When a call occurs in the network that meets these criteria, the application is notified and can control the call. Some legs will already be associated with the call in this case. Another way is to create a new call from the application.

### 3.4.2. User Interaction

The Generic User Interaction service capability feature is used by applications to interact with end users. It consists of three interfaces:

- User Interaction Manager, containing management functions for User Interaction related issues
- Generic User Interaction, containing methods to interact with an end-user
- Call User Interaction, containing methods to interact with an end-user engaged in a call.
The following table gives an overview of the Generic User Interaction methods and to which interfaces these methods belong.

### Overview of Generic User Interaction interfaces and their methods

<table>
<thead>
<tr>
<th>User Interaction Manager</th>
<th>Generic User Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateUI</td>
<td>SendInfoReq</td>
</tr>
<tr>
<td>createUICall</td>
<td>SendInfoRes</td>
</tr>
<tr>
<td>create Notification</td>
<td>SendInfoErr</td>
</tr>
<tr>
<td>destroyUINotification</td>
<td>SendInfoAndCollectReq</td>
</tr>
<tr>
<td>reportNotification</td>
<td>SendInfoAndCollectRes</td>
</tr>
<tr>
<td>userInteractionAborted</td>
<td>SendInfoAndCollectErr</td>
</tr>
<tr>
<td>userInteractionNotificationInterrupted</td>
<td>release</td>
</tr>
<tr>
<td>userinteractionNotificationContinued</td>
<td>userInteractionFaultDetected</td>
</tr>
<tr>
<td>changeNotification</td>
<td>SetOriginatingAddress</td>
</tr>
<tr>
<td>getNotification</td>
<td>getOriginatingAddress</td>
</tr>
<tr>
<td>enableNotifications</td>
<td></td>
</tr>
<tr>
<td>disableNotifications</td>
<td></td>
</tr>
</tbody>
</table>

The following table gives an overview of the Call User Interaction methods and to which interfaces these methods belong.

### Overview of Call User Interaction interfaces and their methods

<table>
<thead>
<tr>
<th>User Interaction Manager</th>
<th>Call User Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>As defined for the Generic User Interaction SCF</td>
<td>Inherits from Generic User Interaction and adds:</td>
</tr>
<tr>
<td></td>
<td>recordMessageReq</td>
</tr>
<tr>
<td></td>
<td>recordMessageRes</td>
</tr>
<tr>
<td></td>
<td>recordMessageErr</td>
</tr>
<tr>
<td></td>
<td>deleteMessageReq</td>
</tr>
<tr>
<td></td>
<td>deleteMessageRes</td>
</tr>
<tr>
<td></td>
<td>deleteMessageErr</td>
</tr>
<tr>
<td></td>
<td>abortActionReq</td>
</tr>
<tr>
<td></td>
<td>abortActionRes</td>
</tr>
<tr>
<td></td>
<td>abortActionErr</td>
</tr>
</tbody>
</table>

The IpUI Interface provides functions to send information to, or gather information from the user, i.e. this interface allows applications to send SMS and USSD messages. An application can use this interface independently of other SCFs. The IpUICall Interface provides functions to send information to, or gather information from the user (or call party) attached to a call.
The User Interaction SCF is used for:

- **Alarm Call**: a ‘reminder message’, in the form of an alarm, being delivered to a customer as a result of a trigger from an application. Typically, the application would be set to trigger at a certain time, however, the application could also trigger on events.

- **Call Barring**: a call barring service, initiated as a result of a prearranged event being received by the call control service. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is accepted and the call is routed to the original called party.

- **Network Controlled Notifications**

- **Prepaid**: The subscriber is using a pre-paid card or credit card to pay for the call. The application allows each time a certain time slice for the call. Afterwards, a new time slice can be started or the application can terminate the call.

- **Prepaid with Advice Of Charge**: The application will send the charging information before the actual call setup and when during the call the charging changes new information is sent in order to update the end-user. Note that the Advice of Charge feature requires an application in the end-user terminal to display the charges for the call, depending on the information received from the application.

### 3.4.3. Mobility

The Mobility SCF provides information about location by using simple method. Its purpose is to allow an application to perform:

- User location interrogation (triggered, periodic or interactive request)
- User location CAMEL interrogation (triggered, periodic or interactive request)
- Subscription and network induced location reports to an Emergency User Location Service
- Triggered reporting
- Interactive request to status report.

The User Location service (UL) provides a general geographic location service. UL has functionality to allow applications to obtain the geographical location and the status of fixed, mobile and IP based telephony users. UL is supplemented by User Location Camel service (ULC) to provide information about network related information. There are also specialized functionalities to handle emergency calls in the User Location Emergency service (ULE).

The UL service provides the IpUserLocation and IpTriggeredUserLocation interfaces. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppUserLocation and IpAppTriggeredUserLocation interfaces to provide the callback mechanism. When periodic or triggered location reporting is used, errors may be reported either when the recurrent reporting is requested, as an error per user in reports, or in the corresponding error method when the error concerns all subscribers in an assignment.
Furthermore, the ULC provides location information, based on network-related information, rather than the geographical co-ordinates that can be retrieved via the general User Location Service. Using the ULC functions, an application programmer can request the VLR Number, the location Area Identification and the Cell Global Identification and other mobile-telephony-specific location information. The ULC provides the IpUserLocationCamel interface. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppUserLocationCamel interface to provide the callback mechanism.

In the case of an emergency call, the network may locate the caller automatically. The resulting location is sent directly to an application that is dedicated to handle emergency user location. If the dedicated emergency call application is using the API, the location is sent to the application using a callback method in the IpAppUserLocationEmergency interface. However, the network does not always send the location immediately (probably when the location procedure is not finished when the call is set up). In this case the network will send an identifier of the caller that can be used to locate the caller using the interface IpUserLocationEmergency.

The User Status Service (US) provides a general user status service. US allows applications to obtain the status of fixed, mobile and IP-based telephony users. The US provides the IpUserStatus interface. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppUserStatus interface to provide the callback mechanism.

### 3.4.4. Terminal Capabilities

The picture below illustrates how the terminal capabilities can be retrieved and their changes monitored:

![Figure 26: Terminal capabilities in OSA/Parlay](image-url)
1. The application retrieves the terminal capability of a terminal
2. The application creates an object to implement IpAppExtendedTerminalCapabilities
3. The terminal capabilities changes are started to be monitored
4. The terminal capabilities have changed and they are reported as requested
5. The report is forwarded internally to the application
6. The terminal capabilities have changed and they are reported as requested
7. The report is forwarded internally to the application
8. An error has happened in the monitoring and it is reported
9. The error report is forwarded internally to the application
10. The terminal capabilities have changed and they are reported as requested
11. The report is forwarded internally to the application
12. The terminal capability monitoring is stopped.

The Terminal Capabilities SCF enables the application to retrieve the terminal capabilities of the specified terminal. Additionally it is possible for the application to request notifications when the capabilities of the terminal change in some way. The Terminal Capabilities service provides SCF interfaces IpTerminalCapabilities and IpExtendedTerminalCapabilities. The application side interface for the reporting is called IpAppExtendedTerminalCapabilities.

The Terminal Capabilities SCF interface IpTerminalCapabilities contains the synchronous method getTerminalCapabilities. The application has to provide the terminalIdentity as input to this method. The result indicates whether or not the terminal capabilities are available in the network and, if they are, it will return the terminal capabilities (see the data definition of TpTerminalCapabilities for more information). The network may override some capabilities that have been indicated by the terminal itself due to network policies or other restrictions or modifications in the supported capabilities. This interface or IpExtendedTerminalCapabilities shall be implemented by a Terminal Capabilities SCF as a minimum requirement. If this interface is implemented, the getTerminalCapabilities() method shall be implemented as a minimum requirement.

The method used is the getTerminalCapabilities(). This method is used by an application to get the capabilities of a user’s terminal and the direction is Application to Network. The results return is the latest available capabilities of the user’s terminal.

The ClassIpExtendedTerminalCapabilities interface can be used as an extended version of terminal capability monitoring. The application programmer can use this interface to request terminal capability reports that are triggered by their changes. Note that the underlying mechanisms for this network feature are currently not fully standardised. This interface, or IpTerminalCapabilities, shall be implemented by a Terminal Capabilities SCF as a minimum requirement. The triggeredTerminalCapabilityStartReq() and triggeredTerminalCapabilityStop() methods shall be implemented as a minimum requirement. An implementation of IpExtendedTerminalCapabilities is not required to implement the minimum mandatory methods of IpTerminalCapabilities.

The methods used are the TerminalCapabilityStartReq(), that is used to request for terminal capability reports when the capabilities change or when the application obviously does not have the
current terminal capability information when this method is invoked, and the `TerminalCapabilityStop()`, that is used to stop reporting for terminal capability changes that were started by `triggered-TerminalCapabilityStartReq()`.

The `IpAppExtendedTerminalCapabilities` interface is used to send triggered terminal capability reports. It is implemented by the client application developer.

The methods used are `TerminalCapabilityReport()`, issued when the capabilities of the terminal have changed in the way specified by the criteria parameter in the previously invoked `triggered-TerminalCapabilityStartReq()` method, and the `TerminalCapabilityReportErr()`, indicates that the requested reporting has failed. Note that errors may concern the whole assignment or just some terminals. In the former case no terminals are specified.

### 3.4.5. Data Session Control

The Data Session control network service capability feature consists of two interfaces:

- Data Session manager, containing management functions for data session related issues
- 2) Data Session, containing methods to control a session.

A session can be controlled by one Data Session Manager only. Data Session Manager can control several sessions.

The Data Session Control service capability features are described in terms of the methods in the Data Session Control interfaces. The following table gives an overview of the Data Session Control methods and to which interfaces these methods belong:

<table>
<thead>
<tr>
<th>Data Session Manager</th>
<th>Data Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>createNotification</td>
<td>connectReq</td>
</tr>
<tr>
<td>destroyNotification</td>
<td>connectRes</td>
</tr>
<tr>
<td>dataSessionNotificationInterrupted</td>
<td>connectErr</td>
</tr>
<tr>
<td>dataSessionNotificationContinued</td>
<td>release</td>
</tr>
<tr>
<td>reportNotification</td>
<td>superviseDataSessionReq</td>
</tr>
<tr>
<td>dataSessionAborted</td>
<td>superviseDataSessionRes</td>
</tr>
<tr>
<td>getNotification</td>
<td>superviseDataSessionErr</td>
</tr>
<tr>
<td>changeNotification</td>
<td>dataSessionFaultDetected</td>
</tr>
<tr>
<td>enableNotifications</td>
<td>setAdviceofCharge</td>
</tr>
<tr>
<td>disableNotifications</td>
<td>setDataSessionChargePlan</td>
</tr>
</tbody>
</table>
The session manager interface provides the management functions to the data session service capability features. The application programmer can use this interface to enable or disable data session-related event notifications.

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management. The interfaces that are implemented by the services are denoted as ‘Service Interface’. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as ‘Application Interface’.

The Data Session Control provides a means to control per data session basis the establishment of a new data session. This means especially in the GPRS context that the establishment of a PDP session is modelled not the attach/detach mode. Change of terminal location is assumed to be managed by the underlying network and is therefore not part of the model. The underlying assumption is that a terminal initiates a data session and the application can reject the request for data session establishment, can continue the establishment or can continue and change the destination as requested by the terminal. The modelling is similar to the Generic Call Control but assumes a simpler underlying state model. An IpDataSessionControlManager object and an IpDataSession object are the interfaces used by the application, whereas the IpAppDataSessionControlManager and the IpAppDataSession interfaces are implemented by the application.

### 3.4.6. Generic Messaging

The Generic Messaging Service interface (GMS) is used by applications to send, store and receive messages. GMS has voice mail and electronic mail as the messaging mechanisms. The messaging service interface can be used by both. A messaging system is assumed to have the following entities:

- **Mailboxes.** This is the application’s main entry point to the messaging system. The Framework may or may not need to authenticate an application before it accesses a mailbox.
- **Folders.** A mailbox has at least the inbox and the outbox as folders. The name of the inbox is «INBOX», and the name of the outbox is «OUTBOX». These folders may have sub-folders. The names of these sub-folders are appended to their parents’ names with ‘/’ as the delimiter. For instance, if there is a sub-folder in INBOX called ‘Personal’ and a sub-folder in that folder called ‘archive’ then the fully qualified names, which are required for all operations, of the four folders are ‘INBOX’, ‘OUTBOX’, ‘INBOX/Personal’, and ‘INBOX/Personal/archive’. The names are case sensitive. A messaging service may have other folders other than the inbox and the outbox.
- **Messages.** Messages are stored in folders. Messages usually have properties associated with them.

The GMS is represented by the IpMessagingManager, IpMailbox, IpMailboxFolder and IpMessage interfaces to services provided by the network. To handle responses and reports, the developer must implement IpAppMessagingManager to provide the callback mechanism for the Messaging service manager.
3.4.7. Connectivity Manager

Connectivity Manager includes the APIs between the enterprise operator and the provider network for the two parties to establish QoS parameters for enterprise network packets travelling through the provider network.

The Connectivity Manager service provides tools for the enterprise operator to set up a Provisioned QoS service in the provider network. The QoS measures used in the enterprise network are outside the scope of the service. The API does not require any specific QoS method to be used in the enterprise network, nor in the provider network. However, in order for Provisioned QoS service to be applied to packets arriving from the enterprise network into the provider network, the packets have to be marked using DS Codepoint marking. Once the packets are so marked, they can enjoy the QoS service provisioned in the provider network.

APIs provide the enterprise network operator on-line access to provision quality of service measures that control the enterprise’s own traffic passing through the provider network. Using APIs the operator can create virtual provisioned pipes (VPrPs) in the provider network to carry the enterprise traffic and support it with pre-specified quality of service attributes. A VPrP can be thought of as a Virtual Leased Line (VLL) provisioned to deliver pre-specified QoS. The provider may offer to the enterprise operator a set of templates that are used by the operator to specify a VPrP. For instance, the provider may offer templates for video conferencing, audio conferencing, Gold Service, Silver Service, etc. Using these templates the operator can select and provision a VPrP that specifies the quality of service attributes for this VPrP.

Elements that can be specified for a VPrP include attributes such as packet delay and packet loss. Characteristics of traffic that enters the VPrP at its access point to the provider network can be also specified with attributes such as maximum rate and burst rate.

The following is an example of a possible scenario:

- The provider prepares a template with operator-specified attributes, provider-specified attributes, and unspecified attributes, one for each QoS level
- The provider generates for the enterprise network a list of all the current sites and their access points to the provider network
- Enterprise operator logs into connectivity manager after being authenticated and authorised by the Framework service
- Operator gets the list of the sites and service access points of the enterprise virtual private network (VPN) already provided to the enterprise by the provider
- Enterprise operator retrieves the set of templates available to the enterprise (as supported by the SLA), selects one, and requests a template for constructing a new VPrP based upon the selected QoS
- Enterprise operator completes the VPrP template: i.e. selects a value for delay, loss, jitter and excess traffic treatment action, enters the SLA ID against which the template could be validated, selects endpoints, load parameters and traffic flow direction, and selects the time requirements desired. The enterprise operator can choose or modify those attributes that are operator-specified attributes in the template. Provider-specified attributes cannot be modified and are inherently part of the service
• Enterprise operator submits the completed VPrP template for validation by the CM service. Operator creates a new VPrP with pending-status that holds these selections
• The provider responds after validating the requests, which may be an approval or a denial (e.g. the requested service is not available at this access point, or at the specified time)
• If the provider approves service, the operator may send packets marked with the templates DiffServ Codepoint, which identifies together with the endpoints the VPrP that carries these packets.

3.4.8. Account Management

The account manager interface provides methods for monitoring accounts. Applications can use this interface to enable or disable charging-related event notifications and to query account balances. This interface shall be implemented by an Account Management SCF. The queryBalanceReq() method, or the retrieveTransactionHistoryReq() method, or both the createNotification() and destroyNotification methods, or both the enableNotifications and disableNotifications methods shall be implemented as a minimum requirement.

The following diagram illustrates how the account management SCF implements voucher handling: the voucher amount is retrieved and credited to the account, then the voucher is destroyed.

Figure 27: Account management in OSA/Parlay
3.4.9. Charging

There are two kinds of charging methods in the Charging SCF specifications: the immediate charge, and the reservation/payments in parts

**Immediate Charge:**

This sequence diagram illustrates how immediate charging is used. Assume a WAP gateway that charges the user €0.01 per requested URL. Since it is acceptable to loose one tick worth €0.01, no prior reservations are made. The WAP gateway sends an immediate debit for each requested URL, and should a payment have as result failure, the user is disconnected.

The operations which handle units are used exactly the same, except that the amount of application usage is indicated instead of a price.

```
1. The application creates a local object implementing the IpAppChargingSession interface. This object will receive response messages from the IpChargingSession object.
2. The application orders the creation of a session. No new object is created for the charging session handling in this example implementation.
3. The application requests to charge the user €0.01.
4. The payment is acknowledged.
5. The acknowledgement is forwarded in the application.
```
6. The application requests to charge the user €0.01
7. The payment is reported to fail
8. The failure report is forwarded in the application. (repeat steps 3 - 5 and 6 - 8 as long as you want to in any order you want to)
9. The application releases the session.

Reservation/Payment in parts:
The sequence diagram illustrates how to request a reservation and how to charge a user from the reserved amount, for instance to charge a user for a streamed video which lasts 10 minutes and costs a total of €2.00. The operations and interfaces that do not provide rating are employed throughout this sequence diagram. We assume the application has already discovered the Charging SCF. As a result, the application received an object reference pointing to an object that implements the IpChargingManager interface.
The operations which handle units are used exactly the same, except that the amount of application usage is indicated instead of a price.

![Figure 29: Payment in OSA/Parlay](image-url)
1. The application creates a local object implementing the IpAppChargingSession interface. This object will receive response messages from the IpChargingSession object.

2. The application opens a charging session, a reference to a new or existing object implementing IpChargingSession is returned together with a unique session ID.

3. In this case a new object is used.

4. The application requests the reservation of €2.00.

5. Assuming the criteria for requesting a reservation are met (the application provider has permission to charge the requested amount, the charged user has agreed to pay the requested amount), the amount is reserved in the session. At this point, the application provider knows that the network operator will accept later debit requests up to the reserved amount. So, the application may start serving the user, for instance by sending the video stream.

6. The successful reservation is reported back to the application. After half of the video has been sent to the user, the application may choose to capture half of the price already.

7. The application requests to debit €1.00 from the reservation.

8. The successful debit is reported back to the application.

9. The acknowledge is forwarded to the application.

10. The application checks if the remaining lifetime of the reservation will cover the remaining 5 minutes of video. Let us assume it does not.

11. The application asks the IpChargingSession object to extend the lifetime of the reservation.

12. Assuming that the application provider is allowed to keep reservations open for longer than 10 minutes, the extendLifeTimeReq() will be honoured and confirmed properly.

13. The confirmation is forwarded to the application.

14. When the complete video has been transmitted to the user without errors, the application charges another €1.00.

15. The IpChargingSession object acknowledges the successful debit at the IpAppChargingSession callback object.

16. The IpAppChargingSession object forwards the acknowledgement to the application.

17. Since the service is complete, the application frees all resources associated with the reservation and session.

The Charging SCF is used by applications to charge for the usage of the applications. The charged user can be the same user as that uses the application. It is also possible that another user will pay the charge.

In the interfaces of the Charging SCF a «Request Number» is used when invoking operations that operate on the user's account (directly or indirectly via reservations) in order to make retries possible after application, service, or communication errors. A retry of these operations can be done by invoking the same operation with the same Request Number. In the callback to the application, the Request Number to be used for the next request operation is returned. This is the only Request Number besides the one in the last request operation that can be used. This mechanism ensures
that an application retries an operation when it does not receive an answer. The use of the Request Number ensures that there can only be one outstanding request per Charging Session. Only after an answer is received (result or error), the next request can be made. Note however that only asynchronous operations that could lead to over or under charging of the user require a request number.

Because responses from the Charging SCF can be delayed in the network the Charging SCF shall guarantee that Request Numbers are unique in a time span where delayed responses can arrive. Suppose, for example, that the response from a retried request is received indicating the next request number to use is 1000. During the period that the response to the original request (which also carries the next request number to use equal to 1000) can arrive, this request number may not be used again.

The units (of different types) that are used in a TpVolumeSet are NOT consolidated by the charging SCF. The application must use the same units when making the reservation and when debiting the amount. For example, when after a reservation of 10 minutes a debit request for 5 seconds is made, an error will be returned.

There are cases where a single instance of the merchant application may serve more than a one service user. Examples are multi-user games or conferences. Typically, the costs for the resources consumed by the single service instance will be split among all service users. On the other hand, a merchant application may show advertisements within its application, and in turn the advertised company may subside a certain percentage of the application cost. A consumer connecting to the merchant application pays only part of the costs, while the remainder is paid by the advertised company. To support this kind of application, multiple users can be specified when a charging session is created. The charging session interface itself is the same no matter if the split charging feature is used or not. It is subject to service level agreements that are negotiated between the OSA client provider and the network operator how the charge is split between the users.
The following table lists properties relevant for Content Based Charging SCF:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description/Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESSPLAN</td>
<td>INTEGER_SET</td>
<td>Indicates the supported address plan (defined in TpAddressPlan.) E.g. {P_ADDRESS_PLAN_E164, P_ADDRESS_PLAN_IP}</td>
</tr>
<tr>
<td>P_SUPPORTED_UNITS</td>
<td>INTEGER_SET</td>
<td>Indicates the unit-types that are supported, e.g. {P_CHS_UNIT_OCTETS, P_CHS_UNIT_SECONDS}</td>
</tr>
<tr>
<td>P_SUPPORTED_CURRENCIES</td>
<td>STRING_SET</td>
<td>Indicates the currency-types that are supported according to ISO 4217:1995, e.g. {«EUR», «DEM», «NLG»}</td>
</tr>
<tr>
<td>P_UNIT_CHARGING</td>
<td>BOOLEAN_SET</td>
<td>Indicates if changing based on units (rather than amounts) is supported. Value = TRUE:unit based charging is supported. Value = FALSE:unit based charging is not supported. If unit charging is supported or not or is selected or not does not tell anything about amount charging.</td>
</tr>
<tr>
<td>P_AMOUNT_CHARGING</td>
<td>BOOLEAN_SET</td>
<td>Indicates if changing based on amount (rather than amounts) is supported. Value = TRUE:amount based charging is supported. Value = FALSE:amount based charging is not supported. If amount charging in supported or not or is selected or not does not tell anything about unit charging.</td>
</tr>
<tr>
<td>P_SPLIT_CHARGING</td>
<td>BOOLEAN_SET</td>
<td>Indicates if split charging feature is available. Value = TRUE:split charging is supported. Value = FALSE:split charging is not supported</td>
</tr>
<tr>
<td>P_DEBITING</td>
<td>BOOLEAN_SET</td>
<td>Upon service registration, this property describes if the SCS supports debiting at all and if it can be turned off by the application. Upon service instantiation, it describes which mode(s) the client has selected.</td>
</tr>
<tr>
<td>P_CREDITING</td>
<td>BOOLEAN_SET</td>
<td>Upon service registration, this property describes if the SCS supports crediting at all and if it can be turned off by the application. Upon service instantiation, it describes which mode(s) the client has selected.</td>
</tr>
</tbody>
</table>
The previous table lists properties related to the capabilities of the SCS itself. The following table lists properties that are used in the context of the Service Level Agreement, e.g. to restrict the access of applications to the capabilities of the SCS.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description/Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_DEFAULT_LIFETIME</td>
<td>INTEGER_ INTERVAL</td>
<td>Defines the defaults lifetime for a charging reservation in milliseconds.</td>
</tr>
<tr>
<td>P_LIFETIME_INCREMENT</td>
<td>INTEGER_ INTERVAL</td>
<td>Defines the duration in milliseconds by which the lifetime of a charging reservation can be extended.</td>
</tr>
<tr>
<td>P_MAX_LIFETIME</td>
<td>INTEGER_ INTERVAL</td>
<td>Defines the maximum lifetime for a charging reservation in milliseconds.</td>
</tr>
<tr>
<td>P_MIN_DEBIT_AMOUNT</td>
<td>STRING_SET</td>
<td>Defines the minimum amounts for a debit operation, depending on currency. Each set element is a string that contains the amount, formatted as a string and followed by the currency. Example: {«1.00 EUR», «0.5 GBP»} means that the minimum amount in Euro is 1.00, while in Pound Sterling it is 0.5.</td>
</tr>
<tr>
<td>P_MAX_DEBIT_AMOUNT</td>
<td>STRING_SET</td>
<td>Defines the maximum amount for a debit operation, similar to P_MIN_DEBIT_AMOUNT.</td>
</tr>
<tr>
<td>P_CREDIT_AMOUNT</td>
<td>INTEGER_ INTERVAL</td>
<td>Defines the range for amounts that can be credited. Valid for any allowed currency.</td>
</tr>
<tr>
<td>P_PARALLEL_SESSIONS</td>
<td>INTEGER_ INTERVAL</td>
<td>Defines the range for the allowed amount of parallel charging sessions.</td>
</tr>
<tr>
<td>P_SESSIONS_HOUR</td>
<td>INTEGER_ INTERVAL</td>
<td>Defines the range for the allowed number of charging sessions per hour.</td>
</tr>
</tbody>
</table>

3.4.10. Policy Management

It is expected that more and more OSA services will use policies to express operational criteria. It is also expected that network providers will host policy-enabled services that have been written by 3rd party application service providers. In order to manage policy information and control access to it, a policy management service is needed. Consistent with this, a policy management service interface manager, IpPolicyManager, has been defined. All policy management interfaces are accessible from IpPolicyManager.

A number of APIs have been defined to obtain services from a policy management service. These include APIs to create, update or view policy information. Additionally, APIs have been defined to facilitate interactions between clients (e.g. a 3rd party application) and any policy enabled service. These include APIs to view policy events, to subscribe to policy events and for the generation of events by clients. All APIs conform to an underlying policy information model. Clients that perform administrative tasks, e.g. create, update or delete policy information must obtain access to Ip-
PolicyManager using the family of obtainInterface() methods supported by the IpAccess interface. Administrative tasks may be performed through methods supported by IpPolicyManager.

Clients that need to interact with a specific policy enabled service (for non-administrative tasks) can obtain access to that service’s interface directly via the selectService() method supported by the IpAccess interface. It should be noted that specific policy enabled services may support additional interfaces and methods that are not defined below. Examples of policy enabled services include: A load balancing service that uses policies to manage application loads on the network, a charging service that determines charging criteria based on policies, a call management service that uses policies to direct end-user calls to appropriate call agents, etc.

The example below is a typical implementation of policy management within OSA/Parlay specifications and shows how a specific policy rule is managed. A rule usually consists of actions and of the conditions under which actions are executed. The sequence shows the creation of a condition and its introduction into the rule.

Figure 30: Policy management in OSA/Parlay
1. Opens the transaction bracket
2. Creates a rule object in the group by passing the name as parameter. The method returns the reference to the new rule object
3. Commits and closes the transaction bracket
4. Opens the transaction bracket
5. Once the rule object created, it can be filled. Here a condition is created on the rule object, thus becoming a part of the rule. Note: conditions defined in such a way cannot be reused in other rules (a repository approach should be used then). Parameters passed are the condition name and the condition type
6. Closes the transaction bracket
7. Now using the repository approach, i.e. reusable condition or action objects, we reuse an action, and therefore ask at the IpPolicyManager interface for a reference to a named repository. The repository name is passed and the repository reference is returned
8. If we know already the name of the action object one retrieves the action directly by passing the name as parameter. Otherwise one has to retrieve the name first by using an action iterator
9. Opens the transaction bracket
10. Now, the actions must be assigned to the rule. Passed parameter is the action list, which is a list of action reference/sequence pairs
11. After having created or retrieved all needed conditions, they must be assigned to the rule. This is done by passing the list of condition to that method. If the rule is active, this will then cause the expression defined in the condition to be evaluated (as often as necessary)
12. Closes the transaction bracket.

3.4.11. Presence and Availability Management

The goal of these interfaces is to establish a standard for maintaining, retrieving and publishing information about:

- Digital identities
- Characteristics and presence status of agents (representing capabilities for communication, content delivery, etc..)
- Capabilities and state of entities
- Presence and Availability of entities for various forms of communication and the contexts in which they are available.

Establishing such a standard in the industry will facilitate creation of many inter-operable services over multiple network technologies and, in addition, allow end users greater flexibility in managing their services and communication capabilities while addressing their privacy concerns.

Consider the following simple but desirable scenario for a communication service. An end-user wishes
to receive instant messages from her management at any time on her mobile phone, from co-workers only on her desktop computer, and in certain cases for the messages to be forwarded to e-mail or even a fax machine/printer. Senders may know her availability for various forms of communication in the way she chooses to reveal it. Alternatively, senders may never know how she will be receiving their messages. This scenario spans over multiple services and protocols and can currently only be solved by a proprietary solution that maintains the required information in an ad-hoc fashion within the application.

PAM is not a replacement for the protocols standardized for various communication and network services. PAM attempts to standardize the management and sharing of presence and availability information across multiple services and networks.

The PAM specification is motivated by the observations that:

- The notions of Identity, Presence and Availability are common to, but independent of the various communication technologies, protocols and applications that provide services using these technologies
- Presence does not necessarily imply availability. End-users or organizations require greater control over making themselves available through various communication devices
- Presence based services need to address privacy concerns on who can access presence information and under what conditions.

Availability management will span over multiple communication services and service providers.

The main goal of Presence and Availability Management is to facilitate the development of a rich set of applications and services that span over multiple communication systems (instant messaging, e-mail, fax, telephony, etc.) and to provide end users with greater flexibility and control in managing their communications. A standardized platform allows software developers to create communication management applications that are independent of the underlying technologies and protocols.

As the next step in the evolution of directory and database enabled applications and services, separate managements of identity and users/organizations availability from specific applications enables uniform and centralized administration of data and creates the potential to bring control over communication services to the user's desktops.

**Identity:**

Identity, for the purposes of the PAM specification, is a limited electronic representation of an entity (i.e. an individual or an organization) that participates in PAM-enabled applications and services. This concept corresponds to the concept of “Presentity” as described in the IETF Common Presence and Instant Messaging Model (RFC 2778).

The main characteristic of an entity that is central to PAM specifications is the name (or handle), by which entities are identified by applications and services. Entities may have multiple names, login ids, account names, etc., by which they are identified. As PAM attempts to abstract over multiple networks and services, it does not assume that a single name will necessarily identify entities across all application domains.

The generalized structure available in 3GPP for user names that may contain various formats for addressing has been adopted for these specifications.
To enable entities to be identified by any of the names associated with them, PAM identities can be assigned aliases. A name and a namespace pair can be defined as an alias of another name and namespace pair. It is important to note that aliases are just synonyms and hence have limited semantics. In particular, they are not powerful enough to model personas each with their own capabilities and privacy requirements. An identity can represent a single entity or a group of identities. Group identities have similar semantics to non-group identities but, in addition, maintain a list of identities that constitute the group. As an example, a sales department may be modeled as a group identity with the identities of the members of the department being member identities of the group. Group identities and their member identities do not inherit anything from each other. No other relationships between identities are within the scope of the PAM specifications.

For flexibility and extensibility, attribute lists are used to associate additional data with identities. Identities are typed to provide a way to manage such attribute lists. An identity type may be associated with a specific set of attributes and all identities of that type inherit instances of such attributes. For consistency with IETF (RFC 2778) defined presence data models, PAM pre-defines an identity type “Presentity” with a list of presence attributes based on RFC 2778 definitions. PAM implementations may map certain existing directory and database data to one or more types to allow access via PAM interfaces. PAM specifications do not specify how the data within the profiles are to be stored. They may be stored within the PAM implementation or mapped to data stored on external directories and databases.

**Agent:**

An agent, for PAM purposes, is a limited electronic representation of a software or hardware device through which identities manifest themselves, or make themselves available to applications and services.

An important characteristic of an agent is a list of one or more capabilities associated with it. A capability is what makes an agent useful. A capability either represents the ability of an agent to participate in communications and content delivery (such as instant messaging, SMS, WAP, voice) or it represents the ability of an agent to report useful information (location, velocity, temperature, mood) of the environment around.

PAM does not specify any pre-defined capabilities. Applications may define and use their own capabilities. Agent instances are identified by names (or handles). As for identities, names exist in the context of a namespace. Within a namespace, a name is assumed to be unique. Two agent instances can have the same name as long as they are in different namespaces. For example, a mobile phone and a PDA designed by two different manufacturers may coincidentally have the same serial number by which they are identified. As PAM attempts to unify services over multiple technologies, it does not assume a name uniquely identifies agent instances across all technologies or across all manufacturers. They can be disambiguated through the use of namespaces. No relationships between agents are within the scope of the PAM specifications.

For flexibility and extensibility, attribute lists are used to associate additional data with agents. Agents are typed to provide a way to manage such attribute lists. An agent type may be associated with a specific set of attributes and all agents of that type inherit instances of such attributes. PAM does not specify any pre-defined attributes or types. Applications may define and use their own
agent types. PAM implementations may map certain existing directory and database data to one or more types to allow access via PAM interfaces. PAM specifications do not specify how the data within the profiles are to be stored. They may be stored within the PAM implementation or mapped to data stored on external directories and databases. Agent instances are associated with one or more identities. This association results in the inheritance of associated agents’ capabilities by the identities.

Presence:
The concept of presence has been used in several application areas, being most explicit in Instant Messaging. Starting from a simple notion of online/offline status, it has expanded to include other context information around the status such as disposition (out to lunch, away from the computer, etc.) and activity status (on the phone, idle, etc.). Location information, on the other hand, has largely been kept separate from what has been traditionally considered presence information. PAM specifications broaden the concepts of presence recognizing that all such information, including location, describes different contexts of an entity’s existence. The unifying property is that the presence information is continually changing and that there is value in knowing the current information at different points in time for services and applications.

For the purposes of PAM specifications, presence is an extensible set of characteristics that captures the dynamic context in which an identity or an agent exists at any point in time. In contrast to the relatively static information about identities or agents (e.g. names, addresses, capabilities), presence refers to dynamic information such as location, status, disposition, etc. Registrations of presence and location information in existing applications are covered by this definition.

Presence information is differentiated from the more static information associated with identities and agents that are stored in attributes. The rationalization for this design is that the presence information is dynamic and has implications on the implementation. Some of the presence information is too dynamic to be maintained in static data stores such as directories and without this hint about the data characteristics, PAM implementers may make sub-optimal decisions on the way the data is stored. Second, presence information typically has expiration data that needs to be understood by the implementation.

The PAM specification recognizes that devices that provide presence information are not necessarily devices that communicate. Certain agents may report presence information but not be capable of communication. Certain agents may be communication devices but may not be able to provide presence information. In general, the presence of an identity is computed from presence information provided by one or more agents and the ability to communicate is derived from one or more communication-capable agents available to the identity.

The PAM specification does not specify the methods by which the presence information is derived. An agent may explicitly register its own presence information or the information may be derived from other network elements. For example, an instant messaging client on a desktop computer can register its status based on when a user is logged in. A mobile phone may do an explicit registration on a WAP server for instant messaging. The phone’s presence for voice calls, on the other hand, may be inferred implicitly by querying the cellular network for the device being on when requested.
The presence of an identity, on the other hand, may be computed using presence information from one or more devices owned by the identity.

Finally, the PAM specification does not require that the presence information be stored explicitly (i.e. in a materialized fashion) in a PAM implementation. An implementation may infer the presence information on demand from the underlying services or networks.

For compatibility with the presence model from IETF (RFC 2778), a type called “Presentity” is pre-defined with the attributes consistent with the IETF Presence model.

**Availability:**

Availability is a property of an identity denoting its ability and willingness to share information about itself or to communicate with another identity based on factors such as the type of requested communication, the calling entity’s identity and preferences or policies associated with the recipient. This is the primary means by which the current PAM specification enables controls for privacy. While presence is, in most applications, a necessity for availability, it does not necessarily imply availability to all. Availability is always with respect to a context. A context in PAM specifications is a set of attributes defining the state in which the availability is requested. For example, the query «Is Jane available for IM for Rob?» identifies the type of communication and the identity of the asker as the context. PAM allows availability to be differentiated based on any context attribute. A «Communication» context is pre-defined in PAM.

Most queries for presence in existing applications can be mapped into PAM availability queries to control the information being given out. Alternatively, queries can be mapped directly into PAM presence queries in situations where privacy controls and policies are not required or all presence data is open to the entity querying. This allows PAM specifications to be consistent with existing presence servers and to serve as the basis for presence services across multiple protocols while providing uniform and flexible privacy controls.

PAM does not specify whether availability is computed on demand or stored explicitly. In some applications, availability may be pre-computed and stored explicitly while in some, it may be computed at each availability request. While the PAM specification provides a mechanism to associate preferences with an Identity to control availability, it neither specifies the syntax and semantics of the preferences, nor the process by which availability is computed. These aspects are left to the implementation. For example, a particular implementation may provide the facility to store preferences as rules such as «I prefer to receive my instant messages on my computer rather than my cell phone unless the message is from my boss or the computer is off, etc.».

As an example, “availability for communication” computation may consist of the following algorithm:

1. Find all devices of the identity being called that are capable of the specified form of communication AND have registered their presence status as available
2. Evaluate the rules associated with the identity being called to select the preferred device(s) from the set of present devices determined in Step 1
3. If there are any devices available satisfying Step 2, indicate the availability of the identity being called via the available devices.
An implementation can choose to provide one or more means to specify preferences. It is expected that if there is industry standardization on the preferences specification, implementations will support such a standard. This is currently outside the scope of PAM.

**Events:**

Events are representations of certain identified occurrences related to the concepts described above. The PAM specification provides for registering interest (i.e. callbacks) in being notified of such occurrences. Any entity that subscribes to the Event is a «watcher» in the IETF terminology (RFC 2778). An implementation is expected to provide such notifications.

Examples of events include:

- Creation/deletion of an identity
- Association of an agent instance with an identity
- Change in presence status or location of an agent instance
- Change in the presence information of an identity
- Change in availability of an identity for a particular form of communication.

PAM specifications contain a set of pre-defined events. Each event is defined by a name of the event, a set of input attribute value pairs that must be provided when an event is registered for and a set of attribute value pairs that are included in the notifications sent out when the event of interest occurs.

**Scope of PAM Information:**

Presence and Availability Management has the following types of information in its scope:

- Identities, which consist of names and aliases of entities participating in communications
- Agent information, which consists of names and communication capabilities of software and/or hardware devices
- Agent provisioning, which consists of associations between instances of agents and identities.
- Presence information, which consists of an identity’s or an agent’s dynamic characteristics such as status and geographical location
- Availability information, which consists of preferences associated with identities and computation of availability, based on the devices present and the current preferences
- Notification of changes to the above pieces of information
- Security issues for access to this information.

The PAM specification consists of interfaces to manage or access the above information. The specification purposefully does not include storage design or storage requirements for any of the presence and availability information. These are to be decided by specific implementations of the PAM specification.
Security and Privacy:

As the Presence and Availability Management interface is designed to share information across administrative domains and to facilitate availability computation based on the identity of the entity desiring communication, security and privacy issues are addressed in the design. Two of the issues considered to be within the scope of PAM are:

- Access control to an implementation of the PAM specification
- Use of an authenticated entity's credentials by methods in the specification.

To understand the distinction between the first two issues, consider, for example, an end-user that logs on to an Instant Messaging client and wishes to send a message. The client (or a gateway to which the client talks to) may access a PAM implementation to determine the availability of the destination for the message. The client (or the gateway) will need to be authorized for access to the PAM implementation independent of the user that logs in. A gateway may, in fact, do this access on behalf of a number of clients and, for performance reasons, wish to authenticate itself just once on start up rather than at each invocation. This authentication is handled by the authentication mechanisms in the OSA Framework common to all services within OSA. Second, each invocation of a particular method will need to contain the credentials of the end-user that logged into the client so that the computation of the availability can take that into account when necessary for privacy issues.

It should be noted that the PAM specification allows the possibility that the authentication of the end-users is not necessarily done within the PAM implementation itself. As long as the authenticated credentials supplied by the client (or gateway) are acceptable for validation and the client (or the gateway) itself is authenticated by the implementation, the authentication of end-users can occur anywhere outside the PAM implementation. A deployment scenario for a particular application is that one or more authentication services are provided as external services over PAM implementations. This design does not preclude the possibility that the client (or the gateway) cannot be authenticated. Therefore, the credentials supplied by the client (or the gateway) may be held to stronger authentication criterion than credentials supplied by a trusted client (or gateway).

Finally, the PAM specification does not mandate the use of authentication within an implementation if the environment in which it is used does not require it.

Privacy issues are addressed primarily by providing a mechanism to control the information flowing out of a PAM implementation based on whatever criterion the end user may choose to specify in the availability preferences and independent of any particular application.

The following security issues were considered to be outside the scope of PAM:

- Authentication of the identity of the end-users or entities. As explained above, this authentication may be provided by a third-party authentication service or it may occur through an authentication service written over the PAM platform. The only requirement is that the type of credentials supplied by the authentication service be acceptable to the PAM platform implementation being accessed
- Encryption of the flow of information between a PAM platform implementation and clients of this implementation. This is dependent on the method of access to the interface which is outside the scope of the PAM specification and hence to be determined by the implementation.
3.5. OSA and Web services

At the time Parlay standards were released, operators have been hit by the general decline of investments in the 2001-2003 years. Parlay had been designed to open up traditional service platforms to services in the IT world. This was indeed possible, using Corba interfaces on the B2B model. However, this approach was left half-way through, because interfacing by Corba also meant implementing a complex low-level API, and overcoming many firewall issues to let messages go through. In other words, the initial Parlay model was opening up the wall garden of telecom services, but the fence was too high to climb for most of the IT community. It did not well fit into the business service model.

Parlay X was designed to overcome these flaws, by completing integration into a true Web service Framework and be readily usable through very simple high-level APIs, eliminating the need for IT developers to deal with the underlying glue of CORBA service management.

The general architecture of a solution including Web Services allows a number of deployment configurations. These are derivatives of a basic architecture model, enabling a variety of options. A typical Parlay X Web Services deployment model includes the publication of Parlay X Web Services through a registry, making those Web Services available for discovery, and for applications to use Web Services access methods to interact with the Gateway, where the Web Service interfaces are implemented. This architecture may be combined with existing OSA deployment configurations, providing the overall architecture illustrated below.
Before going further in the detailed study of the Parlay X APIs that allow smooth integration with the Web service concept, it is interesting to have an idea of what a Web service implementation can look like.

The Call Handling Web Service provides a mechanism for an application to specify how calls must be handled for a specific number. It includes commonly utilized actions:
The set of rules are provided to the Web Service which is responsible for establishing the call handling function. Only one action is taken for a call, and once this action is started the rules will stop being processed (i.e. the service will not control the call anymore). A specific order in which these rules are processed provides a predictable call handling expectation for rules provided. The processing is done as follows:

1. Call accepting determines if the call is accepted or rejected. If the caller is not on the accept list, the call is rejected and rule processing ends
2. Call blocking determines if the call is rejected. If the caller is on the block list, the call is rejected and rule processing ends
3. Conditional call forwarding - each calling number that has a specific forwarding instruction is checked, and the call is forwarded on a match, and rule processing ends
4. Unconditional call forwarding - the called number is changed to the call forwarding number and rule processing ends
5. Play audio - the call is handled by a voice system, which handles all further processing of the call. Rule processing ends when the call is handed off
6. Continue processing call, to complete call to the original called number.

The rules can be provisioned by the application using provisioning methods published by the Web service, like in the following message flow. Method names are explicit enough to suggest the function they fulfil.
The conceptual simplicity of a Web service hides its internal complexity. Once the building blocks such as the “third party call” Web service are created, an application like “click to dial” can use it in a very natural way by calling a few methods as shown in the example below.
The internal complexity lies in the implementation of the call control manager, while its interface remains relatively simple, as illustrated in the table below.

It is worth noting that this interface, which could be considered as an implementation issue, is actually published in the standards, with a level of detail comparable to a classic ITU-T or 3GPP protocol description.

The same is true for the API between the Framework and application, service, or enterprise applications, the Framework IDL, the service WSDL, etc... This allows service providers developing frameworks that could integrate other vendor’s applications without compatibility issues nor bothering with the internal details of how the call is created, or how the application communicates with the Framework.
3.7. Description of the Parlay X SCFs

3.7.1. Third Party Call

The Parlay X 3rd Party Call Web Service provides functions to application developers to create calls in a simple way without detailed telecom knowledge. As depicted below, the application only needs to be able to invoke the service.

Figure 35: Parlay X 3rd Party Call Control
The specified operations are:

- **MakeCall**: setup a voice call between 2 addresses, **CallingParty** and **CalledParty**, provided that the involving application is allowed to connect them. Optionally, the application can also indicate the charging information. In order to receive information on call status, the application has to invoke **GetCallInformation**

- **GetCallInformation**: it retrieves the current status, **CallInformation**, of the call identified by the **CallIdentifier**. This information is available a certain period of time after the call is ended

- **EndCall**: it terminates the call identified by the **CallIdentifier**. It has the same effect as **CancelCallRequest** if the call is in initiate state

- **CancelCallRequest**: it cancels the previously requested call identified by **CallIdentifier**. It only attempts to prevent the call from starting.

### 3.7.2. Call Notification

The Parlay X Call Notification provides simple functions to IT developers to determine how a call shall be treated. The examples of usage of this function include the following:

- **Incoming call handling**: A subscriber receives a call while he is logged-on to the Internet and occupies his telephone line. He is then regarded as busy by the network. The subscriber has an application invoked when somebody tries to call him while he is busy. The application provides the subscriber with a list of choices on how to handle the call (route the call to voicemail, redirect the call to a secretary, reject the call). Based on the subscriber's response, the call is handled in the network. Alternatively, the call is re-routed or released depending on the subscriber's preferences and on some context information (e.g. based on the subscriber's status or location)

- **Service numbers**: An application is triggered whenever a certain service number is dialed. This number is used to connect the caller to one of the maintenance personnel. The application redirects the call to the appropriate maintenance person based on, e.g. calling party number, time, location and availability of the maintenance personnel

- **SMS notification of missed calls**: An application offers the subscriber the possibility of being notified via SMS whenever he misses a call. The application registers to be notified when calls to its subscribers encounter busy, no-answer or not-reachable. The application does not influence the call treatment, but sends an SMS containing the calling party number, the time and reason why the call was missed.

This Web Service's interfaces are:

- **CallDirection**

The message-based invocations are:

- **HandleBusy**: The invocation of **HandleBusy** requests the application to inform the gateway how to handle the call between two addresses, the **CallingParty** and
the calledParty, where the calledParty is busy when the call is received. The application returns the action, which directs the gateway to perform one of the following actions: «Continue», resulting in normal handling of the busy event in the network, e.g. playing of a busy tone to the callingParty, «EndCall», resulting in the call being terminated; the exact tone or announcement that will be played to the callingParty is operator-specific and «Route», resulting in the call being re-routed to a calledParty specified by the application. Optionally, in the action parameter, the application can also indicate the charging information.

- **HandleNotReachable**: The invocation of handleNotReachable requests the application to inform the gateway how to handle the call between two addresses, the callingParty and the calledParty, where the calledParty is not reachable when the call is received. The application returns the action, which directs the gateway to perform one of the following actions: «Continue», resulting in normal handling of the ‘not reachable’ event in the network, e.g. playing of a busy tone to the callingParty, «EndCall», resulting in the call being terminated; the exact tone or announcement that will be played to the callingParty is operator-specific and «Route», resulting in the call being re-routed to a calledParty specified by the application. Optionally, in the action parameter, the application can also indicate the charging information.

- **HandleNoAnswer**: The invocation of handleNoAnswer requests the application to inform the gateway how to handle the call between two addresses, the callingParty and the calledParty, where the calledParty does not answer the received call. The application returns the action, which directs the gateway to perform one of the following actions: «Continue», resulting in normal handling of the ‘no answer’ event in the network, e.g. playing of a busy tone to the callingParty, «EndCall», resulting in the call being terminated; the exact tone or announcement that will be played to the callingParty is operator-specific and «Route», resulting in the call being re-routed to a calledParty specified by the application. Optionally, in the action parameter, the application can also indicate the charging information.

- **HandleCalledNumber**: The invocation of handleCalledNumber requests the application to inform the gateway how to handle the call between two addresses, the callingParty and the calledParty. The method is invoked when the callingParty tries to call the calledParty, but before the network routes the call to the calledParty. For example, the calledParty does not have to refer to a real end user, i.e. it could be a service number. The application returns the action, which directs the gateway to perform one of the following actions: «Continue», resulting in normal handling in the network, i.e. the call will be routed to the calledParty number, as originally dialed, «EndCall», resulting in the call being terminated; the exact tone or announcement that will be played to the callingParty is operator-specific and «Route», resulting in the call being re-routed to a calledParty specified by the application. Optionally, in the action parameter, the application can also indicate the charging information.
• **CallNotification**

The message-based invocations are:

- **NotifyBusy**: A busy notification informs the application that a call between two parties was attempted, but the called party was busy
- **NotifyNotReachable**: A not reachable notification informs the application that a call between two parties was attempted, but the called party was not reachable
- **NotifyNoAnswer**: A no answer notification informs the application that a call between two parties was attempted, but the called party did not answer
- **NotifyCalledNumber**: A called number notification informs the application that a call between two parties is being attempted.

### 3.7.3. Short Messaging

This clause describes a Parlay X Web Service, for sending and receiving SMS messages. The overall Web Service scope is to provide to application developers primitives to handle SMS in a simple way. In fact, using the SMS Web Service, they can invoke SMS functions without specific telecom knowledge. For sending a message to the network, the application invokes a message to send it, and must subsequently become active again to poll for delivery status. There is an alternative to this polling mechanism: an asynchronous notification mechanism implemented with an application-side Web Service. However, it was decided not to provide a notification mechanism in the first release, to make the API as simple as possible, even though the polling mechanism is not as network efficient as the notification mechanism. For receiving a message from the network, the application may use either polling or notification mechanisms. The notification mechanism is more common: network-initiated messages are sent to autonomous application-side Web Services. Both mechanisms are supported, but notification-related criteria provisioning is not specified.

The following figure shows a scenario using the SMS Web Service to send an SMS message from an application. The application invokes a Web Service to retrieve a weather forecast for a subscriber (1) and (2) and a Parlay X Interface (3) to use the SMS Web Service operations (i.e. to send an SMS). After invocation, the SMS Web Service invokes a Parlay API method (4) using the Parlay/OSA SCS-SMS (User Interaction) interface. This SCS handles the invocation and sends an UCP operation (5) to an SMS-C. Subsequently, the weather forecast is delivered (6) to the subscriber. In an alternative scenario, the Parlay API interaction involving steps (4) and (5) could be replaced with direct interaction between the SMS Web Service and the mobile network.
The next figure shows a scenario using the SMS Web Service to deliver a received SMS message to an application. The application receives a Parlay X Web Service invocation to retrieve an SMS sent by a subscriber (1) and (2). The SMS message contains the e-mail address of the person the user wishes to call. The application invokes a Parlay X Interface (3) to the Third Party Call Web Service in order to initiate the call (4).

Figure 37: Parlay X receive SMS scenario
3.7.4. Multimedia Messaging

This contribution defines a Multimedia Messaging Web Service that can map to SMS, EMS, MMS, IM, E-mail, etc. The choice is between defining one set of interfaces per messaging network, or a single set common to all networks; e.g. we could define sendMMS, sendEMS, sendSMS, etc., or just use sendMessage. Although the more specific the API the easier it is to use, there are advantages to a single set of network-neutral APIs. These advantages include:

- Improved service portability
- Lower complexity, by providing support for generic user terminal capabilities only.

The Parlay X specification provides sets of interfaces for two messaging Web Services: Short Messaging (part 4) and Multimedia Messaging (this part), which provides generic messaging features (including SMS). For sending a message to the network (see Send Message), the application invokes a message to send it and must subsequently become active again to poll for delivery status. There is an alternative to this polling mechanism: an asynchronous notification mechanism implemented with an application-side Web Service. However it was decided not to provide a notification mechanism in the first release, to make the interface as simple as possible, even though the polling mechanism is not as network efficient as the notification mechanism.

For receiving a message from the network, the application may use either polling (see Receive Message) or notification (see Message Notification) mechanisms. The notification mechanism is more common: network-initiated messages are sent to autonomous application-side Web Services. Both mechanisms are supported, but the provisioning of the notification-related criteria is not specified. The following figure shows an example scenario using sendMessage and getMessageDeliveryStatus to send data to subscribers and to determine if the data has been received by the subscriber. The application invokes a Web Service to retrieve a stock quote (1) and (2) and sends the current quote - sendMessage - using the Multimedia Messaging Web Service’s Parlay X Interface (3). After invocation, the Multimedia Message Web Service sends the message to an MMS-C using the MM7 interface (4) for onward transmission (5) to the subscriber over the mobile network. Later, when the next quote is ready, the application checks (getMessageDeliveryStatus) if the previous quote has been successfully delivered to the subscriber. If not, it may for instance perform an action (not shown) to provide credit for the previous message transmission. This way, the subscriber is only charged for a stock quote if it is delivered on time.
3.7.5. Payment

A vast amount of content, both information and entertainment, will be made available to subscribers. To support a business model that enables operators to offer integrated billing, a payment API is crucial. Open and inter-operable «payment APIs» are key to market growth and investment protection. The Payment Web Service supports payments for any content in an open, Web-like environment.

The Payment Web Service described hereafter supports payment reservation, pre-paid payments, and post-paid payments. It supports charging of both volume and currency amounts, a conversion function and a settlement function in case of a financially resolved dispute.

Note that certain parameters are negotiated off line, for example, the currency, volume type, default reservation enforcement time, as well as the taxation procedures and parameters.

An application scenario example could be a multimedia service. Assume a subscriber is interested in receiving a stream of, say, a football game. The subscriber selects a game and establishes a trusted relation with the provider. Again, the provider obtains the MSISDN and other information from the subscriber. The subscriber wants to know the service cost and the provider interacts with the operators rating engine (getAmount) taking into account the subscriber’s subscription, time of day, etc. The value returned is a currency amount and is printed on the page that is displayed at the mobile station (MS).

The subscriber then decides to stream the match to his MS. Subsequently, the provider will reserve the appropriate amount with the operator (reserveAmount), to ensure the subscriber can fulfill his payment obligations. The game starts and the provider periodically charges against the reservation (chargeReservation). The game ends in a draw and is extended with a ‘sudden death’ phase¹. The subscriber continues listening, so the existing reservation is enlarged (reserveAdditionalAmount).

¹ The next team that scores wins the game
Suddenly, a team scores a goal, making the game end abruptly and leaving part of the reserved amount unused. The provider now releases the reservation (releaseReservation), and the remaining amount is available for future use by the subscriber. We can now extend this scenario by having the subscriber participate in a draw in which the provider refunds a percentage of the usage costs (refundAmount) based on the ranking of a particular team in the tournament. For example, the subscriber gambling on the team that wins the tournament receives a full refund, while gambling on the team that finishes second, brings a 50% refund.

3.7.6. Account Management

Pre-paid subscribers may have to recharge their accounts. This occurs through an application that interfaces with subscriber either directly or indirectly. Examples of direct interaction are voice prompts and WAP/Web pages, or even SMS. Typically, such multi-modal applications either request a currency amount and, for example, credit card information, or a voucher number plus credentials. The voucher number and credentials are then validated and cause a pre-determined currency amount to be transferred.

The Parlay X Account Management API supports account querying, direct recharging and recharging through vouchers. As a side effect, it may prevent subscribers from having their account balance credits expire.

3.7.7. Terminal Status

Terminal Status provides access to the status of a terminal through:
- Request for a terminal’s status
- Request for a group of terminals’ status
- Notification of a terminal status change.

Terminal status can be expressed as reachable, unreachable or busy - however not all terminals distinguish a busy status, so applications should be able to adapt to what information is available (using service properties to determine available information).

When a request for a group of terminals is made, the response may contain a full or partial set of results. This allows the service to provide results based on a number of criteria including number of terminals for which the request is made, and amount of time required to retrieve the information. This allows the requester to initiate additional requests for those terminals for which information was not provided.

In some applications, terminal status notification will be used to watch for a specific status change. An example is a ‘call when available’ service, where terminal status is checked and determined to be not reachable or busy, and a notification is set up to notify the application when the terminal becomes reachable. Between the time of the original status determination and the time the notification is set up, terminal status could change to reachable, thus the notification on change to reachable would not be sent. Using the “check immediate” flag, after the notification is established, the value of the terminal status will be determined, and if the criteria is matched, then a notification will be sent immediately.
3.7.8. Terminal Location

Terminal Location provides access to the location of a terminal through:
- Request for a terminal’s location
- Request for a group of terminals’ location
- Notification of a terminal location change
- Notification of terminal location on a periodic basis
- Location is expressed through a latitude, longitude, altitude and accuracy.

Like Terminal Status specifications, the response can contain a full or partial set of results. This allows the service to provide results based on a number of criteria.

3.7.9. Call Handling

The Call Handling Web Service provides a mechanism for an application to specify how calls are to be handled for a specific number. Call handling includes commonly utilized actions:
- Call accepting - only accepting calls from a list of numbers
- Call blocking - blocking calls if they are on a blocking list
- Conditional call forwarding - changing the destination of a call to another number for a specific calling number
- Unconditional call forwarding - changing the destination of a call to another number
- Play audio - initiate audio with the caller (e.g. an announcement or menu).

The set of rules are provided to the Web Service responsible for establishing the call handling function. Only one action is taken for a call, and once this action is started, the rules stop being processed. There is a specific order in which these rules are processed, providing a predictable call handling for the provided rules. Processing is done as follows:

1. Call accepting determines if the call is accepted or rejected. If the caller is not on the accept list, the call is rejected and rule processing ends
2. Call blocking determines if the call is rejected. If the caller is on the block list, the call is rejected and rule processing ends
3. Conditional call forwarding - each calling number that has a specific forwarding instruction is checked, and the call is forwarded on a match, and rule processing ends
4. Unconditional call forwarding - the called number is changed to the call forwarding number and rule processing ends
5. Play audio - the call is handled by a voice system, which handles all further processing of the call. Rule processing ends when the call is handed off
6. Continue processing call, to complete call to the original called number.
If no rules are specified in a particular area, then that step is skipped. If rule processing ends without any action being indicated, then the call will continue to the called number.

Call Handling provides its function without further interaction with the Application. This is in contrast to the Call Notification interfaces, which provide notifications to the Application for processing.

3.7.10. Audio Call

The Audio Call service provides a flexible way to provide vocal message delivery. The interface is very simple, not requiring the developer to manage the creation of the call nor the interactions with the call to deliver the voice message.

There are three mechanisms which may be utilized for the vocal message content:

- Text, to be rendered using a Text-To-Speech (TTS) engine
- Audio content (such as .WAV content), to be rendered by an audio player
- VoiceXML, to be rendered using a VoiceXML browser.

The service may provide one, two or all three mechanisms, with the service policies providing the mechanism for determining which are available.

3.7.11. Multimedia Conference

The Multimedia Conferencing is a simple Web Service that allows the creation of a multimedia conference and the dynamic management of the participants and the media involved.

The underlying model of the service is based on the following entities:

- Conference: a (uniquely identified) context participants can be added to or removed from
- Participant: each of the parties involved in the conference. Media can be added/removed for each participant. There may exist a participant that is also the conference «owner», i.e. who can end the call and/or be the reference user for billing purposes
- Media: the conference can utilize multiple media streams to support the participants’ communication. In particular, both audio and video streams are available, including the specific stream direction (i.e. in, out, bidirectional).

An application setting up a multimedia conference must initially invoke the createConference method. The result of this invocation is the creation of a «context» that represents a «virtual» room where users can «meet». A unique identifier is assigned to the just-created conference.

At this stage no participant is connected yet. Subsequently, the application may wish to add participants to the conference. In order to do so, the method inviteParticipant can be used. The method result is to alert the user of the incoming connection request (user’s terminal rings). If the application wishes to check whether the user has accepted the invitation (i.e. is connected) it can invoke (at a later time) the getParticipantInfo method.
Note that:

- As soon as the first participant connects, the conference becomes «active». The duration of the conference is then measured starting from the moment the conference has become active.
- The initial media set utilized by the participant will depend on the conference type and the media actually supported by the participant’s terminal.

During the conference session the application is able to:

- Add (or remove) a specific media stream to a single participant: e.g. adding a video bidirectional stream to a participant that has an audio connection to the conference. This can be obtained by invoking addMediaForParticipant and DeleteMediaForParticipant methods.
- Disconnect a participant from the conference, by invoking disconnectParticipant method.
- Retrieve information related to the conference and its status, by invoking getConferenceInfo and getParticipants methods.

Different conditions can determine the end of the conference:

1. The application may invoke the method endConference, that «forces» the termination of the conference and the disconnection of all participants.
2. The conference owner (if defined) leaves the conference. If the owner is not defined, this condition will apply when all the participants have left the conference (disconnected).
3. The conference duration exceeds a maximum value (specified during the conference creation step).

3.7.12. Address List Management

Address List Management relates to two interfaces:

- One to manage the groups themselves - creation, deletion, query and access right management.
- Another one to manage the members within a group, supporting add, delete and query operations.

Note that addresses are not created using this service, but they must already exist.

Groupe URI format:

The group URI consists of the following discrete elements:

- Scheme: selected by the provider of the group URI.
- Group name: following the conventions of RFC 2396 [3].
- Suffix: may be added by Service Provider (if allowed by creation operation) to create a unique name when the Prefix + Group name already exists.
- Sub-domain: defined by the requester, this is contained within the domain provided by the service provider.
• Domain: defined by the Service Provider, and cannot be specified by the application.

This definition of a group URI enables flexibility on the part of the Service Provider and the Requester, while ensuring unique groups are created and providing transparency of implementation of group storage.

Address lists:
A service that must support groups of address lists may satisfy this requirement through network managed groups. The group URI is passed to the service, and this group URI is resolved to the set of URIs contained within the group. If one or more group URIs are provided in a set of URIs to a service, the service will replace each group URI with its set of contained URIs, and the service processing will apply to the unique union of URIs generated.

If supported by the service policy, zero or more of the set of URIs contained within a group may be themselves group URIs, which would also be resolved. Thus, in this case, the list of URIs the service would process is the union of individual URIs (as a set with no duplicates).

Unless specifically defined in the service semantics, the expected semantic for the results of a service operation will be presented as the results for the set of URIs as processed (the union of non-group and group provided URIs), without group URIs included in the result. This eliminates a variety of complexity issues including duplicate URIs in multiple groups and the differences between a group URI and a URI referring to an endpoint.

3.7.13. Presence

The presence service allows presence information to be obtained about one or more users, and to register presence for the same. It is assumed that the typical client of these interfaces is either a supplier or a consumer of the presence information. An Instant Messaging application is a canonical example of such a client of this interface.

The following figure shows the architecture of the Presence Web Service and the underlying services. The Parlay/OSA PAM SCF is the straightforward option, and implements the presence server with extended identity, device capability, and presence agent management. Parlay/OSA PAM allows aggregation of presence information from internet, mobile and enterprise users, etc. using a presence transport network of SIP or XMPP servers. The Presence Web Service can however communicate directly for example with IMS presence network elements (presence and resource list servers) using the ISC (SIP/SIMPLE) protocol interface.
Presence is related to other strongly linked specifications:

- **Parlay X Terminal Status Web Service and Parlay X Terminal Location Web Service**: Both services deal with information that could be considered part of the user’s presence information. Communication abilities can be derived from terminal status information, and the user’s place type can be derived from his location.

- **Parlay/OSA PAM**: The Parlay/OSA Presence and Availability specification can be considered the big brother of the presence specifications. While Parlay X Presence stays behind Parlay/OSA PAM in terms of flexibility and power - especially concerning attributes and management interfaces - it also extends PAM by introducing end-to-end authorization. The present document aims to be mappable to Parlay/OSA PAM.

- **SIP SIMPLE**: The presence specification aims to be mappable to the SIP/SIMPLE architecture.

- **XMPP (Jabber)**: Many principles of XMPP have been adopted, especially the end-to-end authorization.

- **IETF Rich Presence**

- **Group Management**: Presence of groups is supported by the presence specifications, however their creation and manipulation has to be done using the Parlay X Address List Management Web Service. In the 3GPP presence context, contact lists and group manipulation is done with the XCAP protocol.
3.8. Parlay promises and reality

Because concepts can sometimes be far from reality, this section explores a number of promises raised by the Parlay concepts, and what they become in industry practice.

One must not forget the nature of Parlay: an API (Application Programming Interface). Its definition uses UML (modelling only), IDL (mapping to various programming languages available) and WSDL (description). But Parlay does not replace network protocols. Parlay X offers higher abstraction and WS model, open to much wider Internet application development community, but is still only an interface, not a language (like CPL, LESS) nor a protocol (like SIP, INAP, SMTP).

Parlay can not be used without underlying network communication protocols (e.g. IIOP for CORBA, SOAP over HTTP), nor for connecting network elements. Parlay only provides security at the Framework level: network security issues are not covered, and should be resolved separately in the network (e.g. IP-VPN, IIOP on SSL).

Parlay promises to be available to a wide community of service developers, much larger than the traditional IN service developers. ‘How large’ depends on required knowledge and of their adoption in industry. If IT technology skills are generally available (Java, C++, CORBA), telecom skills and knowledge of specific protocols (SIP, INAP) is still required. The knowledge of Parlay’s abstract network model is not straightforward: state models are rather simple but data types are complex. The standard is still pushed by technology (vendors), but lacks wider operators support, although many are on the path.

Parlay promises to be network-agnostic and to offer network abstraction, a promise met to certain degree. Call Control provides abstract call models that can be used for many network types. Service design is however often dependent on how mappings in Parlay Gateway are implemented (typically the Call Control interface mapping to INAP or SIP). This strong dependency lies in the fact that key mapping documents are not available in the standard, such as MPCC-INAP or MPCC-CAP V3/V4, implying proprietary OSA Gateway mappings. Developers then often need to take care of underlying network features and/or Gateway implementation. With Parlay X, abstraction is higher and mappings to SIP are rather well defined which makes it suitable for integration in NGN networks and mixed Internet/Telecom environments.

Mapping to some widely used languages is available (Java, C++/CORBA). Service design is however still very dependent on development environment. General purpose IDE can fit, but their usage is not straightforward, and specific Parlay IDEs are not widely available (but are emerging).

Parlay APIs enable a wide variety of application developers to create services in much the same manner as generic IT Software is created and deployed today. Fully integrated with the IT World and E-Business capable, they are designed to be technology independent and implementable in many environments. But Parlay’s technological independence means that it is not optimised for a particular computing environment. It uses highly complex data types, which can cause complications in certain computing languages.

Parlay promises to open the operator’s network to service providers and to the IT infrastructure. The fact is that OSA/Parlay remains used in most cases within the operator domain rather than as network opening mechanism. The opening raises several types of issues. Politically, letting third parties have access to the core operator’s IT and telecom infrastructures can be perceived
as a threat to security and confidentiality. Technically, CORBA middleware creates problems when crossing enterprise and service provider firewalls. Economically, operators often have to pay for licences on both Application Servers and OSA/Parlay Gateways, and service providers are reluctant to share these costs.

### 3.9. Parlay’s evolutions

Parlay is still evolving, as the standardization work on Parlay is ongoing.

The following items are some of those addressed in Parlay API Release 6.0:

- Service brokering - capabilities for Service Selection, Service Provisioning
- Feature Interaction and Service Chaining
- Content Management SCF - provides means to introduce application specific content
- Media stream control enhancements, enhancements for application streaming
- Mobility enhancements - Geodetic coding, Quality of Position, Geo-localization converter
- Route translation lookup
- Prompt interaction
- Bulk SMS and MMS messaging
- Subscriber profile
- Address book, calendar functions
- Policy evaluation.

Likewise, the following items are some of those addressed in Parlay X Release 3.0:

- Web Services Framework - Security, Service discovery and Identity assertion
- Parlay X enhancements - Quality of Position, Terminal/Device change
- Message delivery indication, End-to-End QoS control
- Messaging – support for Prompt interaction, support for bulk SMS and MMS management
- Streaming – support for audio/video streaming from/to application
- Subscriber profile management
- Geodetic conversion
- Address book and calendar
- Policy evaluation.
JAIN is a set of APIs that allows to quickly developing new services in providing an abstraction layer to the network elements and protocols.

### 4.1. Architecture

At its core, the JAIN architecture defines a software component library, a set of development tools, a service creation environment, and a carrier-grade service logic execution environment to build next generation services for integrated PSTN, packet (e.g. ATM and IP), and wireless networks. While many new systems may be developed using the JAIN architecture, several basic communications abstractions are carried forward:

Three layers are defined in the JAIN architecture:

- **A Network layer**: this layer defines the communication protocols to use (AIN/IN or SS7 with many protocols - ISUP, INAP, TCAP, MAP… or SIP, MGCP, Megaco, H.323…)
- **A Signaling layer**: this layer deals with the telecommunication switches (SSPs or MSCs or media gateway controllers or H.323 gatekeepers…)
- **A Service layer**: this layer deals with the Service Control Points (SCPs) or any combination of Base Station Controllers (BSCs), Home Location Registers (HLRs), Visitor Location Registers (VLR), and MSCs and with Application Servers.

As illustrated in the figure below, the JAIN architecture includes a Service Creation Environment (SCE) for both trusted next generation network services and untrusted third-party applications. Trusted services (and policies) reside within the core of public networks. Untrusted services are services written by third parties that access functions within the core public networks. Through a secure service provider interface, these third party applications are kept from compromising the reliability or integrity of those networks.

![Figure 40: JAIN architecture](image-url)
The JAIN network topology provides carriers with the ability to deploy next generation network services on devices inside, or at the edge of the Integrated Network, including any Java technology-enabled end-user device. Furthermore, support for all the necessary telephony protocols used between the different network elements in IN, AIN, and IP based (telephony) networks is mandatory. A key aspect of the JAIN component architecture is to move the signaling layer away from proprietary switches into open call control servers, also known as call agents, media gateway controllers, or softswitches. Signaling, the protocols used to establish and terminate communication connections, is the common thread between conventional telecommunications switches and softswitches. The ability to adapt signaling components between networks is paramount to the success of carriers and network service providers.

The following picture is a pictorial representation of where JAIN APIs are defined within a communications platform:

![JAIN APIs Diagram](image-url)

**Figure 41: JAIN APIs**

The softswitch architecture is centered on mapping call control/session interfaces onto the underlying protocol APIs. Since softswitches perform signaling on IP networks, most are equipped with a SIP, MGCP, or H323 underlying protocols. Several softswitches also include SS7 protocols to address interfaces for the existing telephone network. The JAIN initiative offers key components to build open softswitch architectures, and provide service portability through a unified Java interface to develop and deploy next generation services.
The JAIN Framework of APIs is broken into four categories:

- Connection or Protocol Interfaces
- Call Control or Session Interfaces
- Service Logic Interfaces
- Service provider access.

4.2. The JAIN APIs

The JAIN SS7 APIs define Java classes to interface Transaction Capability Application Part (TCAP), Integrated Services Digital Network (ISDN) User Part (ISUP), and MAP. The JAIN IP APIs include SIP and MGCP. This list is supposed to be expanded to meet industry requirements.

4.2.1. JAIN TCAP

TCAP is a layer of the SS7 communications protocol that was developed to add transaction based functionality to the existing telephone network. This includes functions such as maintaining a dialogue with a database or providing the mechanism to access remote switches and activating features within those switches. The TCAP layer is designed for signaling related messages and provides a means for transfer of information from one application at a switch to another application within another network entity. The information passed through the TCAP layer must be transferred between applications, transparently through the network. The JAIN TCAP API specifies a Java API that will provide the interfaces and classes required to initialize and terminate a TCAP session, manage TCAP dialogue identifiers, retrieve, build and send dialogue and component primitives. TCAP provides the means for the invocation of remote operations between signaling point nodes and thus provides generic services to applications such as mobile telephony and free phone service. TCAP is used in the real world today by a variety of applications including:

- «800» Number Translation – One of the first uses of TCAP, where a virtual «800» phone number is converted into a physical route-able phone number
- Calling Name Delivery – A subscriber can see the name of a caller instead of just a caller id
- Do Not Disturb – A subscriber can temporarily block incoming call and revert the calls to a voice mail box based on criteria such as time of day, caller id…

4.2.2. JAIN ISUP

The JAIN ISUP provides an API to the signaling functions needed to support switched voice and data applications. ISUP is defined within the SS7 specifications as a communications protocol used to set-up, manage and release trunk circuits that carry voice and data call over the PSTN.
4.2.3. JAIN MAP

JAIN MAP classes manage the Pan European standard for cellular processing Global System for Mobile Communications (GSM) and the North American standard for cellular processing (IS41). The JAIN MAP interface is concerned with:

- Network Topology: Home Location Register, Visitor Location Register, Base Station Controllers, Mobile Switching Centers...
- Mobile applications such as roaming, hand-off, lookup...
- Wireline interconnect
- Services such as subscriber voting, reporting, short message service, news...

4.2.4. JAIN Operations, Administration, and Maintenance (OAM)

OAM provide a standard portable interface to a service provider to provision and maintain components in a PSTN/IP network. Transmission rates, hardware characteristics, routing configurations, etc. are all part of provisioning a protocol. While other JAIN APIs, such as the JAIN TCAP API and the JAIN ISUP API, define a common interface to proprietary implementations of protocol layers in an SS7 protocol stack, the JAIN OAM API defines a common interface to proprietary management interfaces for an SS7 protocol stack. The JAIN OAM API allows network components creation, deletion, modification and monitoring.

4.2.5. JAIN MGCP

Media Gateway Controller Protocol (MGCP) controls voice and media over packet gateways. Gateways allow for the interconnection of the PSTN with packet networks. This allows users to make calls that span both the PSTN and packet networks. MGCP has been defined as a protocol for controlling these gateways. The JAIN MGCP API allows developers to write MGCP services while the standard is maturing. The JAIN MGCP API will be backward compatible as the MGCP protocol is enhanced. JAIN MGCP API provides an industry standard Java technology interface into proprietary MGCP protocol stacks. It includes the gateway and control interfaces necessary to create, release, modify, and audit connections.

4.2.6. JAIN SIP

Session Initiation Protocol (SIP) enables voice over IP gateways, client end points, PBXs and other communications systems to interface with each other. SIP addresses the call setup and tear-down mechanisms over the Internet. SIP does not include the mechanism for media streaming between caller and “callee” (defines in protocols such as RTP and RTCP¹). Similar to HTTP protocols, SIP is

¹ Real Time Protocol and Real Time Control Protocol respectively
an application client/server protocol, with requests issued by clients and responses managed by servers. SIP enables users to participate in multimedia sessions (or calls) and communicate in both unicast and multicast sessions. JAIN SIP API provides a industry standard interface into proprietary SIP protocol stacks. The JAIN SIP API includes:

- User/Agent Server interfaces
- Proxy Server interfaces
- ReDirect Server interfaces.

4.2.7. JAIN INAP

JAIN INAP is a non-call-related control protocol that allows applications to communicate between various nodes/functional entities of an Intelligent Network. The protocol defines the operations required to be performed between service providers for providing IN services, such as number translation, time of day, and follow me. The JAIN INAP API will be based on the American National Standards Institute (ANSI)/Telcordia Advanced Intelligent Network (AIN 0.2) and International Telecommunication Union — Telecommunication Standardization Sector (ITU-T) CS-2 INAP specifications.

4.2.8. JAIN MEGACO

MEGACO standardizes the interface (Reference N) between the Call Control entity (Media Gateway Controller or MGC) and the Media processing entity (Media Gateway or MG) in the decomposed H.323 Gateway architecture proposed by ETSI TIPHON and adopted by IETF. MEGACO has been defined by IETF MEGACO WG and ITU-T SG-16. MEGACO is central to VoIP solutions and may be integrated into products such as Central Office Switches, Gateways (Trunking, Residential and Access), Network Access Servers, Cable Modems, PBXs..., to develop a convergent voice and data solution.

4.2.9. JAIN H.323

H.323 is the ITU-T standard for “Packet-based Multimedia Communications Systems” and the protocols necessary to achieve the defined system. The H.323 System is designed to move real-time bi-directional multimedia (audio, video, data and fax) across packet-based networks while maintaining connectivity to PSTN, including SS7 circuit-switched networks and H.320 systems. The signaling protocols are defined by H.225.0 and include RAS and Q.931. RAS, or Registration, Administration and Status, handles user management; Q.931, an adaptation and optimization for packet networks of the original ISDN Q.931, handles call signaling; H.323 also uses the H.245 protocol which provides media control by opening the media channels and negotiating the capabilities for each endpoint. H.245 can also change the characteristics of the channel, allowing for asymmetrical rate and codec adaptation between endpoints, features necessary for multimedia interactive communications. JAIN™ H.323 API provides an industry standard interface for implementation of H.323 building blocks: Gatekeepers, Terminals, Gateways and Multipoint Control Units (MCUs) compliant with H.323v.4 and backward compatible with earlier H.323 versions.
4.2.10. JAIN JCC/JCAT

The JAIN Call Control (JCC) and JAIN Coordination and Transactions (JCAT) APIs provide applications with a consistent mechanism for interfacing with underlying divergent networks. The application needs only interface once to a JCC or JCAT interface and the subsequent JAIN adapters will allow calls and data to pass over various networks.

4.2.11. JAIN Service Provider API for the Parlay Specification

The JAIN Service Provider API (JSPA) for the Parlay specification will provide the secure access mechanism to network capabilities. The API will focus on a Java technology based implementation of Parlay and with extensibility to allow other services to be exported by the network operator and discovered by the service provider/user.

4.2.12. JAIN Service Creation and Service Logic Execution Environment

The JAIN Service Creation Environment (SCE) defines a standard set of Java technology-based service building blocks that can be represented in a graphical Java tool or as JavaBeans. The services are defined with interfaces such that the various JAIN SCE building blocks can be integrated together in the SCE to build services. After services are built, they can be tested and deployed in the JAIN Service Logic Execution Environment (SLEE). JAIN SLEE defines interfaces and requirements mandatory for Telco/Internet operations within carrier grade and Internet networks.
5. THE OMA OSE MODEL DETAILED

5.1. Open Service Environment principles

OMA produces open specifications to create building blocks that provide services to end-users or enhance the environment in which services are provided. These specifications have been developed, in most cases, without concern for how they interact with each other, nor do they seek to provide unified and consistent structure by, for example, identifying potential areas of overlap and avoiding duplication.

The term silo has become popular in this context as it highlights the fact that the implementation of services has been done by integrating different components vertically and per-service. Implementation and integration work done for one service can not be reused in others due to the lack of standards. The silo nature of both standards and products results in a number of problems that raise costs and slow down deployment for new services.

The main role of OMA is the specification of OMA enablers. Enablers provide interoperable components, reduce deployment efforts, allow the same applications to operate consistently across a wide variety of environments, and encourage reuse. The OMA enablers, the decomposition into these components and the interactions between them comprise the OSE.

An integral part in the development of the OSE is to promote the reuse of common functions that may be used by other OMA enablers and non-OMA resources, and to create new OMA enablers that provide those common functions. In addition, the OSE encourages the identification of gaps between existing standards in making these gaps potential candidates for new OMA enablers.

5.2. OSE concepts, architecture and interfaces

Intrinsic functions are those functions that are essential in fulfilling the intended task of the specified enabler. For example, the Position Calculation function is Intrinsic to Secure User Plane Location, Authentication is intrinsic to Single Sign On. Non-Intrinsic functions are those functions that are not essential in fulfilling the intended task of the specified enabler. For example, Authentication is a non-intrinsic function to Data Synchronisation.

Any requirements or features that are not intrinsic to an enabler should not be specified within the enabler’s specification. An enabler’s specification should only specify the intrinsic functionality required to fulfil its actual function. The classification of intrinsic and non-intrinsic is subjective, and needs to be done on a per enabler basis.

Enabler specifications should reuse existing specifications where possible. This approach includes the reuse of existing OMA enabler specifications whenever possible (e.g. reuse of presence and group management enablers by the PoC enabler), thus addressing the silo issue. Enabler specifications must specify how to interface to (i.e. invoke) their functions.

In order to protect the underlying resources in an OSE domain from unauthorised requests and to manage the usage of these requests, the OSE enables the exposure of OMA enablers, other functions, resources and applications to each other in a controlled manner.

In any OSE domain, implementations of the OMA enablers expose standard interfaces for application and enabler use. These enabler implementations connect to the actual resources present in the
OSE domain. Through this abstraction, it is possible to add or modify the underlying resources without affecting the interface exposed by the enabler implementations (and therefore without affecting the applications), something that is especially important when using multiple vendors, supporting different network technologies or relying on different providers.

New enablers can be introduced into any OSE domain by developing an enabler implementation that may connect to an underlying resource in that domain. The enabler's interfaces are offered by the enabler implementations for use by applications or other enabler implementations. The interfaces follow the OMA specifications and they are technology-specific implementations of the specified interfaces (e.g. Web services, Java). The enabler's interface(s) can be registered with the (proposed) discovery enabler to allow applications to dynamically bind to the destination enabler.

One way of controlling access to enablers is to use Policies. Policies can be loaded dynamically for policy evaluation and enforcement to protect the enabler. When required, policy definitions may help in extensibility by using the delegation mechanism.

Figure 42: illustrates the architecture elements of both the Service Provider and terminal domains of the OSE. This view focuses on identifying the different elements present in the OSE and their interfaces, but is not meant to be indicative of a deployment model. The OSE architecture does not specify where architectural elements (e.g. applications, enablers, etc.) reside (e.g. they may reside in a Mobile Operator's network or in mobile terminals). The OSE architecture does not mandate the deployment of a Policy Enforcer implementation or of any enabler implementation in any domain, nor a specific deployment model. This allows flexibility in how OMA enablers and the Policy Enforcer function are implemented and deployed.

The figure shows the interfaces in the OSE model:

- $I_0$ is the enabler intrinsic interface (e.g. position calculation for the “secure user plane location” enabler)
- $P$ is a set of parameters used to enforce policies when exposing $I_0$ when the policy enforcement requires additional parameters
- $I_1$ interfaces are interfaces with the execution environment
- $I_2$ interfaces (not defined by OMA) are used by enablers to describe how to invoke an underlying resource's function.
5.3. OSE controlled exposure of enablers and resources

The policy enforcer allows enablers to delegate policies (e.g. authorization, authentication, privacy, charging, etc.). It may also be used to compose enablers into higher-level functions. The interfaces to these higher-level functions, although based on composition of defined OMA enablers, have not been defined in OMA.

The Policy Enforcer applies the same rigid procedures for enablers and applications that reside either in the same environment or across different environments. This is achieved by having the Policy Enforcer process all requests to and from the enabler implementations and enforce the appropriate policies. The specific logic used to handle (e.g. route or intercept) such requests is also part of the Policy Enforcer function, regardless of implementation.

The domain that provides a resource may set policies. These policies may also be combined with other policies derived from preferences or rules set up by end-users or from the terms and conditions (Service Level Agreements) agreed for third parties to use a resource. The domain may also enforce additional policies on behalf of other parties. It will only associate policies (and the resulting evaluation and enforcement) to an enabler implementation able to delegate.

Components providing the policy enforcer function are not required to be deployed in the OSE when deployments do not need policies to be applied to exposed enabler implementations. When an en-
abler is able to delegate functions such as authentication, the domain can supply policies enforced by the Policy Enforcer to perform these functions.

A request originating from an application or an enabler implementation may arrive to the Policy Enforcer through a variety of mechanisms and may be processed in a variety of ways, as illustrated in the following figure.

**Figure 43: Enabler exposure in the OSE model**

**5.4. Using exposed resources in OSE**

As illustrated in the following figure, once an application has discovered the enabler interface (1), and bound to it (2), the policy enforcer controls exchanges with enablers (3) using the bound interface (4). Steps 1a/1b describe two alternative steps at application development time. Step 1c is an alternative discovery process that can take place at execution time.

After the establishment of a relationship, a third party can discover the resources exposed by the domain. This may be achieved through the use of a discovery service or enabler. It is also possible that the interfaces of a resource are communicated with other exchanges between the domain and the application developer when developing the application.

After the applications bind to the enabler interfaces the Policy Enforcer processes the exchanges to control third party access to the enablers. As an architectural element, the Policy Enforcer controls
any exchanges. However, there may be cases when the policy to be applied may be a zero policy whereby the Policy Enforcer does not have to process the request. When a domain owner chooses not to apply policies on requests, since the Policy Enforcer does not process any requests, the domain owner may choose not to deploy the Policy Enforcer.

![Diagram showing the process of using exposed resources in OSE]

**Figure 44: Using exposed resources in OSE**

### 5.5. Migrating from silo enabler architectures to OSE through Policy Enforcement

When deployed, regardless of its realization, the Policy Enforcer has to support enforcement of the domain owners’ policies in a variety of scenarios. Deployed entities in the OSE may interact with each other differently, depending on how the controlled exposure of enabler implementation is achieved (e.g. the existence of policies to be applied on requests, the content of the policies, or the existence of additional parameters (P) required to satisfy policies).

The following figure illustrates how interfaces exposed depend on the content of the domain policies for particular enabler implementations. It illustrates the cases where a Policy Enforcer implementation:

1. Protects an enabler implementation from requests / responses, when P parameters are required to satisfy existing domain policies
2. Protects an enabler implementation from requests / responses, when P parameters are not required to satisfy existing domain policies. In this case, the I0+P exposed to the requestors
by design is actually the same as the I0 of the target enabler implementation (P="0")

3. Is intended to protect an enabler implementation from requests / responses, but only “zero policies” are present for that enabler implementation. In this case, the I0+P exposed to the requestors by design is actually the I0 of the target enabler implementation (P="0"). The dashed lines represent the fact that the Policy Enforcer does not process such request / response

4. Is not needed (no policies exist for some enabler implementations).

![Diagram](image)

### Legend:
- 3P or Terminal domain boundaries
- Deployed entity
- Request flow
- Requestor (E) Application, enabler, other resource
- Requestor (I) External to the domain
- Infernal to the domain

**Figure 45: OSE Policy Enforcer implementations**
6. MIGRATION OF IN SERVICES WITH OSA

6.1. IN-SDP evolution strategies

The Intelligent Network was standardized to offer a variety of services such as free phone or VPN beyond those of a traditional public switched telephony, by moving the services away from the switches onto separate platforms, such as Service Control Points (SCP) and Service Nodes (SN). This idea – the heart of the IN concept - allowed decoupling the development of services from the switches on platforms not necessarily provided by the switch vendors.

But this opening was limited by two main factors. First, the IN protocols are a member of the SS7 standards, a highly reliable protocol but a complex one, only mastered by a few designers. Second, although IN protocols were standardized by ITU-T and ETSI (or Bellcore in North America), most vendors extended it with their own message sets and kept the API proprietary.

In the mobile world, IN protocols were supplemented to enable services like roaming or pre-paid charging, using the CAMEL protocol suite standardized by ETSI.

With the offer decline in traditional circuit-switched technologies, the opening of the telecom environment to providers coming from Internet culture, and the ever-growing need to couple the operators’ network and Information Systems, the replacement of complex IN architectures became an increasing concern.

By providing access to the core functions of the telecommunications network, such as call control, location, charging, account-management, payment, SMS sending, presence management or user-interaction, OSA/Parlay is the key to this evolution.

There are a number of implementations of OSA/Parlay in the market, from providers such as AePONA, Alcatel, Ericsson, Herit, Huawei, Infite, Lucent, Marconi, Telcordia, Nokia. They depend on the network environment (fixed, mobile, 3G, NGN) and on the operator's architecture. Overall, three main evolution scenarios between IN and OSA/Parlay have been observed:

- The introduction of a new network element: the OSA/Parlay Gateway, also called a Service Capability Server (SCS), mapping between applications located on a OSA/Parlay Application Sever, and the Intelligent Network (switches, SCP, CSCF) using SS7-based protocols. The OSA/Parlay Application Server is located within the network. The OSA/Parlay API is not published externally; it is used internally within the network
- SCP upgrade to support the OSA/Parlay API, enabling to develop (in the IN Service Creation Environment) and use (in the Service Logic Execution Environment) services using this API. In this scenario, services developed with traditional IN SIBBs and OSA/Parlay API may coexist
- The development of next-generation IN platforms, which are designed from the start to use the OSA/Parlay APIs internally. In this scenario, the next-generation IN platform will include an OSA/Parlay application server, often externally sourced. All services on the platform are created using the OSA/Parlay APIs.
Figure 46: IN migration scenarios

Conceptually, these options can be viewed as different integration levels of the same bricks as depicted in the figure above:

1. The OSA/Parlay gateway has interfaces to the network and its legacy interfaces (SS7), which can be a separate element and/or a software add-on to an existing SCP
2. The OSA/Parlay AS that constitutes the execution Framework
3. The application itself.

Note that in the first two scenarios, the interface from the fixed or mobile network (HLR, MSC, GMSC) remains unchanged. In particular, the SCS interfaces with the networks using regular MAP or INAP protocols.

Another option is to combine these architectures in the network, and create islands of services only available to a limited category of subscribers (premium service users, MVNO subscribers), such that the infrastructure is solicited for only those services because only those subscribers with a Camel mark (Camel Subscription Information, or CSI) in the HLR will trigger it. This approach offers a good compromise between innovation and risks (from a business and architecture perspective).

6.2. IN call model evolutions

6.2.1. Model mapping issues from traditional IN

When an existing IN service evolves to an SDP architecture, a key compatibility issue is the differences between the call model in the IN and OSA/Parlay environments. This can be declined in three areas:

- Trigger/event management
- Trigger priority handling
- Call model management.
6.2.2. Event triggering and notifications

In IN and CAMEL, a trigger is positioned at a certain stage of the BCSM (Basic Call State Model), and can be associated with criteria (such as called party number). The trigger is performed on behalf of the originating or terminating party, leading to a O-BCSM and a T-BCSM. The trigger mechanism is sequential: each application invoked is identified by a unique Service Key. The triggers are provisioned statically in the switch itself or in the VLR/HLR in mobile environment.

OSA/Parlay API being independent from the underlying network (“network agnostic”), it does not make a difference between a half-call model like in IN, and a full call model like in SIP. OSA/Parlay assumes the network operator, who is fully aware of all services available, is responsible for provisioning triggers. For Multi-Party Call Control (MPCC), OSA/Parlay has been enhanced to add a method “enableNotifications” in addition to the existing “createNotifications”.

“enableNotifications” is a predefined request the OSA/Parlay application can use to implicitly enable or disable operator-predefined trigger notification and criteria.

“createNotifications” is a specific request the OSA/Parlay application can use to explicitly request trigger notification and criteria.

Trigger provisioning data in OSA/Parlay include:

- Call event type (corresponds to Trigger Detection Point in BCSM for IN)
- Monitor mode: interrupt or notification, like the request or notify mode in IN (TDP-R, TDP-N). The interrupt one stops call processing in the network for as long as the service handles the call, while the notify mode does not stop it
- Destination or Originating address
- Additional data depending on the event (e.g. cause value for busy state).

Whether these criteria can be combined or not (e.g. with AND/OR, wildcard match, regular expression patterns, etc..) is vendor dependant in IN and CAMEL. OSA/Parlay defines these capabilities that allow fine-tuning the criteria to exactly match the operator’s needs.

Using a combination of network defined notifications (triggers provisioned by the network operator) and implicit application defined notifications (triggers provisioned by the application) allows handling flexible triggering while being compatible with the triggering defined in the IN or CAMEL environments.

This can be illustrated by the following call flow, which also presents an example of message mapping between IN and OSA/Parlay protocols in the SCS.

In this example, a predefined notification was created (by switch configuration) to trigger when an incoming call (IDP or Initial Detection Point) arrives for a specific called party number range.

Since it was enabled by the application (at step 1), the IDP received at step 2 generates a notification to the application (step3), which then creates a leg for the B-party (step 4), sets up a trigger for the B-party answer detection point (step 5), and then instructs the switch to resume call processing for both parties. (steps 6 and 7).

It is worth noting that in the IN model, steps 6 and 7 translate into a single step 9 (at this stage, only
one leg exists since the B-party has not even been called, while 2 leg objects have been created by the application).

When the B-party answers (step 11), translating into an EventReportBCSM primitive, the Parlay gateway is notified.

6.2.3. Trigger priority handling

Because triggering is a sequential process, several applications can be triggered during a given
call, but not at the same time, for example a pre-paid call to a free phone number will first trigger for credit availability on behalf of the originating party, then for number translation on behalf of the terminating party. When it becomes necessary to choose between two applications triggered at the same detection point with the same criteria\(^1\), these applications can be “packaged” in one “super-application” responsible for the decision making. But this is clearly a workaround.

This limitation was lifted in CS-3\(^2\) and in OSA Release 6/Parlay 5, where MPCC was introduced. To address the interaction between several triggers, the “cascade chain” principle, originally introduced in CS-3 is followed. It defined a “logical order” in which applications are invoked, in the requests (upstream) and the response (downstream) messages. Following this principle allows simulating concurrently active triggers as if they were sequential.

A clear limitation is that the stream order should be TDP-independent; otherwise the provisioning would be very complex to manage.

---

**Figure 48: Trigger priority handling in IN and OSA/Parlay**

### 6.2.4. Call model management

IN and CAMEL have a “switch-centric” view of the call model, where there is an originating and a terminating party. The trigger detection points are defined separately for the originating basic call state model (O-BCSM), and the terminating one (T-BCSM). In this model, each call is declined in two half-calls, with one O-BCSM instance and one T-BCSM instance. Depending on the application, its triggers are set either on the O-BCSM (e.g. free phone) or on the T-BCSM (e.g. call hunt), pertaining to the half call concerned. Likewise, each half-call identifies a call leg.

---

1. An example of that type of conflict: call barring and call forwarding
2. CS stands for Capability Set. CS numbers identify Camel protocol releases.
Parlay was conceived to be network and protocol agnostic. It addresses both the IN view and the SIP/IMS view, where the session is more relevant than the leg (“application centric”). In order to support both models, call model and leg objects have been defined in the Parlay API, including state machines at both levels. This allows compatibility with existing models in all environments. An example of this appears in the previously described call flow where, upon arrival of an IDP, the application creates a leg for the A-party and the (not yet existing) B-party.

### 6.3. SDP-IN message mapping at the OSA/Parlay gateway

The simplicity of the OSA/Parlay API implies that the concrete tasks required to perform the desired actions are deported on the OSA/Parlay gateway.

As discussed before, OSA/Parlay API requires a message to process each of the legs involved in the call, reflecting the adaptability of the call model management the API permits.

When it comes to the details of implementation, the mapping becomes more complex because somewhere the physical resources have to be reserved and managed. This can be illustrated in the following use case where the end user is required to dial digits (e.g. a PIN code for authentication). Depending on whether DTMF recognition is performed locally on the SSF or on a remote SRF, the INAP call flow will be adapted by the gateway which hides the underlying complexity from the application.

**Figure 49: How Parlay abstracts resource handling**
Call flow description:

1. The application sends a sendInfoAndCollectReq message to instruct the gateway to send dialed digits sent by the end user and collect the associated information.
2. The gateway connects to a local or remote resource (gsmSRF) to collect the digits authenticating the subscriber.
3. When authentication is completed, the gateway returns the acknowledgement to the application.

Even the simplest request of all: call routing, requires several INAP messages to encompass event notification handling messages, which must be explicitly requested in INAP, but are implicitly included in the routeReq API request.

![Figure 50: How Parlay abstracts routing and event notification](image_url)

Call flow description:

1. The application sends a routeReq message to instruct the network to route the call.
2. The gateway subscribes to BCSM events like busy tone, in order to be able to react if routing was unsuccessful, and still control the call.
3. The gateway routes the call through the Continue primitive.
4. If routing was successful, it sends a Continue CAP message to release control of the call and let it progress in the network.
7. IMS Services and the OSA Model

7.1. OSA in IMS Reference Architecture

Just like OSA/Parlay interfaces call control switches in the voice-centric environments (the MSC in the GSM architecture or the Central Office in the fixed network), it interfaces the S-SCSF, which is in charge of controlling sessions, in the IMS reference architecture.

7.2. OSA/Parlay Gateway Integration in IMS

The OSA/Parlay gateway can be seen as an evolving component that may interface the IMS through SIP (or Diameter for the HSS) just like it interfaces the GSM network through INAP/CAP. In the IMS reference architecture, it fits between the core network and the Application Servers, and may also interface the MMS with Parlay version 5.
7.3. IMS service deployment strategies

By abstracting the underlying networks interfaces (fixed, mobile 2G & 3G) and protocols (INAP, MAP, CAP, SIP), Parlay fits in the high-level roadmap of telecommunication networks that IMS represents. This also means that a network operator may independently manage the services it offers to customers from the network architecture. This strategy allows innovative services to be launched quickly, while protecting investments made in the service area for example during a 2G-3G transition, or allows the smoothing of migration to NGN architectures. The protocols will be implemented once for all services in the OSA/Parlay gateway, able to interface with legacy environments as well as to evolve to IMS architectures.

In the representation below, we find a model similar to the one discussed in relation to IN migration, where the S-CSCF represents the network interface in this case.
On the application side, the abstraction provided by Parlay/Parlay X makes it available to applications managing information beyond the traditional telecommunication domain, such as those coming from Information Systems, ERP that unleash the power and range of possible applications. Because the interface is independent from the network, access type, vendor, and protocol, it has special appeal to global and integrated operators that may launch a single service on a large scale over all types of interfaces. The deployment can be easily scalable as the number of service users grows. Customers feel a seamless experience as they move between environments, a key feature for highly profitable customers such as international corporate travelers.

This strategy is also made possible by the following key point: an IMS network does not depend on service platforms so much as an IN-based network does. In the GSM world, IN services are critical ones (routing, pre-paid, number translation are examples of this dependency). In IMS, it is possible to introduce a low-end OSA/Parlay gateway without the need to make all existing services fit in this architecture. In other words, the gateway supports a scalability scenario by which it grows as more services are introduced and adopted by subscribers.

Another strategy may also be to launch services on a limited scale and in a basic version as “commodity service”, priced accordingly, and easily accessible to customization by the end-user through a customer self-care interface on the Web. The OSA/Parlay service-oriented architecture permits such evolutions.

This approach can be opposed to traditional deployments based on high-end applications servers. It allows limiting the investments in the initial deployment phase and testing service adoption by the customer in a readily-available environment (such as the Web) before introducing it to a larger scale environment that can be very limited initially (such as IMS).
It is favored by the excellent OSA/Parlay interfacing with billing systems, which offers rapid pricing evolution capabilities as service adoption grows, a key point for operators’ decision makers.

7.4. IMS and the OSA Multi-Party Call Control model (MPCC)

The MPCCS allows an application to establish multi-party calls where several legs can simultaneously be connected. In fact, the MPCCS allows applications to create a leg and to route it. In SIP, establishing a session requires at least two SIP endpoints or User Agents (UA).

MPCCS, which beside 2-party call encompasses application-initiated 1-party and multi-party calls, can be mapped to SIP, implying the OSA SCS behaves as a SIP application server on the ISC interface.\(^1\)

The correspondence between OSA and IMS is described in the 3GPP Technical Report 29.998-04-4\(^2\). Indeed, MPCC is best suited to address the needs of IMS sessions, which are by essence dealing with multiple parties. In this model, the OSA SCS includes a SIP Application Server and offers an API for an OSA-based application.

![Scope of OSA – MPCCS mapping](image)

Figure 54: Scope of OSA – MPCCS mapping

7.4.1. Trigger setting

Filter Criteria (FC) is the information the S-CSCF receives from the HSS that defines the criteria based on which the S-CSCF sends the SIP initial request to the OSA SCS.

---

\(^1\) ISC (or IMS Service Control) is the interface between the served CSCF and the IMS Application Server

\(^2\) OSA API Mapping for OSA Part 4: Call Control Service Mapping; Subpart 4: Multiparty Call Control ISC
Initial Filter Criteria (IFC) are filter criteria stored in the HSS as part of the user profile and are downloaded together with addresses of the assigned application servers (e.g., OSA SCS addresses) upon user registration. They represent a provisioned subscription of a user to an application. Application server specific data is also exchanged between HSS and the OSA SCS during registration (via Sh interface).

Filtering is done in the S-CSCF on SIP initial request messages and can be based upon:

- Any initial SIP method (e.g. REGISTER, INVITE, SUBSCRIBE, MESSAGE)
- Direction of the request is with respect to the served user – either mobile originated (MO) or mobile terminated (MT) to registered user; or mobile terminated to unregistered user
- Session description information
- The present/absent content of a particular SIP header.

### 7.4.2. Mapping OSA Call Session and Call Leg to SIP

In the MPCCS the CallSessionID designates the call as seen from the application, i.e. the ID used to identify a call session. The MPCC API uses it to identify a call session.

In SIP, a SIP dialogue (or call) is identified at each UA with a dialog ID, which consists of a **Call-ID value, a local tag and a remote tag** and by a **globally unique Call-ID**. The Call-ID is created when a user agent sends an INVITE request to initiate a dialog. This INVITE request may generate multiple acceptances (case of group conference), each of which are part of the same call.

For a UAC (User Agent in Client role), the Call-ID value of the dialog ID is set to the Call-ID of the message, the remote tag is set to the tag in the “To” field of the message, and the local tag is set to the tag in the “From” field of the message (these rules apply to both requests and responses).

For a UAS (User Agent in Server role), the Call-ID value of the dialog ID is set to the Call-ID of the message, the remote tag is set to the tag in the “From” field of the message, and the local tag is set to the tag in the “To” field of the message.

However, the semantics of SIP Call-ID are somewhat different from traditional telephony. They identify an invitation of a particular client. This means that a conference in SIP may raise several calls with different Call-IDs. In traditional telephony and in MPCCS this would always be the same call.

In MPCCS, a call leg designates the association between a call and an address as seen from the application and is identified by a callLegSessionID, i.e. the ID used to identify a call leg session. It represents an addressable user in the call.

### 7.4.3. SIP Call-id &dialog vs. OSA Call & Call Leg Session ID

There is a correspondence between the concepts Call and Call Leg in OSA and Call-ID and dialog in SIP respectively. The correlation applicable depends on the mode (e.g. Proxy, B2BUA, UA) in which the controller (OSA SCS) operates. These role models are discussed in further detail in the next section.

There is no easy mapping between SIP and OSA MPCCS call and call leg concepts because the
definition of a SIP dialog always includes two user agents (UA). Therefore, the mapping depends on the SIP server role that OSA SCS plays in a SIP session.

When the controller operates in UA mode there can be a simple 1:1 correlation between OSA call-Leg and SIP call-ID. In other cases a somewhat more complex correlation applies that demands supplementary information such as “To” and “From” header fields in SIP to be correlated with the OSA leg identifiers (callLegSessionID).

For example, if SIP server in OSA SCS acts as a proxy server, then the 2-party call has only one dialog in SIP (between the 2 user agents), while OSA MPCCS expects 2 legs (one from the calling party to OSA SCS and another from OSA SCS to the called party).

Where an application demands full leg control in SIP the SIP server in OSA SCS should always act as UA (or B2BUA), or 3rd party controller. Only the latter mode of operation in SCS achieves a direct 1:1 correlation between SIP dialog and OSA SCS MPCCS call leg.

7.4.4. OSA Call and SIP Dialogue correlation table example

The correlation can be represented by the table below in the case of a (stateful) proxy server. The correlation shown corresponds to the case of an initial INVITE invitation from caller. The Request-URI is a SIP URL that indicates the user or service to which the request is addressed.

The MPCCS callSessionID is assigned by the SCS and represents a correlation to the SIP call-id in the SIP INVITE request message (not a direct mapping because the generation of a SIP call-ID for an invitation is the task of the inviting UA and the creation of a unique callSessionID for an OSA application is the task of the SCS).

<table>
<thead>
<tr>
<th>SIP</th>
<th>Headers</th>
<th>OSA API</th>
<th>Leg</th>
<th>CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dialog #1</td>
<td>call-ID (1)</td>
<td>callSessionID (1), MPCCS Call Object</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>local tag in From header (1)</td>
<td></td>
<td></td>
<td>callLegSessionID (1),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MPCCS Originating Call Leg (1) object</td>
</tr>
<tr>
<td></td>
<td>remote tag in To header (1)</td>
<td></td>
<td></td>
<td>callLegSessionID (2),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MPCCS Terminating Call Leg (2) object</td>
</tr>
<tr>
<td></td>
<td>Request-URI (1)</td>
<td>targetAddress (1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 55: Correlation between OSA call and SIP dialogue
7.5. SIP Server Roles in OSA SCS

The OSA SCS behaves as a SIP server toward the ISC interface (towards the S-SCCF). The SIP application server may act in different roles or modes.

The role of UAC and UAS as well as proxy and redirect servers are defined on a transaction-by-transaction basis. For example, the user agent initiating a call acts as a UAC when sending the initial INVITE request and as a UAS when receiving a BYE request from the callee. The same software can act as a proxy server for a request and as a redirect server for the next one.

Besides these modes of operation, more advanced service applications require the Back-to-Back User Agent (B2BUA) and 3rd Party controller modes. The OSA SCS offers different modes of SIP server operation as described in the following sections.

7.5.1. OSA SCS acting as a SIP Proxy server

In this mode of operation, the incoming SIP Request is simply sent by the S-CSCF to the OSA SCS, which then acts as a SIP proxy server, returning the Request back to the S-CSCF which then proxies it towards the destination. This mode would be used by applications that need to manipulate data conveyed in the SIP signaling between a UAC and a UAS, like changing destination address (call forwarding services), but do not need to intervene on the call as such, and do not want to control it.

Figure 56: OSA SCS in SIP proxy role
7.5.2. OSA SCS acting as UA

When acting as User Agent Terminating, the incoming SIP Request is proxied by the S-CSCF to the OSA SCS which then acts as a terminating UA (UAS).

When acting as User Agent Originating, the OSA SCS acts as an originating UA (UAC) and generates a SIP Request it sends to the S-CSCF, which then proxies it towards the destination.

7.5.3. OSA SCS acting in redirect mode

This mode can be used by applications that need to request a redirection of a call by the network to a new destination, e.g. due to number changed (callee moved). The application must then provide the new contact address(es) and leave the call.

During the Redirect operation, the OSA SCS may terminate the dialog by requesting a call redirection given a list of possible new addresses to contact contained in the redirection response request. SIP messages 301 (“Moved Permanently”) or 302 (“Moved Temporarily”) allow redirecting the call by simulating a destination address change.

![Diagram of OSA SCS in SIP redirect server role](image-url)
7.5.4. OSA SCS acting as a B2BUA

In this case the controller, i.e. the OSA SCS, takes over the ownership of the call set-up by acting as a Back-to-Back User Agent (B2BUA). The OSA SCS acts as a UAS for the INVITE received from caller (UAC), and then as a UAC when it initiates a call to the callee (UAS).

The incoming SIP Request is proxied by the S-CSCF to the OSA SCS which then generates a new SIP Request for a different SIP dialog it sends to the S-CSCF, which then proxies it towards the destination. This mode can be used by service applications that need advanced signaling control, i.e. the capability to intervene on a call. Examples may be applications that need to release a call (e.g. prepaid service) or a single user, add or replace a user (follow-on call), generate messages during the call, or act on mid-call events from a call party (e.g. re-INVITE).

7.5.5. OSA SCS acting as a third Party Controller

In this mode the OSA SCS generates a new SIP Request for a different SIP dialog and sends it to the S-CSCF which then proxies it towards the destination. The OSA SCS may generate one or more different SIP dialogues in this way. This may be combined with the OSA SCS behavior as a B2BUA for the multiple SIP dialogs, i.e. when more than 2 parties are involved in the call.

Figure 58: OSA SCS in SIP third party controller role
This mode allows application-initiated one party, two-party and multi-party calls. It may also be associated with B2BUA mode of operation, e.g. where the application requires to invite a 3rd party into a 2-party. The B2BUA mechanism is one pattern that the 3rd party call controller can use to get control of a session. It allows a controller to take over the control of a session initiated by another party. Once in control, it can handle the session by generating requests and responses on the different call-legs.

### 7.6. Session description for application controlled calls

#### 7.6.1. Principles

In this section, the SDP is understood as the session description protocol, in the context of the RFC 2327, which defines a multimedia session as “a set of multimedia senders and receivers and the data streams flowing from senders to receivers”. The purpose of SDP is to convey information about media streams in multimedia sessions to allow the recipients of a session description to participate in the session. SDP includes the type of media (video, audio, etc), the transport protocol (RTP/UDP/IP, H.320, etc) and the media coding format (H.261 video, MPEG video, etc).

A mechanism is needed that allows a controller like OSA SCS to create, modify, and terminate calls with other entities. Third party call control refers to the ability of one entity, in this case the OSA SCS, to create a call in which communications are actually between other parties. A SIP mechanism for accomplishing third party call control that does not require any extensions or changes to SIP is an application of the tools enabled through the SIP specification RFC 3261. It enables a controller like the OSA SCS to create calls/sessions with any entity that contains a normal SIP User Agent.

The basic principle behind the third party mechanism applied for OSA MPCCS application initiated calls is simple. The OSA SCS acting as a controller on request from the OSA application first calls one of the users, A, and presents the INVITE without any media.

When this call is complete, the OSA SCS has the SDP needed to communicate with user A (returned in the call accept message 200 OK). The OSA SCS can then, if requested by the OSA application, use SDP A to establish a call to user B. When this call is completed, the OSA SCS has the SDP needed to communicate with user B. The information is then passed to user A.

The result is that, upon request from the application, there is an established OSA call leg (SIP dialogue) between the OSA SCS and user A, a call leg (SIP dialogue) between the OSA SCS and user B, and media between user A and user B. The aim here is to keep the OSA application based session control for MPCCS as simple as possible, but also generally usable, and avoid SDP awareness in the OSA SCS acting as the controller.

Note that an OSA application needing to control which media types should be allowed on a call (for example restrict call to a given media type such as voice) can also use the Multimedia Call Control Service (MMCCS), which enhances the MPCCS with multimedia control capabilities.
7.6.2. Example: OSA SCS Application initiated One-Party Call

An example of an application initiated One-Party Call could be a booked «wake-up call», i.e. a call to be set-up at a predefined time and date from the network initiated, by an OSA application using the MPCCS.

The application requests a call to be set-up to user A. The OSA SCS sends an INVITE to the user A, without any SDP (meaning the OSA SCS does not need to assume anything about the media capabilities of the devices). User A responds with its SDP a1, in a 200 OK, which is immediately acknowledged (ACK) with an on-hold SDP generated by the OSA SCS.

Call flow description:
1. This message requests the OSA SCS to create a call object
2. This message instructs the OSA SCS to create a call leg for user A
3. This message requests the call leg for user A to inform the application when the call leg answers the call
4a. The created OSA terminating call leg is requested to route the call/session to the specified destination for user A
4b. The OSA SCS acting as a logical UAo1 generates an INVITE request message with no SDP to S-CSCF providing the destination address of user A. The OSA SCS SIP server is in SIP UA Originating Endpoint mode
4c. The S-CSCF proxies the INVITE request toward user A
5a. User A answers the call and responds with its SDP (SIP 200 OK including SDP a1)
5b. The S-CSCF proxies the SIP 200 OK including SDP a1 to the originating UAo1 in the OSA SCS
6a. The OSA SCS being the controller immediately generates an ACK with an on-hold SDP sent to the S-CSCF. It takes SDP a1, and generates another SDP which has the same media composition, but is on hold. This allows making the call without reserving resources like codecs in the network, which are unnecessary in this case of a machine-to-user call
6b. The S-CSCF proxies the ACK with SDP on hold toward user A
6c. The leg object in OSA SCS passes the result of the call being answered back to the application in OSA AS.
In a telecommunications landscape driven by the ever-growing penetration of Internet paradigms (low cost access, rapid service deployment, unmanaged network, openness) and the massive decline of circuit switching, telecom operators face new challenges like never before in their industry. In this section, we explore some of the telecom industry trends and how SDP concepts can help facing them.

**8.1. From voice transport to content delivery**

Circuit-switched based telephony services have been rapidly declining since the introduction of voice services over ADSL. The pace has been astonishing, surprising even the historical operators. With over 80% penetration in the EU, the mobile is now also a mature market. Its small growth is increasingly captured by new actors like MVNO, who develop content-oriented offers (music, videos) over a basic set of standard telecommunications services. For example, in France, MVNO represented 63% of the growth in Q2/06.

Driven by the massive decline of transport in packet networks, conveying media flows is increasingly perceived as a utility. Users are decreasingly willing to pay for it, while they still are willing to pay for content delivery.

Bundled with basic voice/data transport offers, traditional telephony services (call forward, conference, call waiting.. not to mention voice mail service) are taking the same path, and are less perceived as an added value by the consumer. They become part of the standard package just like the airbag became a standard feature in the automobile industry. However, as the number of services grows, their interactions become increasingly complex and confusing for the end user. SDP concepts can help by providing OSA-based service interaction managers (SIM), which are specifically designed for handling these interactions and can be delivered in both GSM and IMS environments.

The same is true also for reliability. The traditional career core model (five nines reliability, mission critical software design, high availability architectures) is now considered fulfilled and does not drive investments like it used to. Still, reliability is not in the genes of the Internet Protocol like it is for SS7. To illustrate the paradigm shift, in the IP standard, level 3 must drop packets in some circumstances (an IP packet life time is represented by the “Time To Live” parameter, which represents the maximum number of hops the packet can traverse), something that was not even considered in the SS7 world. In an IP-dominated world, reliability is provided by cheap cost of infrastructure dimensioning, rather than by protocols, even if reliability concerns have also invaded the Internet world. Still, subscribers assume reliability is granted and are not willing to pay for its value.

In a market where regulators encourage fair competition (fast portability, local loop unbundling) churn is made very easy for subscribers. Operators then increasingly focus on content delivery (real-time TV, video on demand, etc..) and access to exclusive content becomes a key differentiator. This strategy is illustrated by the leading MVNO in France: M6, NRJ Mobile, and Virgin mobile are all content providers.

Significantly, the traditional IN services that still drive innovation are increasingly tied to a content offer. For example, in order to benefit from the personal ring back tone service, the end user has to buy the music. The spectacular success of high-end content-based SMS+ offers also confirms that end-users are willing to pay high prices for contents like ring tones or logos for their handset.
Major integrated operators are now focused on having access to the largest catalogues of music, video, and live events, and are generally willing to pay high prices, especially for exclusivity (sports, concerts, star interviews).

In the content-delivery battlefield, SDP concepts can be exercised in the platforms in charge of delivering video on demand (streaming), with an efficient integrated access to several network types, billing systems, and various content providers.

8.2. Innovating with fixed-mobile convergence

Today, end-users want a seamless, anywhere, anytime experience with a single high-tech device usable over several networks, without even being aware of which network they’re using: we can call this the ubiquity experience.

What seamless means here is a homogeneous user experience between fixed and mobile networks, especially at home with WiFi or Bluetooth interfaces in UMA architectures and, ultimately, complete handover scenarios (including between fixed and mobile networks).

Triple play (Voice, Internet, TV) and, as fixed-mobile convergence emerges, quadruple play offers are driving the market, adding more complexity and interoperability issues. Tomorrow, Digital Video Broadcast offers could complete this type of offer. What this could mean for operators is that services focused on access (high bandwidth ADSL, FTTH, UMTS or EDGE access networks, HSDPA, UMA, DVB) could be the next market killer and the investment focus.

In the fixed area, revenues of circuit-switched voice calls that used to be cash-flow machines are dropping, driven by increased pressure from regulators and by technology. Traditional competition is increased by the presence of new incumbents, which sometimes don’t own or even manage a network and use the Internet to transport their services (Skype, Yahoo, etc…).

More significantly, mobile revenues, traditionally protected by strong entry barriers (licenses, investments) could also be at threat. Internet providers start packaging offers where the owner of a dual WiFi or Bluetooth + GSM handset could use any network-connected set top box nearby to make almost free “mobile” calls. Doing this is no less than creating a virtual mobile network at no cost, in the same fashion as the UMA model (Unlicensed Mobile Access’), but with no infrastructure.

Even if this type of approach is far from delivering the features of a real mobile network, it can be good enough to make calls from the «home zone» (operators have observed that a significant portion of mobile calls are made from home). Environments like business centers, train stations or airports are the next target. In other words, Internet access providers are starting deploying a cheap “hot spot” offer, without any infrastructure of their own. Significantly, these actors consider the Internet access box they’re selling to be part of their network and not as a customer premises equipment. The box can be potentially shared as a network access point by other customers passing nearby with a dual handset, just as a roaming user would use a foreign switch in the mobile network.

1 http://www.umatechnology.org
In the fixed-mobile convergence area, the key is integrated access. The technology is already there to provide it, but networks are not ready yet to perceive the customer as a single user whether he’s behind a fixed line, an ADSL access, a traditional radio interface, or a WiFi access through a dual handset. Achieving this requires the merger of two major service enablers:

- Billing mediation between networks, and, within the mobile network, between pre-paid and post-paid customers. This mediation platform should be real-time oriented and provide transparent billing for end user, including communities, such as the family.
- Service profile databases, including all technical data (IMSI, IMEI, IP @, fixed line, email, portability), all service subscriptions and authorization modes.

On both topics, in a pre-IMS environment, it is more realistic to build mediation platforms between the various networks rather than imagining a single unified platform. At such, the migration to IMS can be a window of opportunity for SDP environments to become the reference mediation platforms.

### 8.3. Opening networks

The transition from GSM to IMS, the emergence of new business opportunities made possible by the introduction of multimedia and convergence, and the new business models that IMS facilitates with service providers are all strong drivers for opening networks.

Their combination makes the network architecture evolution a real challenge, as different scenarios emerge, that require opening the networks in different ways:

- Migration from GSM to IMS core network. In the services area, it requires the seamless support of Intelligent Network services, which are critical to service continuity (pre-paid service, number translation). The obvious answer is the migration of the service infrastructure towards an OSA/Parlay gateway able to interface both networks. The business case is even justified in a pure GSM network when the core network is shared between two vendors that support slightly different INAP protocols and need a mapping gateway to access a standardized service.
- Introduction of innovative IMS services as prototypes (typically Web services) over a fixed access is a tempting approach (the mobile operator becomes a virtual ISP), but if the service is adopted, subscribers will want it on their mobiles. Designing these new services with Parlay X APIs in the first place would allow migrating them smoothly and with a short time-to-market in a true mobile environment.
- Operators need to differentiate themselves from the competition, today from the MVNO they host in their network, tomorrow from ISP and service providers in the IMS world, while still opening the network on a service basis to partners. Service providers also need access to enablers provided by the operator, and one could imagine differentiated pricing on an enabler basis. For example, the presence or the location enablers are strategic for specific services that provide high added value, and could be priced accordingly. This requires a Framework able to manage efficiently secured authentication and subscription to services and enablers.
- Globalization, merger and acquisitions are economical drivers in a world where global
sence must be combined with significant market positions on a country basis. Still, sy-
ergies are not always what one would expect in a merger because the infrastructures are mostly national and can not be put in common use. Even if services are also largely offered on a national basis today, they should be more integrated tomorrow. In particular, the new convergent multimedia services that IMS promises should be launched on a European scale. This is an opportunity for operators with significant positions in Europe such as Orange or Vodafone to share their service infrastruc-ture. A single AS could be accessed from several national networks through an SDP infrastructure.
This section provides the most important references dealing with SDP concepts and associated links to the relevant Web sites.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Document Type</th>
<th>Web site</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAIN</td>
<td>JAIN API specifications</td>
<td><a href="http://jcp.org/jsr/tech/jain.jsp">http://jcp.org/jsr/tech/jain.jsp</a></td>
</tr>
<tr>
<td>JAIN</td>
<td>JSLEE and the JAIN initiative</td>
<td><a href="http://java.sun.com/products/jain/">http://java.sun.com/products/jain/</a></td>
</tr>
<tr>
<td>OMA</td>
<td>OMA Service Environment architecture</td>
<td><a href="http://www.openmobilealliance.org/release_program/ose_ad_v1_0_2.html">http://www.openmobilealliance.org/release_program/ose_ad_v1_0_2.html</a></td>
</tr>
<tr>
<td>3GPP</td>
<td>3GPP working group 5 (Open Service Access) page and specifications</td>
<td><a href="http://www.3gpp.org/ftp/Specs/html-info/TSG-WG--C5.htm">http://www.3gpp.org/ftp/Specs/html-info/TSG-WG--C5.htm</a></td>
</tr>
<tr>
<td>Parlay</td>
<td>Parlay group</td>
<td><a href="http://www.parlay.org/">http://www.parlay.org/</a></td>
</tr>
<tr>
<td>Parlay</td>
<td>Formal analysis of OSA/Parlay authentication</td>
<td><a href="http://eprints.eemcs.utwente.nl/696/">http://eprints.eemcs.utwente.nl/696/</a></td>
</tr>
<tr>
<td>Sigtran</td>
<td>SIGTRAN charter and specifications</td>
<td><a href="http://www.ietf.org/html.charters/sigtran-charter.html">http://www.ietf.org/html.charters/sigtran-charter.html</a></td>
</tr>
</tbody>
</table>
This section lists the major SDP vendors, with their SDP product line names and URLs. An updated list of Parlay vendors can also be found at [http://www.parlay.org/en/products/](http://www.parlay.org/en/products/).

<table>
<thead>
<tr>
<th>Company</th>
<th>Product line</th>
<th>Web site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alcatel-Lucent</td>
<td>Alcatel-Lucent Application Server (formerly MiLife AS)</td>
<td>Search for «milife» in <a href="http://www.alcatel-lucent.com">http://www.alcatel-lucent.com</a> and click on the first result</td>
</tr>
<tr>
<td>IBM</td>
<td>IBM SDP</td>
<td><a href="http://www-03.ibm.com/industries/telecom/doc/content/solution/1148652202.html">http://www-03.ibm.com/industries/telecom/doc/content/solution/1148652202.html</a></td>
</tr>
<tr>
<td>Kabira</td>
<td>Convergent service broker</td>
<td><a href="http://www.kabira.com/Products/Telecommunications/Service_Broker">http://www.kabira.com/Products/Telecommunications/Service_Broker</a></td>
</tr>
<tr>
<td>Nortel Networks</td>
<td>AS5200</td>
<td><a href="http://products.nortel.com/go/product_content.jsp?segId=0&amp;catId=null&amp;parId=0&amp;prod_id=47181&amp;locale=en-US">http://products.nortel.com/go/product_content.jsp?segId=0&amp;catId=null&amp;parId=0&amp;prod_id=47181&amp;locale=en-US</a></td>
</tr>
<tr>
<td>Teligent</td>
<td>Convergent application portfolio</td>
<td><a href="http://www.teligent.se/">http://www.teligent.se/</a></td>
</tr>
</tbody>
</table>

1 In June 2007, Aepona merged with Appium, a leading Swedish SDP provider. Aepona Causeway and Appium XWay product lines have merged into the Aepona USP.

2 The SDP Alliance is the collaboration of seven telecoms software product companies (Aepona, ChangingWorlds, Cibenix, Mobile Cohesion, Openet and Xiam) that together provide a pre-integrated solution with internal and external enablers.
For any remarks or comments on this document, please contact:

François Mouillaud (francois.mouillaud@devoteam.com) has 17 years experience in the telecom industry and joined Devoteam Solutions in 2000. His assignments in the telecom services area include Sigtran M3UA introduction in a major service platform provider’s portfolio, support of SMS services in production and introduction of the Color Ring Back Tone IN service for a mobile operator, specification of a major evolution of the pre-paid IN service and MVNO offer for a leading manufacturer, and validation of Push To Talk services in IMS environment. An active community member, he created Devoteam University courses (SS7, IN service design training), and contributed to several others (NGN, UMTS). In 2006, he joined Devoteam SRIT, a Lannion-based division that bears telecom integration and development offers for operators and equipment providers. Since August 2007, he is co-leader of Devoteam's Telecom knowledge community.

Patrice Crutel (patrice.crutel@ausystems.fr) was a pioneer in the introduction of OSA at auSystems, and an initial member of the group that launched OSA at Ericsson. As a technical expert on the subject, he created and delivered trainings on OSA/Parlay and was implicated in standardization groups. He is currently involved in the benchmark and development of major Parlay and Parlay X applications, such as fixed-mobile convergence or home zone, for local and global mobile operators.

Delphine Le Grand (delphine.legrand@ausystems.fr) has 8 years of experience in the telecom industry. She has been fully involved in all OSA activities at auSystems. She contributed to a course on the evolution from IN services to OSA/Parlay technology, and is currently involved in the benchmark, specification and development of major OSA applications, based on fixed-mobile convergence, for local and global, fixed or mobile operators.

Member of Devoteam Consulting, Hakim Bessila (hbessila@siticom.com) has 10 years of experience in the industry. He has been involved in network audits, fixed and mobile infrastructure cost reduction projects, and optical fiber deployment with several major industry accounts and network carriers. He also created a technical and market study on service platform evolutions for mobile operators.

Oliver Quaranta (olivier.quaranta@devoteam.com), project director at Devoteam Consulting, has 10 years of experience in Telecom services for operators. He has been working for Mobile operators in various value added services projects within SMS, voice and Wap portals products. As a consultant, he has deep experience in convergent offers and services for broadband and mobile access. He is currently involved in the Media field where he is creating multi device TV/VOD content services.