MASTER

OSA/PARLAY on a SIP network

Cybulski, A.P.

Award date:
2004

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
OSA/PARLAY on a SIP NETWORK

André Cybulski

Coaches: dr.ir. A. Moerdijk (Ericsson Eurolab, Rijen)
Supervisor: prof.ir. J. de Stigter
Date: January 2002

The Faculty of Electrical Engineering of the Eindhoven University of Technology does not accept any responsibility regarding the contents of Master’s Theses.
MASTER THESIS

OSA / PARLAY ON A SIP NETWORK

Student: André Cybulski
Faculty: Electrical Engineering / Information Engineering
Supervisors: Prof. Ir. Johannes de Stigter
           Dr. Ir. Ard-Jan Moerdijk
Company: Ericsson EuroLab, The Netherlands
Date: 31. 01. 2002
Preface

This Master Thesis works out several research questions related to implementation concepts and design of the Third Generation Network. It gives an evaluation of the Session Initiation Protocol capabilities as the core network signaling protocol, examines SIP-based programming methods for creating and providing multimedia services and clarifies relation of SIP to Parlay / OSA service architecture.

This project completes also my education at the Technical University TUE at Eindhoven in the faculty Electrical Engineering with specialization Informatietechniek. By deciding to do my graduation project in the telecommunication area by Ericsson in Rijen I wanted to take a closer look at the process of designing and bringing up to live a new technology for the future telecommunication world. I wanted also to be one of those people that face the challenge to change the way that people communicate with each other. I am very glad I could participate in the one of biggest research projects in the history of telecommunication.

My acknowledgements go to both supervisors prof. ir. Johannes de Stigter and dr. ir. Ard-Jan Moerdijk and to my colleague ir. John Kroeze. I would like to thank them for helpful and effective collaboration. I am also very grateful to Ericsson for giving me the possibility to graduate in this company.

Andre Cybulski

© Copyright 2002, Ericsson EuroLabs Netherlands
Summary

One of the most anticipated developments in the communication technology is the convergence of Internet and mobile networks. This is leading to the new 3G Network based on a new service architecture, that is going to be access and terminal independent. This will enable that the same services are available to the users wherever one is in, independent of the network, terminal type or current location. Within 3GPP, the body responsible for UMTS standards, this concept is called the Virtual Home Environment. One of the VHE enablers is the Open Service Architecture that provides for the applications in the Service Network an open and standardized interface to the Core Network resources. OSA is based on the Parlay APIs that have been adopted by the telecommunication community in the context of 3GPP. APIs are offered by the so-called Service Capability Server that provides the access to the network in a controlled and safe manner. In the Core Network, as specified from UMTS Release 5 and onwards, the Session Initiation Protocol has been chosen as the signaling protocol.

There are still a lot of questions to be answered regarding Service Network, Service Enabler and Core Network architectures. As some of the starting points for OSA and SIP are different the main goal of this Master Thesis, done within Ericssons’s IP Multimedia Task Force was to investigate the relation between the OSA and all IP Network with SIP in its core.

First of all it was needed to evaluate SIP as the signaling protocol with respect to the features that it should support for establishing of the multimedia end-to-end connections. SIP can be compared to ISUP from ISDN protocol and is totally different from today’s session control protocols used for telecom services. The first chapter contains the Master Thesis project definition and a list of research tasks. The second and third chapters have an informative character and introduce to the research topics. They can be used for reference while reading the rest of this document.

After investigation of SIP the research has been focused on the service creation methods in the SIP environment. The main task was to discover how different SIP-based service creation methods compete with each other. The comparison table presents different aspects of the programming methods arranged into the essential stages of the service lifecycle. The resulting conclusion was that SIP Servlets API are the most competitive along SIP programming methods and can be used for providing a variety of services. The key advantages include multithreaded run-time environment, extensibility, standardized framework for adding new servlets, capability to run in a distributed network environment and an easy integration with the Internet technology via existing Java servlets APIs. As this method primary targets the trusted service providers, CPL has been identified as a perfect solution for the end-user service creation. CPL provides excellent portability since the user can store the scripts on the terminal and upload to the server in any network and place on Earth. The SIP REGISTER method can be used for this purpose. If the user may wish to use another terminal it may be required to use the Internet Directory services to store the scripts on the Internet for easy access. Comparison of Parlay and SIP service creation environments showed that both methods are well suited to be used in the 3G service creation concepts. The main advantages of Parlay / OSA in the context of service creation are maturity, support for 3rd parties and usability for not only 3G networks which in fact are not implemented yet but also for 2G network.

As the SCS plays a key role in the providing of IP-based Multimedia Services the next task was to validate whether SCS can offer appropriate mapping of Parlay / OSA interface onto SIP. This addressed issues about mapping of the Parlay methods onto SIP requests methods and responses. Creation and analyze of the sequence diagrams for service execution call flows brought up important conclusions about the leak of support for some functionality as compared to SIP.
There are some valuable features in the SIP protocol that seem not to be accessible for the Parlay application. An example is the media negotiation capability, which is not being supported by the Parlay APIs. The solution to this problem would be to define a new interface or modify the existing one to better expose some protocol specific functions. The validation of the interoperability of SIP with Parlay in the service execution gave an answer to the question how the SIP network architecture should be mapped onto IP Multimedia architecture. It clarified that the SCS and S-CSCF network entities should play different roles during the service execution and they can transitionally work in different modes from UA Client through SIP Proxy and B2BUA to 3rd Party Controller mode.

The last task was to suggest the improvements of the Parlay interfaces in relation to the discovered unsupported features. From the list of issues four most important problems have been chosen for investigation. It was important to find conceptual solutions without drawing into details but in some cases complete solutions have been sketched. The focus was on the application control over the call leg in the network, the ability to specify the media and routing related parameters and on the negotiation capability of the Parlay interface. The first problem originates in the SIP forking capabilities. The second and third deal with the different media types, compression and encryption methods and other essential aspects of the signaling in the network. To solve those problems several solutions have been proposed including new Parlay methods and new data types. The last issue was to add the SIP address type to the set of Parlay address related data types.

Summarizing, the statement can be made that despite of some mismatch in the architecture, interoperability of OSA and SIP is possible. Slightly modified OSA / Parlay APIs can easily provide the access to specific SIP capabilities and thus keep on playing a key role in the service provisioning in an all IP-based Core Network.
Methodology

To achieve good integration of entire project and successfully complete this Master Thesis within well-defined period of time it is helpful to think out some strategy that could ensure the realization of this task. For this reason this graduation project has been divided into several phases that are discussed below.

SIP is a new application layer signaling protocol for establishing multimedia connections brought up to the standards in the early 1999. Because of the significant role it will play in the new telecom world it is necessary to investigate the relation of SIP to Parlay / OSA architecture especially emphasizing the service creation and execution platforms. The first task will be to describe the protocol and identify its capabilities. This will help to take decision related to the network architecture and service provisioning in the 3G network. The main objective should be to discover the advantages and drawbacks of SIP as a signaling protocol. The second important issue is the service creation environment. This requires a solid investigation in the area of SIP and Parlay / OSA programming methods. The SIP programming methods recommended by the IETF include Common Gateway Interface for SIP, Call Processing Language and Sip Servlets API. Those service creation tools will be evaluated and compared to the Parlay / OSA service development environment. The key issue is to position all existing programming tools and identify the best opportunities for multimedia service creation for the 3G Network. This will be realized by presenting the properties of all service creation methods in the form of the table arranged according to the service lifecycle.

The next step should be validation of the interoperability of SIP-based core network with OSA / Parlay interface. The best solution would be selecting a few basic services, which are already supported by Parlay on the interface to the CS Network and analyze the interworking scenarios with the PS SIP Network. Assuming existence of the suitable Parlay to SIP mapping provided by SCS the functional compatibility of SIP and Parlay will be tested. The main goal of this task will be to better understand the service execution signaling issues and related problems. Can existing methods of Parlay APIs provide desired level of control over the service execution in the SIP Core Network and what are the limitations around this issue.

Then the mapping from Parlay to SIP needs to be addressed. This will be accomplished by working out some problems identified in the previous step. This will require detailed understanding of Parlay methods and interfaces and also the signaling schemes and behavior of SIP network elements should be examined. This should bring up some suggestions and implementation proposals for extension and enhancement of the Parlay interfaces.

After all the Master Thesis research will be enclosed with providing conclusions and suggestions related to this work that could be useful for continuation of the 3G Network standardization process.

This document targets the following readers:

- Service Developers
- Service Providers
- 3G Network Architecture Designers
- People involved in the standardization of 3G Networks
- All other people interested in the concept of OSA / Parlay Service Architecture on a SIP Network
This document covers the following issues:

- SIP protocol, SIP network architecture and SIP capabilities
- SIP Service Creation Environment
- Comparison of SIP and Parlay / API service programming platforms
- Validation of the SIP and OSA / Parlay interoperability
- Identification of SIP features that are not supported in Parlay
- Solution proposals

These issues have not been addressed here in details:

- Session Description Protocol
- Handling of Data Streams
- Mapping of Parlay on SIP in SCS
- Service Management
- Business Strategies
# Contents

**PREFACE** ................................................................................................................. I

**SUMMARY** ............................................................................................................. III

**METHODOLOGY** ..................................................................................................... V

**CONTENTS** ............................................................................................................... 7

**ABBREVIATIONS** .................................................................................................... 11

**FIGURES** .................................................................................................................. 13

1. **MASTER THESIS DEFINITION** ........................................................................ 15

2. **OPEN SERVICE ARCHITECTURE** .................................................................. 17
   2.1 **SUMMARY** .................................................................................................... 17
   2.2 **INTRODUCTION** .............................................................................................. 17
   2.3 **NETWORK ARCHITECTURE** ......................................................................... 18
   2.4 **FRAMEWORK INTERFACE** ............................................................................ 19
   2.5 **SERVICE INTERFACES** .................................................................................. 19
   2.6 **SERVICE CAPABILITY SERVER** .................................................................... 20
   2.7 **CURRENT STATUS OF STANDARDIZATION PROCESS** ............................ 21
   2.8 **ARCHITECTURAL PROPOSALS** .................................................................... 22

3. **SIP NETWORK** ................................................................................................... 25
   3.1 **SUMMARY** .................................................................................................... 25
   3.2 **SIP PROTOCOL** ............................................................................................. 25
   3.3 **SIP CAPABILITY** ............................................................................................ 26
   3.4 **SIP NETWORK ARCHITECTURE** ................................................................ 27
   3.5 **SIP MESSAGES** ............................................................................................ 30
   3.6 **SIP ROUTING AND ADDRESSING SCHEMES** ........................................... 31
   3.7 **SIP ADVANTAGES** ....................................................................................... 32
   3.8 **SIP DRAWBACKS** .......................................................................................... 33
   3.9 **SIP MESSAGE FLOWS** .................................................................................. 34

4. **SIP SERVICE CREATION ENVIRONMENT** ...................................................... 35
   4.1 **SUMMARY** .................................................................................................... 35
   4.2 **SERVICE LOGIC** ........................................................................................... 35
      4.2.1 **SERVICE LOGIC TASKS** ....................................................................... 37
      4.2.2 **LOCATION OF THE SERVICE LOGIC** ................................................ 37
      4.2.3 **SIP MESSAGE PROCESSING** ................................................................ 38
      4.2.4 **ARCHITECTURAL ISSUES** .................................................................... 38
   4.3 **COMMON GATEWAY INTERFACE FOR SIP** .............................................. 39
      4.3.1 **LANGUAGE CHARACTERISTICS** .......................................................... 40
      4.3.2 **SERVICE ARCHITECTURE** ................................................................. 40
      4.3.3 **SIP CGI EVALUATION** ......................................................................... 42
   4.4 **CALL PROCESSING LANGUAGE** ................................................................. 45
4.4.1 ABSTRACT ................................................................. 45
4.4.2 LANGUAGE CHARACTERISTICS ................................. 45
4.4.3 SERVICE EXECUTION ............................................... 47
4.4.4 CPL SCRIPT CREATION ............................................ 48
4.4.5 CPL EVALUATION ................................................... 49

4.5 SIP SERVLETS API .................................................... 51
4.5.1 INTRODUCTION ...................................................... 51
4.5.2 SIP SERVLET DESIGN GOALS .................................. 52
4.5.3 OVERVIEW .......................................................... 53
4.5.4 SERVICE DEPLOYMENT ........................................... 54
4.5.5 SERVICE EXECUTION .............................................. 55
4.5.6 SIP SERVLET EVALUATION ..................................... 56

5. PARLAY / OSA SERVICE CREATION ENVIRONMENT .......... 59
5.1 SUMMARY .............................................................. 59
5.2 INTRODUCTION ........................................................ 59
5.3 PARLAY COMMUNICATION PRINCIPLES ......................... 60
5.4 SERVICE DEVELOPMENT ENVIRONMENT ......................... 61
5.5 Parlay API CAPABILITIES ........................................... 62
5.6 STANDARDIZATION BODY .......................................... 63
5.7 REFERENCES .......................................................... 64
5.8 EVALUATION .......................................................... 65

6. PARLAY / OSA VERSUS SIP SERVICE CREATION ENVIRONMENT .......................... 65
6.1 SUMMARY .............................................................. 65
6.2 COMPARISON TABLE ................................................. 66
6.3 CONCLUSIONS ......................................................... 68

7. SERVICE CONTROL IN IMS SUBSYSTEM VIA SIP .................... 73
7.1 SUMMARY .............................................................. 73
7.2 IMS SERVICE ARCHITECTURE ...................................... 73
7.3 IMS SERVICE EXECUTION .......................................... 75
7.4 WORKING METHOD .................................................. 76
7.5 ASSUMPTIONS ......................................................... 76
7.6 RESULTS ............................................................... 77
7.6.1 CALL BANDING ..................................................... 77
7.6.2 CALL REDIRECTION ............................................... 78
7.6.3 CALL FORWARDING .............................................. 79
7.6.3.1 CALL FORWARDING BUSY .................................... 79
7.6.3.2 CALL FORWARDING NO ANSWER ......................... 80
7.6.3.3 CALL FORWARDING UNCONDITIONAL (SCS IN PROXY MODE) 81
7.6.3.4 CALL FORWARDING UNCONDITIONAL (SCS IN B2BUA MODE) 82
7.6.4 PREPAID CALL ..................................................... 83
## 7.7 Open Issues

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7.1 Allowable Service Points of Interest (SPIs)</td>
<td>84</td>
</tr>
<tr>
<td>7.7.2 Loop Detection</td>
<td>84</td>
</tr>
<tr>
<td>7.7.3 SCS and CSCF Operational Behavior</td>
<td>85</td>
</tr>
</tbody>
</table>

## 7.8 Conclusions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>85</td>
</tr>
</tbody>
</table>

## 8. Parlay OSA Enhancement

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Summary</td>
<td>86</td>
</tr>
<tr>
<td>8.2 Parlay / OSA Relation To SIP</td>
<td>86</td>
</tr>
<tr>
<td>8.3 Parlay / OSA APIs Unsupported Features</td>
<td>88</td>
</tr>
<tr>
<td>8.4 Research Problems</td>
<td>90</td>
</tr>
<tr>
<td>8.5 SIP End-Point Addressing</td>
<td>91</td>
</tr>
<tr>
<td>8.5.1 SIP Address Type</td>
<td>91</td>
</tr>
<tr>
<td>8.5.2 Problem Description</td>
<td>91</td>
</tr>
<tr>
<td>8.5.3 Proposed Solution</td>
<td>91</td>
</tr>
<tr>
<td>8.5.4 Open Issues</td>
<td>92</td>
</tr>
<tr>
<td>8.6 Call Leg Control</td>
<td>95</td>
</tr>
<tr>
<td>8.6.1 Summary</td>
<td>95</td>
</tr>
<tr>
<td>8.6.2 Parlay Versus SIP Call Models</td>
<td>95</td>
</tr>
<tr>
<td>8.6.3 Assumptions</td>
<td>98</td>
</tr>
<tr>
<td>8.6.4 Proposed Solutions</td>
<td>97</td>
</tr>
<tr>
<td>8.6.5 Proposed Implementation</td>
<td>100</td>
</tr>
<tr>
<td>8.7 Session Description Control</td>
<td>103</td>
</tr>
<tr>
<td>8.7.1 Abstract</td>
<td>103</td>
</tr>
<tr>
<td>8.7.2 SDP Discussion</td>
<td>104</td>
</tr>
<tr>
<td>8.7.3 Proposed Solutions and Implementations</td>
<td>106</td>
</tr>
<tr>
<td>8.8 Session Parameters Negotiation Capabilities</td>
<td>111</td>
</tr>
<tr>
<td>8.8.1 Problem Description</td>
<td>111</td>
</tr>
<tr>
<td>8.8.2 Network and Service Requirements</td>
<td>113</td>
</tr>
<tr>
<td>8.8.3 Proposed Solution and Implementation</td>
<td>113</td>
</tr>
</tbody>
</table>

## 9. Conclusions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>115</td>
</tr>
</tbody>
</table>

## 10. References

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>119</td>
</tr>
</tbody>
</table>
## 11. Appendix

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1 Planning Tables</td>
<td>123</td>
</tr>
<tr>
<td>11.2 Ericsson Eurolabs Netherlands</td>
<td>124</td>
</tr>
<tr>
<td>11.3 SIP Address Format</td>
<td>125</td>
</tr>
<tr>
<td>11.4 Parlay Basic Data Types</td>
<td>127</td>
</tr>
<tr>
<td>11.5 Relevant Parlay Address Types</td>
<td>129</td>
</tr>
<tr>
<td>11.6 Relevant Parlay Routing Methods</td>
<td>133</td>
</tr>
<tr>
<td>11.7 SIP Forking</td>
<td>137</td>
</tr>
<tr>
<td>11.8 Related SIP Methods</td>
<td>138</td>
</tr>
<tr>
<td>11.9 SIP Loop Detection</td>
<td>140</td>
</tr>
<tr>
<td>11.10 3GPP General Objectives</td>
<td>141</td>
</tr>
<tr>
<td>11.11 Call Establishment On-Hold</td>
<td>142</td>
</tr>
<tr>
<td>11.12 Wireless Application Protocol</td>
<td>143</td>
</tr>
<tr>
<td>11.13 Pint / Spirits Service Architecture</td>
<td>159</td>
</tr>
</tbody>
</table>
Abbreviations

3G  Third Generation Network  
3GPP  Third Generation Partnership Project  
AAA  Authentication, Authorization and Accounting  
ACK  SIP Acknowledgement Message  
ALG  Application Level Gateway  
API  Application Programming Interface  
AS  Application Server  
ATM  Asynchronous Transfer Mode  
B2BUA  Back To Back User Agent  
BGP  Border Gateway Protocol  
CAMEL  Customized Application of Mobile Enhance Logic  
CAP  CAMEL Application Part  
CGI  Common Gateway Interface  
CN  Core Network  
CORBA  Common Object Request Broker Architecture  
CPL  Call Processing Language  
CS  Circuit Switched  
CSCF  Call State Control Function  
CSeq  Call Sequence Number  
DCOM  Distributed Common Object Model  
DNS  Domain Name System  
DTD  Document Type Definition  
ELN  EuroLab Ericsson Netherlands  
ETM  Ericsson Telefoon Maatschappij  
ETSI  European Telecommunication Standardization Institute  
FC  Filter Criteria  
GCC  Generic Call Control  
GGN  GPRS Gateway Node  
GGSN  Gateway GPRS Support Node  
GPRS  General Packet Radio Service  
SGSN  SGPRS Support Node  
HLR  Home Location Register  
HSS  Home Subscriber Server  
HTTP  Hyper Text Transfer Protocol  
I-CSCF  Interrogating Call State Control Function  
ID  Identity  
IETF  Internet Engineering Task Force  
IIOP  Internet Inter-Orb Protocol  
IMS  IP Multimedia Subsystem  
IMTC  International Multimedia Teleconference Consortium  
INAP  Intelligent Network Application Part  
IP  Internet Protocol  
ISC  IP Multimedia Service Control  
ISDN  Integrated Service Digital Networks  
ISUP  ISDN User Part  
ITU  International Telecommunication Union  
J-SCS  Jambala Service Capability Server  
JAIN  Java APIs for Integrated Networks  
JDK  Java Development Kit  
JVM  Java Virtual Machine
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP</td>
<td>Mobile Application Part</td>
</tr>
<tr>
<td>MGCP</td>
<td>Media Gateway Control Protocol</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MMUSIC</td>
<td>Multiparty Multimedia Session Control WG</td>
</tr>
<tr>
<td>MPC</td>
<td>Multiparty Call Control</td>
</tr>
<tr>
<td>MRF</td>
<td>Media Resource Function</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>OSA</td>
<td>Open Service Architecture</td>
</tr>
<tr>
<td>P-CSCF</td>
<td>Proxy Call State Control Function</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PDP</td>
<td>Packet Data Protocol</td>
</tr>
<tr>
<td>PLMN</td>
<td>Public Land Mobile Network</td>
</tr>
<tr>
<td>PS</td>
<td>Packet Switched</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Networks</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Java Invocation</td>
</tr>
<tr>
<td>RTCP</td>
<td>Real-Time Transport Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-Time Transfer Protocol</td>
</tr>
<tr>
<td>S-CSCF</td>
<td>Serving Call State Control Function</td>
</tr>
<tr>
<td>SCE</td>
<td>Service Creation Environment</td>
</tr>
<tr>
<td>SCF</td>
<td>Service Capability Feature</td>
</tr>
<tr>
<td>SCIM</td>
<td>Service Control Interaction Manager</td>
</tr>
<tr>
<td>SCS</td>
<td>Service Capability Server</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SPI</td>
<td>Service Points Of Interest</td>
</tr>
<tr>
<td>SSF</td>
<td>Service Switching Function</td>
</tr>
<tr>
<td>TCAP</td>
<td>Transaction Capabilities Application Part</td>
</tr>
<tr>
<td>TRIP</td>
<td>Telephone Routing over IP</td>
</tr>
<tr>
<td>UA</td>
<td>User Agent</td>
</tr>
<tr>
<td>UAC</td>
<td>User Agent Client</td>
</tr>
<tr>
<td>UAS</td>
<td>User Agent Server</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VHE</td>
<td>Virtual Home Environment</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice Over IP</td>
</tr>
<tr>
<td>W-CDMA</td>
<td>Wideband Code Division Multiple Access</td>
</tr>
<tr>
<td>WAP</td>
<td>Wireless Application Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>Extended Markup Language</td>
</tr>
</tbody>
</table>
## Figures

### Content

<table>
<thead>
<tr>
<th>Section</th>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>Open Service Architecture</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Figure 1. Open Service Architecture overview</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Figure 2. Parlay/OSA Interface Architecture</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Figure 3. Parlay / OSA Interfaces</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Figure 4. History of the standardization</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Figure 5. Services Architecture: scenario 1</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Figure 6. Services Architecture: scenario 2</td>
<td>23</td>
</tr>
<tr>
<td>3.</td>
<td>SIP Network</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Figure 7. Example of Registrar Server functionality</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Figure 8. Example of Proxy Server and User Agent Client functional entity</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Figure 9. Example of Redirect Server functional entity</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Figure 10. SIP Protocol Stack</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Figure 11. Back-to-Back User Agents communication</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Figure 12. Session Redirection</td>
<td>34</td>
</tr>
<tr>
<td>4.</td>
<td>SIP Service Creation Environment</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Figure 13. Service Life Cycle</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Figure 14. Example of Java-based User Agent Client and SIP Server</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Figure 15. Service Logic collocated with SIP Server</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Figure 16. SIP CGI Protocol Stack</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Figure 17. SIP Server - CGI Script Interaction</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Figure 18. CPL Protocol Stack</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Figure 19. CPL Language Construction</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>Figure 20. Retrieval and execution of the CPL script</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Figure 21. CPL decision graph</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Figure 22. SIP Servlets protocol stack</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Figure 23. SIP Servlets API package structure</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Figure 24. SIP Servlet API service architecture</td>
<td>55</td>
</tr>
<tr>
<td>5.</td>
<td>Parlay / OSA Service Creation Environment</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Figure 25. Parlay / OSA System Architecture</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Figure 26. Service Capability Server View (Stand-Alone SCS)</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Figure 27. Parlay call sequence Diagram</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Figure 28. Parlay Protocol Stack</td>
<td>61</td>
</tr>
</tbody>
</table>
7. SERVICE CONTROL IN IMS SUBSYSTEM VIA SIP

Figure 29. IP MM Service Architecture
Figure 30. Service Platform in the Proxy Mode
Figure 31. Auxiliary symbolic names for the S-CSCF interfaces
Figure 32. Sequence diagram: Call Barring
Figure 33. Sequence diagram: Call Redirection
Figure 34. Sequence Diagram: Call Forwarding Busy
Figure 35. Sequence Diagram: Call Forwarding No Answer
Figure 36. Sequence Diagram: Call Forwarding Unconditional (SCS in Proxy Mode)
Figure 37. Sequence Diagram: Call Forwarding Unconditional (SCS in B2BUA Mode)
Figure 38. Sequence Diagram: Prepaid

8. PARLAY OSA ENHANCEMENT

Figure 39. Sequence Diagram: Media Negotiation

Appendix

10. APPENDIX

APPENDIX 1 PLANNING TABLE

Figure 40. Planning Table

APPENDIX 11 ON-HOLD CALL ESTABLISHMENT

Figure 41. SIP call controlled by a 3rd party controller

APPENDIX 12 WIRELESS APPLICATION PROTOCOL (WAP)

Figure 42. WAP Protocol Stacks
Figure 43. WTA Service Architecture and service invocation methods
Figure 44. The task of the WAP Gateway
Figure 45. WTA Framework Configuration
Figure 46. WTA Repository
Figure 47. Successful Channel Installation
Figure 48. Incoming Call Selection service
Figure 49. Push service invocation

APPENDIX 13 PINT / SPIRITS ARCHITECTURE

Figure 50. The Service Control Gateway as interface to SCF / SSF functions
Figure 51. PINT / SPIRITS service network architecture
Figure 52. Latest architectural proposals
Figure 53. Registration to SPIRITS services
Figure 54. PINT Call
Figure 55. PINT Protocol Stack
1. Master Thesis project definition

At this moment telecommunication networks like GSM are based on circuit switched technology. However, within 3GPP it is specified that core network of UMTS will evolve to an all IP (packet switched) network and the Session Initialization Protocol (SIP) has been chosen as the protocol for setting up end-to-end multimedia sessions. Regarding the all IP network architecture and SIP, there is a number of items to be investigated related to the Open Service Architecture.

This graduation project must comprise the following problems:

1. How does SIP relate to the Open Service Architecture and OSA / Parlay APIs in the 3G Service Creation Environment?
   - Which capabilities, does SIP and OSA / Parlay APIs offer to the service creation tools?
   - What are drawbacks and limitations in using SIP protocol and OSA / Parlay interface in the application development environment?
   - What are the strengths and weaknesses of SIP in relation to OSA / Parlay and what are the possibilities for service network developers to use SIP along with OSA / Parlay API's to deploy new value added services in the 3d generation networks.

2. Relations of SIP to OSA in standardization of UMTS networks.
   - How does SIP Application Servers and OSA Application Servers fit together in the 3d generation networks with respect to the 3rd Generation Partnership Project (3GPP) specifications?
   - How can SIP be mapped onto the OSA/Parlay APIs and vice versa in relation to the Ericsson products for the circuit switches core network?
   - What would be the best architecture for deployment of the SIP and OSA/Parlay based value added services as seen from the technical point of view?
2. Open Service Architecture

2.1 Summary

This chapter provides an overview of the concept of Open System Architecture. Introduction presents the latest trends in the telecommunication world and provides a brief description of logical structure of OSA. It presents the history of specification and adoption of Parlay APIs and describes the concept of 3GPP. It introduces also the Service Capability Server that allows applications to access the network capabilities via those standardized APIs. Then the Parlay / OSA service architecture is described followed by a presentation of Framework and Service APIs. This is proceeded by detailed description of SCS an its deployment scenarios. Next to this the current OSA / Parlay standardization is described and the architectural proposals for the network architecture is presented. This chapter has an informative character and can be used latter for reference.

2.2 Introduction

The past years, communication industry is going through a period of explosive changes, which are both enabling and driving the convergence of network functionality for application development. In today's network, applications and services are a part of the network operator's domain and are built using Intelligent Network technology, along with a considerable time to market. However, with emerge of mobility and IP, easy and fast creation and deployment of innovative applications, becomes a challenge. An increasing demand for bandwidth, connectivity features and economy that can not be supplied by the CS mobile networks in the present form, is leading the mobile telecommunication world to start considering IP as the major 3rd generation Core Network technology. The ease of developing new applications together with IP's ability to communicate between different networks has led to IP being seen as a convergence layer that promises to evolve from a mere data platform to a provider of a much larger variety of services. The IP protocol has opened up a whole range of communication applications, which may allow operators to develop totally new value added services as well as to enhance their existing solutions. Because of real-life limitations on how quickly changes occurs in network, both traditional mobile circuit switched services and IP multimedia services need be supported simultaneously.

For this reasons within UMTS standardisation, new service architecture called Open Service Architecture (OSA) has been defined. The Open Service Architecture is based on horizontal layering principles, where applications are logically found in the upper layer, called Application Layer or Service Network. It provides a distributed service solution where applications are able to access the core network functionality by means of open standardized API's, regardless of different core network access technologies. API's are a set of programming interfaces, defined within the Parlay Group and later standardised in 3GPP, the body responsible for UMTS standardisation. They make up the application-network interface and allow the Application Servers within the Service Network, to access the Core Network capabilities, via the so-called Service Capability Servers (SCS's). In the architectural view SCS's are located in the Service Enabler Layer, and can be positioned on the border between the Service Layer and Network Layer.

The open architecture and platforms supported by the IP protocols and operating systems may lead to applications and new opportunities that are more difficult to replicate using a standard switched centralized solution.
But OSA provides an extendible and scalable architecture that allows attaching of new Service Capability Features (SCF) and Service Capability Servers in the future releases of UMTS, with a minimum impact on the applications.

2.2 Network Architecture

OSA raises the abstraction level, which means that it hides network specific protocols and thus simplifies application development. API's enable a new generation of network applications to be developed by Application Service Providers (ASP's) totally independent of the underlying voice/multimedia network. The key benefits of the Parlay/OSA API's are faster development process and reduced expenses.

As indicated, Service Capability Servers (SCS's) are logical entities that implement the API's (SCF's) and potentially interact with the core network. They are found in the OSA layer, referred to as the Service Enabler layer.

The Parlay APIs consists of two categories of interfaces:

- Service Interfaces (Service Capability Server Interfaces). Offer applications the access to a range of network capabilities and network status information.
- Framework Interfaces. These provide the infrastructure capabilities like authentication, SCF’s discovery and registration, fault management, and so forth.

Figure 1. Open Service Architecture overview

Figure 2. Parlay/OSA Interface Architecture
2.3 Framework Interface

The set of SCFs defined within the Parlay/OSA Framework include:

- Trust and Security Management.
- Service Discovery.
- Service Registration.
- Service Factory.

The applications access the functionality of the Core Network, interacting with the Parlay/OSA Framework Interface first. It provides controlled access to the SCS and makes it possible for applications to discover (Service Discovery) and even register (Service Registration) new services in the secure and convenient way (Trust and Security Management). After authorization, the Parlay/OSA Service Interface will create an instance of Service Capability Feature (Service Factory). The reference to this SCF instance is returned to the framework and the framework forwards the reference to the application. From this moment on, the application is able to use the SCF and get access to the required Core Network functionality. Decision whether the application is allowed to use the SCF and under which conditions is defined by so-called Service Level Agreement (SLA) between the Network Operator and Service Provider.

2.4 Service Interfaces

The Parlay/OSA Service Interface include the following API's:

- Call Control
- User Interaction
- Messaging
- User location / User status
- Connectivity management
- Terminal capabilities
- Data session control (OSA only)

The figure below depicts the various Parlay APIs (squares) and internal interfaces (ellipses).

![Parlay/OSA Interfaces Diagram](attachment:image.png)

Figure 3. Parlay / OSA Interfaces
Further, the Call Control API, on which this project will focus on, is made up from:

- Generic Call Control Service
- MultiParty Call Control Service
- MultiMedia Call Control Service
- Conference Call Control Service

### 2.5 Service Capability Server

A Service Capability Server can be deployed on a core network node (for example directly on the HLR), or it could be deployed as stand-alone, separate node that uses a suitable network protocol to communicate with the HLR. As the last approach provides more benefits than the first one, 3GPP has chosen it, when mapping the SCF to the Core Network protocols. The disadvantage of implementing the SCS on the core network node is that when end-user triggers are managed dynamically, based on user location, an application would have to establish connection to all possible network nodes, in order to find the one where triggers are stored, in case when trigger occurs. Besides the SCS is limited to only those API's provided by the core network node. For stand-alone case there are two possible solutions:

- Physical OSA Gateway (All API implementations within one physical node)
- Logical OSA Gateway (Includes Framework and a few SCS components)

The last one, also known as a “distributed approach” offers easy update and introduction of new API's when standardized protocols are used to communicate with the Core Network node. The only advantage of the physical (collocated) OSA gateway is that only one gateway node needs to implement the Framework interface. The Framework Interface due to its critical functionality should be deployed on a network node with a 99.99% of availability. The most common solution will probably include one Parlay / OSA Framework gateway with the ability to register new SCF’s and with basic SCS’s. Other SCS’s will be then implemented on the other network nodes.

For security and performance reasons, it is not meant to enable direct communication between the Application servers and the SCS’s. An SSL-based Firewall will be used to allow secure communication over the network. Also the access to the network resources and information's should be managed in a secure manner. Every SCS can be marked with a set of Service Properties, which should be used by the Framework to validate an application. For this authentication procedure application identifies itself with its own set of allowable services that are captured by the Service Level Agreements created instantly when a new application is introduced to the Framework. This application related data should be stored on the SCS in order to be able to validate the application regarding SLA at any latter time.
2.6 Current status of the standardization process

In 1998 the Parlay group started developing APIs to allow enterprises to access core network capabilities. The first Parlay release (Parlay 1.2) contained APIs for a framework that provided infrastructure functionality like authentication and service selection, basic call control and user interaction, and access to mailboxes. In the next phase (Parlay 2) the framework was enhanced with e.g. capabilities to register new services, discovery of services and load management. Additionally call control was extended with more flavors (multi-party, multi-media, conferencing) and new APIs were added for user location / user status and provisioned QoS (also called connectivity management). Around the same time as the work on Parlay 2 started, 3GPP and ETSI started working together on APIs to be used for service development for 3rd generation networks. Unless some differences between 3GPP and Parlay, it was realized that the Parlay APIs can be used in the 3GPP / ETSI context as well. Therefore the relevant Parlay APIs were brought to 3GPP and ETSI for standardization and were referred to as OSA (Open Service Architecture). In addition to the relevant Parlay APIs new ones were defined for terminal capabilities and data session control (GPRS) in order to support multimedia. Furthermore, mappings from the API to network protocols have been produced. During this time a lot of effort was spent on aligning the APIs of Parlay and 3GPP / ETSI.

Although the API details were aligned still two different namespaces were used, which led to technically different APIs and thus different developer communities. To achieve one developer community Parlay, 3GPP and ETSI got together and nowadays work on the APIs that is common to Parlay, 3GPP and ETSI is done jointly. In the past, for reasons of ensuring interoperability, telecommunications standards bodies have normally focused on protocols rather than APIs. Now, because of main aim of the Parlay Group to encourage application development, the focus is placed on an attractive API's. In the near future the solution market will ultimately choose which underlying protocol-stack standards should be deployed to support the APIs and its future extensions. At this moment 3GPP has produced Rel.4. The OSA Rel.4 work has been finalized in June 2001 together with Parlay 3.0 and ETSI OSA version 1. Currently the focus of the joint group is on the multi-media aspects and user profile.

Figure 4. History of the standardization
2.7 Architectural proposals

It has already been mentioned, that the core network of UMTS will evolve to an all IP (packet switched) network and the SIP (Session Initialization Protocol) has been chosen as the protocol for setting up end-to-end multi-media sessions. Regarding the all IP architecture and SIP there are a number of items to be investigated related to the OSA architecture. Both 3GPP and ETSI telecommunications community that has adopted PARLAY, have recently also adopted SIP. SIP comes with its own service architecture. The architecture is promising, but is still in its infancy, and does have a few shortcomings. There are also a number of SIP capabilities that would certainly enhance some services, but unfortunately can not be accessed via the Parlay/OSA interface. PARLAY could be used in complement to the service creation tools currently being specified by the IETF for SIP. Furthermore it can play a key role, as common tool, for service creation in hybrid circuit switched/SIP environments.

Now considering the standardization process, there are two main concepts for deployment of Parlay/OSA services. The first scenario relies on the core SIP specification while the second one makes use of the set of SIP – extensions proposed by IETF. Both are presented on the diagrams below.

**Architecture Scenario 1**

In the figure above the SIP protocol is used along the interface between the SIP Client represented by UE and the serving multimedia server (S-CSCF). The CSCF is a SIP server. It implements also Proxy and Redirect Server and includes User Agent behaviors. It can be used to provide multimedia session initialization to all kinds of SIP-enabled terminals in the IP network and can support establishment of GPRS-based data transport connections. The Call State Control Function communicates with Service Capability Server (SCS) via the Sc reference point using a set of APIs. The appropriate SIP to Parlay mapping should be made before accessing the services provided by SCS.
SIP will be probably also used to communicate with Multimedia Resource Function (MRF), which is responsible for bearer control and service validation for multiparty / multimedia sessions. The HSS functionality is to register and at any latter time to provide the CSCF with user location information for mobility management reasons and it stores and maintains the user profile with all registered services and user preferences. Further, standard OSA interface is used to communicate with the OSA Application Servers in the Service Network. In this scenario, a number of specific issues can be identified, where support for SIP could be improved in Parlay. None of these changes should be major, and should not affect the usage of Parlay with other signaling protocols. The enhancement of Parlay APIs is addressed in the last chapter number 8.

The second approach, shown in the following figure, assumes enhancement and standardization of existing SIP protocol to provide service control support. The decision of the 3GPP to use SIP as the signaling protocol for third generation networks means that SIP will become highly significant for next generation networks. Although Parlay has been designed as a network-independent call control layer, it would be of limited value if it could only support the common minimum features of the different network signaling protocols. Indeed, Parlay already contains some optional parameters, like the address type for example, which allow fuller support of specific network protocols. But now it is very likely that Parlay specifications will require some changes in order to take advantage of the capabilities offered by SIP which are currently not accessible via the Parlay interface.

3GPP has adopted both solutions. The final 3G Core Network architecture is presented in chapter 7 on Figure 29.
3 SIP Network

3.1 Summary

This chapter is intended to give a general description of the SIP protocol. After short historical overview of SIP development within IETF the capabilities of the protocol will be presented along with brief presentation of its messages set. Then, the network architecture will be discussed to enlighten different types of network nodes that can be put in the core of or on the edge of the SIP network, depending on its functionality. Latter the focus will be on SIP addressing scheme and QoS issues. Then finally follows an attempt to list some advantages and drawbacks of SIP as signaling protocol.

3.2 SIP protocol

Over recent months there has been growing interest in the use of SIP as a protocol for multimedia session control. Furthermore, the decision of the 3GPP to use SIP as the signaling protocol for supporting multimedia sessions in IMS subsystem in the future network means that it is now likely that SIP will become highly significant for next generation networks.

The Session Initiation Protocol (SIP) is a text-based protocol, similar to HTTP and SMTP used to setup, modify and terminate Internet media communications. SIP was developed as an open standard by the MMUSIC working group of the IETF and was released as RFC 2543 in March 1999. From the end of September 1999 SIP has been transferred into a group of its own for future development (SIP working group of IETF).

Whilst SIP saw relatively slowly progress throughout 1996-1997, it has gain its acceptance in the first year of interest for Internet Telephony in 1998 resulting by the end of this year in completion of its specifications. From there, industry acceptance of SIP grew exponentially. Vendors and service providers have finally discovered its scalability, extensibility and flexibility what accelerated its adoption in the application and service environment. It was the matter of time for SIP to attract the standards bodies like ITU, ETSI, IMTC and JAIN and to start deployment of real SIP networks.

SIP, fundamentally, is a control channel for establishing other sessions, which could for example be a media session. It doesn't provide the media transport itself even does not handle the negotiation of the session parameters. It relies on SDP to exchange the terminal capabilities and uses RTP protocol to handle the audio or video data. Because SDP is the most widely used session description protocol together with SIP it will be assumed from now on that SIP inherits the capabilities of SDP. Hence SIP as transport protocol for SDP can easily support the media type and other session parameters negotiation.

SIP was designed within certain assumptions in mind: scalability and component reuse. Since users can reside anywhere on the Internet, the protocol needs to work with a variety of protocols and should be applicable with all future extensions on IP network within next years. Users could be invited to lots of sessions, so the protocol should be scaleable in both directions. It has been also taken into consideration to develop a new protocol with the minimum of effort and to make it as easy as possible. So rather than inventing a new one the already existing tools, developed within the IETF were used. That includes things like MIME, HTTP, URL and SDP. This resulted in a protocol that was well integrated with IP applications such as web and e-mail.
3.3 SIP capabilities

SIP can simply handle initiation, modification and termination of session but doesn’t explicitly define what kind of session it is. It can be used for transport of SDP packets instead, which can describe the nature of session being established and session parameter negotiation. The network service support and functionality provided by SIP can be described in the following way:

- Invitation and determination of user location.
  Every user might have a few SIP addresses at home and at work associated with a number of PC’s or mobile phones. Since the session initialization is one of the most difficult aspects, it is due to SIP to determine the address where the user being connected actually resides at a particular moment.

- Delivering the description of the session.
  Since the called party has already been contacted it’s needed to inform the end-point terminal which kind of session is being set up. SIP involves Multipurpose Internet Mail Extensions (MIME) to specify the protocol to be used (SDP, HTTP, Audio, Voice, and so forth) and the content. The SDP protocol has been chosen as most suitable for session description.

- Determination of user capabilities and session parameter negotiation.
  After both end-points has agreed on protocol to be used for further communication, SDP encapsulated in SIP messages carries over the capabilities of terminals and a set of acceptable parameters can be negotiated.

- Determination of the user availability and delivering of the response.
  If user has been located, session description delivered and call accepted, SIP is used to convey the response to the originator and can be used at any letter time to modify the agreed set of parameters. Everything should be done is just simply reinitiate the session by sending the same INVITE message as before but with a new session description contents. So adding or removing audio and video stream, switching between different codecs or hold and mute can be as easy as SDP supports those.

- Network Resource Reservation.
  After signaling has completed, SDP can be used for establishing of virtual pipe along the network to provide the minimum latency in delivering of packets during the session. If it fails to agree on some QoS parameters, the session establishment will still proceed but relying only on the well-known best effort delivery mechanism along the network.

- Termination of the session.
  When session already in progress, SIP can be used anytime to terminate the connection and free up the network resources.

26
3.4 SIP messages

SIP is based on the request-response communication model and consists of two types of messages: requests and responses. All messages to establish, modify and terminate the session use the same Call ID, but different CSeq numbers.

The request messages are:

- INVITE
- ACK
- REGISTER
- CANCEL
- BYE
- OPTIONS

The responses describe the status of the requests and are three digits codes.

Here follows two examples of SIP messages for REGISTER message with no SDP content and for INVITE message including negotiation parameters:

**REGISTER** sip:ericsson.se:5060 SIP/2.0
CSeq: 1 REGISTER
Via: SIP/2.0/UDP 150.236.30.107:5095
Contact: sip:voice@ericsson.se;q=0.5
Contact: sip:uffe@erm.ericsson.se;q=0.1;expires=3600
To: sip:ulf@ericsson.se
From: sip:ulf@ericsson.se
Content-length: 0
Call-id: 4@ericsson.se
Expires: 7200

**INVITE** sip:ulf@sun31.su.erm.ericsson.se SIP/2.0 (1)
via: SIP/2.0/UDP 150.236.30.108:5062
from: sip:thomas@su-pc43
to: sip:ulf@sun31.su.erm.ericsson.se
call-id: 945197353100@150.236.30.108
cseq: 22853 INVITE
user-agent: Ellemtel-PICo/R2H
contact: sip:thomas@150.236.30.108:5060
content-type: application/sdp
content-length: 252

v=0
o=ermthho 945197353100 945197353100 IN IP4 150.236.30.108
s=Basic Session
c=IN IP4 150.236.30.108
t=945197353100 0
m=audio 2328 RTP/AVP 8 0 96 98 99
a=rtpmap:96 SC6/6000
a=rtpmap:98 SC3/3000
a=rtpmap:99 RT24/2400
3.5 SIP network architecture

Example services that can be supported by SIP:

- Voice
- Video
- Video-mail
- E-mail
- Buddy Lists
- Instant Messaging
- Online Games
- Virtual Reality Rooms
- Emergency calls
- Image, JavaScript, Java Applet, MP3 Transport.

Here follows a list of SIP network nodes that make up a SIP network.

- **User Agent (UA).** Receives and sends SIP messages. It can play the role of the client and the server as well
- **User Agent Client (UAC).** Sends SIP requests
- **User Agent Server (UAS).** Accepts calls and sends SIP responses.
- **Location Server.** This network element can be collocated with Registrar or Redirect Server and is used to request information about all possible current locations of the party being called. Some non-SIP protocol that is suitable for real-time storing and retrieving of user related information could be used for this purpose.
- **Registrar Server.** This element, usually collocated with Proxy or Redirect Server, may be adapted to provide the location-based services. This server is capable of receiving a REGISTER message, access and check user profile and can retrieve and store user location. One user can be registered at several different addresses, and then an attempt to contact him will make a Proxy Server to issue an INVITE messages to all possible addresses. This so-called address binding must be updated periodically otherwise Registrar Server will remove current user associations with those addresses.

![Diagram of Registrar Server functionality](image)

**Figure 7. Example of Registrar Server functionality**

- **Proxy Server.** Utilize UA functionality of SIP server and SIP client with additional ability to modify SIP messages and make the routing decisions. It can interpret and rewrite the SIP message but also can issue a CANCEL message to the originating party if the connection times out, but generally does not control or maintain the state of the call control. Every end-to-end communication must proceed via at least one Proxy Server that will add its own identity in the message header to specify the return path for the message. It can also fork the INVITATION message to multiple addresses where the called party appears to be registered. This ability allows reaching multiple locations in the hope to find the user at one of them.
The Proxy Server performs the following actions:

- **Proxying of Requests:** Receiving a request, adding or modifying any of the headers, deciding on a set of servers to forward the request to, and forwarding it to them.
- **Returning Responses:** Receiving a response, adding or modifying any of the headers, and passing the response towards the client.
- **Generating Requests:** Creating a new request, originating at the server, placing headers and a body into the message, and sending it to a server.
- **Generation of Responses:** Receiving a request, generating a response to it, and sending it back to the client.

There are general three types of Proxy Server:

- **Call-Stateful Proxy.** These network elements are found on the edge of the network and provide a rich set of services, based on call state.
- **Transaction-Stateful Proxy (Stateful Proxy)** is used closer to the core of the network and track only requests and responses without the knowledge of session or Call State. Once the end-to-end connection has been set up this Proxy Server breaks the association with the session and handles all following messages totally independent from the previously established session.
- **Stateless Proxy.** This server is located in the core of the network and keeps no tracks of forwarded SIP requests and responses.
- **Redirect Server.** Unlike a Proxy Server it can not initiate a request, nor accept the calls or to deal with the call control. This server will communicate with the Location Server to obtain the current address of the called party. It returns zero or more addresses in its response by removing the original address and adding a list of addresses the party being called is associated with. It allows contacting next hope server directly.

As already stated SIP is underlying protocol independent and can be transferred by TCP and UDP as well. SIP uses SDP for session description, the RTP (Real-time Transfer Protocol) for media transmission and the RTCP (Real-time Transport Control Protocol) as the control mechanism for getting feedback information about the session. Normally the RTP uses an even port number and the RTCP uses the next higher (odd) port number. The following two sections will describe very briefly these two protocols. The following figure clarifies the SIP protocol stack architecture.
3.6 SIP routing and addressing schemes

SIP uses address format very similar to popular e-mail URL’s and usually a simple addition of “sip:” prefix before the mail URL will converts it to a valid SIP-address i.e. sip: user@host. The “user” part can be a user name or a telephone number and the “host” part can reflect domain name or a numeric network address. If the host is an Internet Telephony gateway, the user field may directly encode the VoIP phone number, which allows the letters or numbers as well. The standard fixed network or mobile phone numbers can also be used but this rises some problems with address interpretation. Those E.164 phone numbers defined by ITU are limited to 15 digits and are globally unique. The only underlying problem at the SIP server is distinguishing between the “real-ISDN-phone-numbers” and a “service-names-which-looks-like-a-ISDN-number”. The intention is to enter the telephone numbers that do have resources somewhere on the Internet associated with it into the Internet DNS domain system. The DNS lookup mechanism can be then used by a SIP client to discover the resources globally associated with a given phone number or user ID in combination with some predefined assumptions. The SIP Proxy Server will do first the necessary number-to-domain translation and then search the DNS lookup table to identify the URL that should be contacted and all this just after dialing the unique E.164 number. The only requirement is the existence of the resources in the network associated with that number. For other PSTN numbers, having no corresponding entry in the DNS, SIP server will forward the session request to some default PSTN gateway. A special TRIP (Telephony Routing over IP) Servers usually collocated with SIP Servers use BGP protocol to maintain and exchange the information’s on what gateways are available to establish calls to a ranges of PSTN phone numbers. This allows for multiple service providers to efficiently route the signaling through each other’s gateways avoiding at the same time unnecessary costs. It should be noted here that the owner of a network or a domain name is actually the only one that decides how to route the signaling messages in the network.
### 3.7 SIP advantages

SIP protocol has been designed by the Internet experts as an application-layer protocol for creating and terminating calls. But it can be also used for implementing much complex services like:

- Call redirection between different terminal types
- Time-of-day routing
- Stereo and Hi-fi sound over the network
- Multiplayer mobile gaming with voice channel
- Conference/chat with:
  - private rooms
  - whiteboarding.
  - application sharing
  - shared web browsing
- Multimedia messaging
- Emergency location with voice conversation, navigation and picture transfer
- Image/video clips
- User profile transfer
- CPL or CGI script transfer
- Contact buddy list
- Announcements

Probably also Mobile e-commerce, Banking, Insurance agents and many other services could be implemented with SIP.

Here is the list of the SIP overall advantages.

- Separation of the call setup signaling from the media channels. This allows for better use of the network resources.
- Advanced support for mobility by proxying and redirecting requests to the user current location.
- Can be used in combination with any existing datagram or stream conference control protocol like ATM, UDP, TCP, RTP, and RTCP.
- Contains small set of messages and is easy to understand and work with for applications developers
- Based on highly successful and efficient HTIP transport protocol and easy to integrate with existing IP networks
- Quick establishment of end-to-end connection with just one INVITE message
- Support for SDP what gives the possibilities to negotiate end-user terminal capabilities and server features
  - Negotiate capability for media types and audio codecs used for communication
  - Interrogating for terminal capabilities
  - Quality of service support
- Simple text based protocol
- Easily extensible and scalable
- Supports for data encryption
- Support for Unicast, Multicast (call forking) and full Mash communications over the network.
3.8 SIP drawbacks

- NATs. One of the most important problems with SIP is when SIP messages have to pass through the Network Address Translators (NATs). IP V4 Network Address Translation (NAT) has become an increasingly common function in the Internet for a variety of reasons. NATs are used to interconnect a private network consisting of unregistered IP addresses with a global IP network using limited number of registered IP addresses. NATs are also used to avoid address renumbering in a private network when topology outside the private network changes for variety of reasons. A large number of Internet applications work transparently with NATs, but unfortunately it is not the case with SIP. NAT has the potential to interrupt end-to-end nature of Internet applications, thereby threatening end-to-end security and other end-to-end functions. In addition, NAT has topology restrictions and other constraints on the protocols and applications that run across NATs. NATs break many protocols that act as establishment for other protocols such as SIP. They provide a boundary between the private IP addressing of a network and the public Internet and are primarily used by an enterprise it's not able to access some blocks of IP numbers from their ISP in a secure way. When using SIP, the signaling have to pass through NATs causing a problem, because the IP addresses and port numbers for the media session being established are encapsulated by SIP in the body containing SDP packets that actually carry all those information. SIP is an application layer protocol for initiation of multiparty calls and conference sessions so there can be multiple sessions within a particular call. Since SDP carries IP addresses and not Host Names, the external caller User Agent will send media to an IP address that is not globally routable but is only an valid within that private network. Custom-specific Application Level Gateways (ALGs) are required to perform translations for those applications.

- Firewalls. The media content in communication between the users behind a firewall can get lost and connection gets broken as well as with NATs. It is due to that media is sent on dynamic ports and to dynamic addresses. The possible solution to this problem is to have rules dynamically installed through application aware devices such as proxies.

- Ambiguity. SIP-syntax parsers can in some cases hardly distinguish between different contexts of SIP messages containing the same message-parameter combination. As result, a wrong interpretation of messages can lead to undesirable signaling messages.

- Easy extensibility can lead to incompatibility between different SIP based application service providers.

- Can be difficult to synchronize with the network resource reservation mechanism and, if used in the centralized network, to maintain the call control because of its own signaling capability for session initiation.

- SIP encapsulates the IP addresses and port numbers for the media session in its body what can cause routing problems by the SIP Proxy Servers.
3.9 SIP message flows

The following figure illustrates the B2BUA Communication.

![Figure 11. Back-to-Back User Agents communication](image)

The following figure illustrates the redirection functionality of SIP Redirection Server.

![Figure 12. Session Redirection](image)
4. SIP Service Creation Environment

4.1 Summary

One of the most anticipated developments in technology is the convergence of Internet and Mobile networks. The SIP protocol, model consistent with HTTP, has been selected by the Third Generation Partnership Project (3GPP) as the signaling protocol for establishing end-to-end connections in the 3G Networks. As developing services requires APIs, several interfaces have been designed to program the services delivered by SIP.

The focus of this section will be on the SIP-based service creation and deployment. First we will discuss the general issues behind the SIP-based service execution by the SIP Server. Next we will present currently available tools for programming services for SIP platform including Call Processing Language (CPL), SIP Common Gateway Interface (CGI) and SIP Servlets. Finally all presented service creation methods will be compared and evaluated in the context of the service lifecycle.

4.2 Service Logic

There are a variety of signaling protocols used for establishing end-to-end connection on the Internet like H.323, Media Gateway Control Protocol (MGCP) and SIP, to call a few. SIP has been chosen by 3GPP as the protocol for establishing the data path between the parties involved in the multimedia session. SIP is a well-suited programming platform because of similarities with HTTP protocol. One of the powerful advantages of SIP is the separation of signaling from the media channels and the ability to fork one SIP request to multiple addresses in order to locate the user. SIP provides also the functionality to avoid the routing loops, exchange the terminal capabilities and to maintain the state of transaction being established.

This part of Master Thesis is about how to implement SIP-based services in the 3GPP networks. To clearly indicated with which stage of service development process we deal the Service Life Cycle is presented below.

![Service Life Cycle Diagram]

Figure 13. Service Life Cycle
SIP Server processes SIP messages according to the SIP specifications (RFC 2543) but the range of services it can provide is limited to the basic protocol specific actions. For this reason an enhancement on the message processing by the server is needed to meet the large variety of different 3GPP services.

An example protocol stack implemented in the SIP Server and relation to the Service Logic components is presented below. The server is suggested to be a Java Server but this is not a requirement.

![Figure 14. Example of Java-based User Agent Client and SIP Server](image)

The decisions on ...

- which message should be sent
- where the message should be send (destination or next hop address determination)
- at which time should the server sent the message
- whether message needs to be modified
- how the packets should be formatted, etc...

... can be made based on some service logic added to the SIP Server. The information how to process the message can be obtained by the service logic from the message itself (headers or body content) or from external sources like local or remote databases and scripts. Service logic can also make use of variety of services available on the Internet like:

- Web pages
- DNS services
- Directory services
- Internet mail services
- QoS control services
- Policy repositories
- Instant messaging services
- 800 number services and other PSTN / ISDN services
- Media servers, etc...
4.2.1 Service Logic Tasks

The service logic will communicate with the SIP Server instructing it how to process incoming SIP messages or even make the server to issue a SIP requests totally independent of incoming network events. The service control application can also generate the content to be sent entirely on its own and forward it to the server. The decision whether or not the SIP Server should consult the service logic on reception of a SIP message can be controlled by the administrator set local policy or by the service logic itself.

The only thing needs to be done is to provide an interface between the SIP Server and the service control logic. Depending on security requirements and the confidence level in respect to the service creator the interface can be simple (text-based) or more complex (object-based). In the last case the service control application will use a very few control functions interfacing to SIP Server in order to keep the interaction save and secure. This approach allows the services to be easily created directly by the end-users and uploaded to the server for execution. Because the end-users or third party service providers are seen as untrusted the execution environment of the service logic, available resources and the interface to the SIP Server will be limited and strongly controlled by the network administrator. If the network administrator provides the services, the simple, text-based control interface can be used allowing extended functionality and better flexibility.

4.2.2 Location of the Service Logic

The best solution would be to have a service control application running on a separate server. Keeping the service logic separate from the SIP Server insures persistence of the service control functions even if the server crashes and increases the overall communication security level. A specific protocol or well known distributed technologies like Java Remote Method Invocation (RMI), CORBA or DCOM could be used as the interface between the application and the server. This approach introduces considerations about losses and delays specific for every network and the need for using encryption. In this case the service logic of a single SIP Server can be distributed over the network and executed by multiple computers. In co-located approach the interface is very simple and can be implemented with a single API. One possible solution with service logic co-resident with the SIP Server and API-based interface is presented below.

![Service Logic Diagram](image-url)

**Figure 15. Service Logic collocated with SIP Server**
4.2.3 SIP message processing

The basic model for processing of incoming messages is as follows:

1. Server receives the message
2. Server determines whether or not the service logic should be invoked for the processing of the message
3. If server decides to handle the message on its own some default server action according to protocol specification will be taken
4. If interaction with service logic needed the server will identify the service that must be called for the specific user
5. Then server passes the entire message, its part or just only the SIP method name up to the service logic for processing
6. The service logic process the message based on service configuration and other user related information available from some local or remote database
7. The service logic returns the set of instructions back to the server
8. Server proceeds with the processing of the message and performs the actions specified by service logic or combines that information with its own processing rules.

The SIP Server will always take some action independently from the program controlling the service like inserting the server identity in the Via or Record-Route message header and setting Max-Forward header to limit the number of hops of the message. Example services that could be programmed with SIP include:

- redirecting Calls to the voice / video mail boxes or to the web pages
- replacing Call Waiting Notifications by Instant Messaging
- use streaming media servers for recording and playing the content
- high quality sound over the Internet
- sharing applications
- virtual reality rooms

4.2.4 Architectural issues

Now the challenge is how to effectively control the execution of the services. This address the following issues:

- where the service code should be executed
- when the service should be executed
- how the service logic interfaces with the SIP Server and the SIP protocol stack
- which information about SIP message should be provided to the service logic
- what is the level of control the service logic has over the SIP Server
- what are the restrictions on the resources available to the service logic
4.3 Common Gateway Interface for SIP

For many years the Common Gateway Interface (CGI) has served in the World Wide Web as popular means towards programming web services. Due to the similarities between the Session Initiation Protocol (SIP) and the Hyper Text Transfer Protocol (HTTP), CGI is a good candidate for service creation in a SIP environment. SIP inherits from HTTP its client-server interaction and much of its syntax and semantics and programmers already familiar with CGI and web service creation can use it as a starting point for developing SIP based services.

SIP CGI script executes on the top of the SIP protocol stack implemented by the server.

![SIP CGI Protocol Stack](image)

**Figure 16.** SIP CGI Protocol Stack

Most significant characteristics of CGI-based service creation environment include:

- Use of existing scripts from HTTP and Internet Telephony
- Only for trusted service providers
- Access to many network resources
- Event driven server-script interaction
- CGI maintains the state of the transaction only during transaction is active. After the end-to-end connection has been established the server deletes the states associated with that transaction.
- Script requires detailed information on the SIP message being processed by the server and SIP protocol operation
4.3.1 Language characteristics

SIP CGI provides a powerful tool for developing Internet telephony services rapidly. It is based on HTTP CGI and has a textual interface between the CGI script and the SIP Server. SIP CGI targets trusted service providers like network or server administrators. CGI has a number of advantageous properties that makes it attractive as a service creation tool. Those are:

- **Language independence**: CGI works with any language like Perl, C, VisualBasic, TLC and many others that support access to environment variables.
- **Exposes all headers**: CGI exposes the content of all headers in the HTTP request to the CGI application through the environment variables.
- **Creation of responses**: CGI is advantageous in that it can create all parts of a response, including headers, status codes and reason phrases, in addition to message bodies.
- **Component reuse**: There exists a wide range of CGI utilities in the Internet Telephony world that are also easy applicable to SIP CGI and can be used to program the services.
- **Familiar environment**: Many web programmers are familiar with CGI.
- **Extensibility**: Since CGI is an interface and not a language, it becomes easy to extend and reapply to other protocols, such as SIP.

4.3.2 Service Architecture

The SIP Server communicates with the CGI script process via its standard input (STDIN) and standard output (STDOUT) interfaces. It also uses a number of environment variables to pass the data towards the script. When SIP message arrives the server will set environment variables (also called metavariables) containing the following information:

- Information about request headers (like To, From and Contact header).
- Server configuration information and version of CGI it supports
- Information about the body of the message
- User information

In the following figure an external process that interfaces to the SIP Proxy Server executes the CGI script. The script is invoked on reception of a SIP message. The message is passed to the script via STDIN interface and script returns the output instructing SIP Server to perform some actions.

![Figure 17. SIP Server - CGI Script Interaction](image-url)
These are example environment variables used by the server:

REGISTRATIONS
GATEWAY_INTERFACE
REQUEST METHOD
REQUEST_URI
CONTENT_TYPE
CONTENT_LENGTH
RESPONSE_TOKEN
SCRIPT_COOKIE

After setting metavariables the content of the body will be forwarded to the script via STDIN interface and the script will be executed.

In general when a new SIP message arrives by the SIP server the interaction between the server and the script would be as follows:

1. Determine whether interaction with the service logic is required
2. If so, map SIP address to script
3. Set Environment Variables
4. Create new process
5. Send message body to the process via STDIN
6. Execute script
7. Script returns to the server the SIP Action Lines via STDOUT
8. Script generates Token and instructs the server to set a Script Cookie as the current state of new transaction (only by requests and proxied requests).
9. Script sets script triggers to indicates the server when and under which conditions the script can be called again
10. Server terminates the process
11. Server merges SIP message with script output and performs specified actions

The script output is returned to the SIP Server on STDOUT interface and includes Action Lines like the one presented below which instructs the server to replace the contact header:

```
CGI-PROXY-REQUEST sip: john@example.com SIP / 2.0
Contact: sip: server@company.com
```

In this case the server will perform its standard actions then merge output of the script with SIP message by replacing the Contact header with "sip: server@company.com" and then proxy the SIP request to: "sip: john@example.com".

The script can generate multiple Action Lines containing different CGI headers and instructing the server to modify or delete headers in the SIP message, to set the cookies or request the cookie value for specific transaction. The Action Lines can instruct the server to perform the following operations:

- Forward request
- Forward response
- Generate request
- Generate response
- Default operation
- Set or remove the cookie for a specific transaction
- Return transaction cookie
Some of CGI headers and corresponding parameters are:

- **CGI-PROXY-REQUEST** (*next hop address*) SIP / 2.0
- **CGI-REMOVE** (*headers to remove*) SIP / 2.0
- **CGI-FORWARD-RESPONSE** (*opaque token*) SIP / 2.0 (The server stores each received response until transaction is completed and assigns to it an unique identifier which script can use at later time to indicate which response should be sent by the server)
- **CGI-AGAIN** (*Boolean*) SIP / 2.0 (indicates whether it is allowed to call the script again during specific transaction)
- **CGI-SET-COOKIE** (*Token*) SIP / 2.0 (applies only to requests and proxied requests)
- **CGI-REQUEST-TOKEN** (appended to **CGI-PROXY-REQUEST** requests the server to apply the Token when receiving associated response and calling the script next time)

Summarizing, the message processing by the server depends on the output of the script. The level of control CGI script has over the formatting of the message can vary. The simple script can let the server to handle all message headers processing. More complex script can command the server to replace or remove some headers or the script can generate entire SIP message on its own. Due to the concept of simple network architecture the complex script would execute on the edge of the network and the simple one in the core of the network. It is up to network administrator to decide which level of control the CGI script will have over the server during service execution.

### 4.3.3 SIP CGI evaluation

SIP CGI targets experienced and trusted developers. These developers are supposed to know what they are doing, and there are very few restrictions on the resources available to the service logic they design. This logic can use all information available on the messages received by the server. Furthermore, this same logic can exert a great level of control over the behavior of the server. The processing of the responses to the messages a SIP CGI script has instructed a server to send can lead the script to send a new set of instructions to the server for proxying or generating another set of messages. A SIP CGI script therefore exerts a great level of control over which messages are sent by a server and when they are sent. A SIP CGI script also exerts a great level of control over the content of the messages it instructs a SIP server to proxy or generate. By default, the server will follow the rules in the SIP RFC [5]. However, the script can instruct the server to add, delete, or replace specific headers in the message. This applies even to messages that are proxied.

SIP CGI drawbacks:

- Because the CGI script terminates immediately after generating an output (Synchronous interaction with the server) it can not play an active part after transaction is set up. This restrict using SIP CGI for implementing PINT services [Appendix 13], as it requires the CGI script to instruct the SIP Server to issue SIP request independently of the network event.
- Only one script is allowed to run during transaction
- CGI script is resource consuming as each time when a message arrives a new process needs to be created.
- Must run on the same machine as the SIP Server because of use of environment variables
- CGI scripts must be careful not to interfere with authentication. In particular, adding or removing header fields that are below the Authorization header will cause the message to fail authentication at the user agent.
- When a SIP request is encrypted, the headers, which are in the clear, are passed to the server according to this specification. The encrypted portion of the request is passed to the script as a body. Any SIP headers output by the script will be added to the message. However, scripts should be aware that these might be discarded if they also exist within the encrypted portion.
- Some SIP header fields may carry sensitive information, which the server SHOULD NOT pass on to the script unless explicitly configured to do so. For example, if the server protects the script using the Basic authentication scheme, then the client will send an Authorization header field containing a username and password. If the server, rather than the script, validates this information then the password SHOULD NOT be passed on to the script via the HTTP_AUTHORIZATION metavariable.

- Limited state and information sharing between different script invocation even for the same transaction.

SIP CGI advantages:

- CGI script is an executable file and can access many network and system resources like email, web, database and file store. This can be accomplished by using already existing APIs for different languages.
- CGI supports many different languages
- Many Web services developers are familiar with CGI
- To simplify the service creation the CGI libraries could be distributed and installed on the servers
- CGI application can be created and tested in a very convenient way by installing for example Perl compiler and SIP Server supporting CGI on the home PC.
- SIP CGI can be uploaded to the server like CPL using SIP REGISTER message.

An open issue is support for universal access. While users may still be able to access their services when they move within the domain, they cannot access them when they move to a foreign domain. There is also no general framework for interworking with other advanced services architectures. An example could be how CPL script may be used by end users to tailor generic services developed using SIP CGI. This will of course require interactions between SIP CGI and CPL scripts but this topic has not yet been addressed.
4.4 Call Processing Language

4.4.1 Abstract

CPL is XML-based scripting language targeted to end-user service creation. It is protocol independent and can be used with SIP but also with H.323 or MGCP protocols. XML give the CPL an easy and flexible syntax to describe the services in context of different data types and relationships between those data. Users can write scripts in this language or use some graphical tools generating CPL to define theirs own services. Then the script can be installed on the server by simply uploading it from the user’s terminal like mobile phone or PC. Because of the fact that untrusted party creates the script and the script runs on the service provider's platform the script will be restricted from containing elements that would harm or damage the service execution environment. No external programs can be called from within the CPL code and there is no possibility to call a function recursively. This will ensure the system administrator that the script will execute in a safe manner without unauthorized modifying other user's data or accessing some files or resources on the server.

The protocol stack for the CPL enabled SIP Server is presented below. Look at 4.4.3 on how to create and upload a CPL script to the server.

![CPL Protocol Stack](image)

Figure 18. CPL Protocol Stack

4.4.2 Language characteristics

Generally CPL script consists of two types of information:

- Ancillary Information. It is a description on the script the server may use to process a script correctly. Because it does not specify any operations or decisions itself, it is an optional component and is actually not implemented yet.
- Call Processing Actions. This information directly describes the behavior of the SIP Server on the session set-up event.
• Top-level Actions (Triggered by the signaling events)
  - Outgoing Action (Script owner is the calling party)
  - Incoming Action (Script owner is party being called)
• Subactions (Actions being called from other actions with restriction to be called recursively)

The CPL script distinguishes between incoming and outgoing top-level messages that describe the action to be taken depending on who, the destination or respectively session originating party own the script.

![Figure 19. CPL Language Construction](image)

CPL script is assigned "application / xml " MIME media type.

**Script example:** call redirection to sip:smith@phone.example.com

```xml
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">
<cpl>
   <incoming>
      <location url="sip:smith@phone.example.com">
         <redirect />
      </location>
   </incoming>
</cpl>
```

**CPL language primitives (node types)**

• Switches. These are choices the script can make based on attributes of call request or if current state matches some CPL conditions.
• Location modifiers. Used to add, remove or specify user locations to be contacted. The location list, which can be specified directly or referenced indirectly, can be contained in an external database. Every Location Node will perform actions on this database.
• Signaling operations. These are proxy, redirect and reject actions. Proxy forwards the request to all registered addresses associated with a specific user and specified in the database. Redirect sends the message to the SIP User Agent containing current set of user's locations.
• Non-signaling operations. These actions generate user notifications and handle event logging. An alternative to logging incoming calls is to send an e-mail or instant message to the users in case they are not able to accept the call.
4.4.3 Service execution

CPL script can be executed by a CPL Server that resides on a separate network node than the SIP Server. The SIP Server could simply forward the SIP message to the CPL Server that after processing could return the control messages back to the SIP Server using for example SOAP protocol. Another approach is to have CPL Server co-located with the SIP Server and to use an internal API for interaction. In stand-alone approach server can forward the message to the CPL server to decide which script to be run or can make this decision itself sending the name of the script towards the CPL Server.

The following figure presents one of possible solutions for stand-alone approach where SIP Server forwards the SIP message directly to the CPL Server, which performs necessary processing, and returns instructions back to the server using SOAP protocol. The CPL Server retrieves the user's CPL script from the SIP Location Server then verifies it and executes it. This results in a set of instructions sent back to the SIP Proxy Server.

![Figure 20. Retrieval and execution of the CPL script](image)

The processing of the script boils down to performing some well-defined operation and taking decisions according to the intermediate processing results as defined by the script itself that represents a decision graph. This idea is depicted in the figure below.

![Figure 21. CPL decision graph](image)
Transition decisions are based on:

- registration information (caller information and preferences)
- request information (destination and caller addresses)
- media description
- external information

As shown in the previous figure CPL script introduces (imposes) service creation through describing them as decision graphs. Every node in the graph represents the action or decision to be taken when specific conditions are matched during interpreting CPL script. There can be multiple outputs generated by the single node depending on the current state of script execution. The lowest level nodes produce no output and the default action specific for the current node or the protocol specific action will be taken dependent on the processing state. Then the script terminates. The graph is directed and acyclic what means there is only one path from the root node to every other node in the graph and no loops are possible. The check of the script during submission would be accomplish by constructing a graph object model corresponding to the script and searching for cycles within it. The depth of the graph would give also good estimation for the maximum execution time of the script. The syntax and semantic of the script can be verified by comparing it against an XML specification captured in the so-called Document Type Definition (DTD). Additionally XML has built-in mechanism for extending the set of supported XML Tags and Attributes what makes it very attractive for new services creation. New XML language elements can be declared in the header of the document and XML parser can easily determine whether it can support in the script requested features.

Every node represents a primitive of the language, action to be taken or decision to be made. The CPL parser walks through the graph assembling and combining the actions to provide execution of the service. As already stated all nodes except the lowest level generates output represented in the figure below as transition between the nodes.

### 4.4.4 CPL Script Creation

The script can be created using the following methods:

- End-user creation
  - Manually
  - Using a web application to generate the CPL script automatically
- Third party script provider
- Server administrator script

Then the script associated with a specific user (actually with his address) can be uploaded to the server, verified and registered using several methods. It can be done using HTTP upload or directly in the SIP message by issuing a SIP REGISTER message with the CPL script contained in the body of the message. Because the script can contain sensitive information like end-user's username, password or telephone number the encryption and authentication should be used to transport the script securely to the server. The server administrator can also decide to allow a compilation of the frequently executed script or its part in order to accelerate the service execution. The CPL script can support only the basic operations and provide a basic set of functionality. The script location on the network can vary dependent on the system administrator policies. It can be stored in the database directly on the SIP Proxy or Registrar Server or on the SIP Location Server.
[10] specifies that example database would contain an entry for every user and include the list of user's currently registered locations, user's scripts and information on subscribed services and condition under which script can be called. The database can also specify other information like user e-mail and IP address, he's preferences and terminal capabilities.

### 4.4.5 CPL Evaluation

CPL is based on XML markup language. XML provides easy compatibility and easy integration with the well-known Internet technology. This is very practical, simple and extensible programming language successfully used during the last years for programming of the Internet Telephony services. CPL can be also used for implementing H323 based services but is supposed mainly to be used in the SIP environment.

CPL targets directly the end-user and seems to be very attractive for the wide community of simple users that can define theirs own services without any extensive knowledge. This is true because the CPL scripts can be generated automatically. CPL script provides the easiest way to create and customize the services.

One other crucial advantage is its support for mobility what plays very important role in the VHE. The script that could be stored in the end-terminal and could be uploaded to the network server anytime the user switches on the SIP-enabled device and independently of the location or network type all around the word. The only condition is that the network supports running CPL scripts. There is possible to combine Network Operator's script with those provided from end-user so there are different levels of control possible during the execution of the script.

CPL follows the IN service model and is strictly formalized. It uses decision graphs technique to describe the execution of the service. This is a valuable feature. It lets to analyze the worst case scenarios and to estimate the service execution time by analyzing the script. This check should also guarantee the termination of a CPL script.

CPL can be stored on the user's terminal and uploaded to the server in the SIP network where the user is currently located independent of the place on Earth.

On the other hand it is quite difficult to integrate CPL services with other software components, it does not provide PSTN user interaction features (like play announcement, collect digits) and is hardly extensible. The disadvantageous is also limitation of CPL to only simple service creation and it seems that CPL may not be well scalable in the network with thousands different scripts belonging to all subscribers.

An open issue is how CPL script, transported in the body of the SIP massage, could be used to influence the request handling dispositions on the SIP Servers in the network.

Summarizing it can be stated that CPL can be used for service creation purposes but actually is much better suited for service customization. It is still well defined how the interaction of the scripts should be handled when the caller and callee scripts reside on the same server. Also the interaction between the client and server administrator script should be clarified.
4.5 SIP Servlets API

4.5.1 Introduction

The SIP Servlet API defines a mechanism, which enables the encapsulation of service logic in so-called SIP Servlets on the top of the protocol stack of a SIP Server. The SIP Servlets are generic Java server extensions in the form of Java classes that run inside the Java Virtual Machine (JVM) and are handled by separate threads within the server process. The following figure depicts again the SIP protocol stack and location of SIP Servlet.

![SIP Servlets protocol stack](image)

**Figure 22.** SIP Servlets protocol stack

The SIP Servlet API is similar in nature to the HTTP Servlet API, but there are some important differences. HTTP servlets run only on the origin servers where the HTML content must be generated dynamically. In SIP network proxy servers play a significant role and must support more complex service execution or SIP messages routing and have more complicated state model. As such, SIP servlets must be able to run on all SIP UA, Proxy, Redirect or Registrar Servers. Because SIP Servlets are written in Java they are portable across the operating systems and different types of application servers. They are structured into generic part (javax.servlet) and a SIP specific part (javax.servlet.sip). The generic part defines the general relationship between the Servlet Engine (SIP Server) and servlets and SIP specific part that implements the relation between the Servlet Engine and the SIP protocol stack. This structure is depicted in figure below.
Some characteristics of the SIP Servlets API based architecture:

- SIP Servlets API provides the access to all parts of the SIP message
- SIP Servlets can implement services shared by many users or just a single user
- SIP Servlet can play an active part beyond the session setup (to play announcement for example). Servlets can register themselves as listeners on a transaction via the `SipTransaction` interface.
- SIP Servlet stores Transaction State in a database or in a single servlet class
- Single SIP Servlet can handle messages belonging to multiple transactions
- Messages can share information by setting attributes on transaction object
- SIP Servlets have an easy access to a wide variety of existing APIs, e.g. directories, databases, CORBA, the Java Media Framework, etc. and they can reuse the Java security infrastructure, e.g. the sandbox model for safely executing untrusted code.
- Servlets can be given (controlled) access to the contact database maintained by a proxy or registrar server (via `ContactDatabase` interface)

### 4.5.2 Design Goals

The SIP Servlet API version 1.0 specification must address the following requirements:

- Allow network servers to respond to SIP requests, proxy SIP requests, and initiate new SIP requests the delivery of SIP related services.
- Provide session management support, allowing users to deposit and retrieve data from objects, which potentially span multiple SIP requests, calls, and even multiple protocols.
- Provide high level access to SIP objects, such as requests and responses, with the ability to manipulate key headers and field values. Emphasis is on simplicity and minimality rather than completeness.
- Must hide the complexities of SIP wherever possible; developers should not need to be SIP experts
- Definition of rule based mappings from SIP requests to servlets which will process them
- Definition of a security model
- Definition of an XML DTD for SIP Servlet deployment descriptors
- Definition of a jar based file format for SIP Servlet applications (similar to the Web Archive (war) format defined for the HTTP Servlet API)
- Access to location databases
4.5.3 Overview

The SIP Servlet API sufficiently defines the programming interface between the SIP Server and a single SIP Servlet. The API presents to the application writers an abstracted view of SIP protocol stack by encapsulating SIP messages into objects and interfaces. In SIP Servlet execution environment the service is defined as a collection of interacting SIP Servlets, filters or other resources and associated with them configuration data. Before going into details on service execution a briefly glossary of SIP Servlets API is presented.

**Servlet Engine / Servlet Container**
A process within a SIP Server that interacts with the servlets via SIP Servlet API.

**SIP Servlet**
Java code for encapsulation of service logic. It implements the SipServlet interface.

**Primary Servlet / Root SIP Servlet**
Shared object via which servlets communicate with each other. This servlet is called as first by the Servlets Container on reception of a SIP message.

**Message Dispatcher**
An interface that intercepts all servlet invocations implemented by the Primary Servlet. Allows the PrimaryServlet to regain the control over the message between the servlet invocations.

**Servlets Descriptor**
Describes, for a given servlet, the symbolic names of the servlets that can be specified by this servlet as result of the message processing. It specifies also the expected functionality and desired lifetime policy of those servlets. In the deployment phase those symbolic names are mapped onto the actual servlet classes. The XML-based descriptor format can be used.

**Servlet Lifetime Policies**
The SIP Servlet API leaves the decision when to destroy a servlet entirely to the Servlet Container (SIP Server). This standard policy causes problems for servlets, which communicate with the outside world with other means than the SIP Servlet API.

**Service Dispatcher**
Has the task to identify the right service for a given SIP Message and give it back to the Servlet Container in the form of a symbolic name representing the Primary Servlets of this service.

**Service Descriptor**
The configuration data of a SIP Service. Again the XML-based descriptor format can be used. Service Descriptor contains:

- Descriptors of the servlets of the service
- Descriptors of the custom filters of the service
- Filters to be applied for every servlet
- Mapping of the symbolic names onto servlets classes
- Service Primary Servlet
- Indication flag whether the service is distributable or not
- Optional initialization parameters
- Declaration of a custom contact database implementation

**Service**
Collection of interacting SIP Servlets.
Servlet Interaction
Processing of the incoming SIP message or response by multiple servlets.

Filters
Filters are objects that intercept invocations of a servlet and preprocess the SIP Message object that shall be passed to the servlet. They can inspect and modify the message, or they can even replace or wrap the message object with a custom implementation. The SIP messages are forwarded to the servlet as objects in a vendor-specific way. The filters allow to custom the interface between the server and the servlet and are applied before the servlet is being called. They are not applied to the transaction but to the single message that arrives by the SIP Server. The single servlet can be configured to be filtered by arbitrary number of filters. Filters are implemented as objects and can be applied to the following cases:
- Authentication and authorization
- Call logging
- Firing certain event on a general events channel

To make it possible that every Servlet Container can use same filters, it is necessary to define the org.ietf.sip.Filter interface package in the SIP Servlet API. The filter developer creates a filter by implementing this interface.

Filtering
The customization of the request and response objects defined by the API.

Servlet Mapping
The mapping of the symbolic name specified by a servlet onto servlet instance.

Contact Database
An Interface to access the SIP Server vendor's database. LDAP database is an option. The database can be filled with registration informations other than those contained in SIP REGISTER requests. The Contact Database defined by the API is implemented by the Servlet Container vendor, prohibiting to plug-in custom implementations. Filters can be used to customize this interface.

4.5.4 Service Deployment
The SIP Servlets API is going to be the standardized interface for the SIP servers. The benefit of standardizing the servlet API is, of course, that servlets can be packaged as small applications and be deployed on any conformant SIP server or user agent. SIP servlets will typically reside on network servers where they will be responsible for making routing decisions. SIP Services can be packaged as a single compressed file using Java standard archive format (JAR) together with the Service Descriptor. Service Descriptor provides a definition of the service and allows the SIP Server or its administrator to check the service according to some rules based on local administrator polices defined for the server. Servlets should be named using globally unique names. First the servlet code is loaded into the servlet engine's Java Virtual Machine. The code may be loaded from a local file system, a remote Web server or something entirely different. Then the servlet is instantiated and initiated. The Servlet Engine instantiates the servlet class using the default (empty) constructor, casts the resulting object to a SipServlet and passes the servlet its configuration: public void init(ServletConfig config). The ServletConfig provides access to the servlets view of the servlet engine, a ServletContext object. Different servlets may have different views of the servlet engine, i.e. different ServletContexts.
The servlet is then repeatedly invoked to handle SIP messages. The method invoked depends on whether the message is a request or a response:

```java
public boolean gotRequest(SipRequest req)
pubie boolean gotResponse(SipResponse res)
```

When a server decides to terminate a servlet, e.g. because the server is being shut down or a new version of the servlet is being loaded, the servlets destroy method is invoked: public void destroy () and the server destroys any reference it holds to it, thus making the servlet instance to get garbage collected. The servlet code would typically be loaded only once, although some servlet engines may support a class reloading mechanism for ease of development.

### 4.5.5 Service execution

When a SIP message arrives the SIP server implementing a SIP stack will process the message and cast it to the Servlet Container which will check which service is required to run by invoking the Primary Servlet. Every user can have a Primary Servlet installed on the server but the incoming message could also be mapped on the specific users group. Identifying the user or users group could be based on the association of the message headers and available data resources. A simple lookup table would provide information on to which user group corresponding address belongs expressed by some symbolic name of the Primary Servlet to be executed. Now the filtering on the SIP message can be performed proceeded by the invocation of the Primary Servlet. Primary Servlet returns the symbolic names of the servlets for the given service and interaction pattern between those servlets. The Servlets Container forwards the SIP Message to those target servlets. The call of any servlet can be preceded by applying the filters. Any servlet can return a sequence of the symbolic names to the Servlets Container. The servlet does not know the next servlet. It only generates a symbolic name that is mapped onto the servlet instance. After invocation of every servlet the control goes back again to the Primary Servlet which can apply some postprocessing or directly forward the message to another servlet.

![SIP Servlet API service architecture](image)

**Figure 24.** SIP Servlet API service architecture
Here follows the service execution description in a sequential order.

1. SIP Server implementing a SIP stack receives the message and after processing it gives it to the Servlet Container
2. Now it has to be checked, which service is required by calling `getServiceName(SipRequest request)` of the Service Dispatcher
3. The Service Dispatcher checks the Require SIP message header whether the service is there specified. If not it will use the previously described rules
4. Service Dispatcher returns a symbolic name of the servlet name of the chosen service, the Primary Servlet
5. Now `doFilter(SipMessage message, FilterChain chain)` can be called to apply filtering on the message
6. Servlet Containers Wrap the message now to the Primary Servlet for processing by calling `gotRequest(SipRequest requ)` or `gotResponse(SipResponse resp)` via the Message Dispatcher.
7. Primary Servlet process the message and returns via Message Dispatcher the list of symbolic names for recognized service and interaction pattern between those servlets.
8. Message Dispatcher resolves the appropriate servlet instance by using the mapping of symbolic names to servlets provided by the configuration data. The configuration data might map the name "alice-customer-servlet" to a servlet of class `my.org.AliceCustServlet`.
9. The filters for the first servlet can be applied and the message is cast to the target servlet
10. The servlet process the message and return the control to the Primary Servlet and can specify a new list of symbolic name of the servlets that should be invoked
11. The Primary Servlet can apply some postprocessing or directly forward the message to another servlet
12. If the previous servlet returned symbolic names of other servlets those are invoked before moving to the next step
13. The processing of the message ends. This can result for example in sending an SIP request by applying the `sendRequest(SipRequest requ)` method on `SipResponse` interface

### 4.5.6 Evaluation

The SIP Servlets API standardized within IETF is based on the HTTP Servlets API known from the Internet web-application programming environment. The services can be composed by using a set of building blocks, the so-called servlets, and the SIP Servlet API that allows interaction between those components and the SIP server. SIP servlets implement service logic extending the generic SIP server functionality in the form of Java classes that run inside the Java Virtual Machine (JVM) and are handled by separate threads within the server process. This provides a powerful service execution environment that can handle multiple services simultaneously. The execution of a single service can involve multiple servlets to run. Servlets interaction provides a detailed level of control over the service execution. The SIP Servlet API allows SIP servlets to manipulate SIP requests and responses. The SIP Servlet API also supports to load and initiate new servlets that enhance the functionality of SIP for implementing and running new services. New servlets can be uploaded dynamically to the server while the service is being requested. Because SIP Servlets are written in Java they are portable across the operating systems and different types of application servers.
Another valuable feature is that services based on SIP servlets can run in a distributed environment. This means that the SIP Server can reside physically on another network node than the SIP servlets. Other advantages include extensibility, a standardized framework for adding new servlets and easy integration with the Internet technology via existing Java servlets APIs like Java Database Connectivity (JDBC) and Remote Method Invocation (RMI).

Another key benefit of using SIP Servlets API for service creation is that the end user can use them to program the behavior of their own terminal [18]. Collocation with the traditional HTTP engine provides an integration of Web and telecommunication services. This common HTTP and SIP servlet engine could be enhanced with a programming environment on top of it to allow the user to modify or write new servlets.

Because of the similarities of HTTP Proxy and SIP Redirect and UA Servers with respect to the request-response communication model it is easy to implement SIP Servlets API for Redirect and UA Servers by mean of reusing existing HTTP Servlets API and extending with SIP specific features [18]. The different functionality of the SIP Proxy Server would require different kind of API for forwarding requests and another API for interaction between those two generic APIs. The advantage of this solution to design two different SIP Servlets APIs is the significant simplification of design process by reusing existing servlets and HTTP Servlets API. The single API approach has the benefit of having one implementation model for all SIP servers in the network, however some functionality like request forwarding would be never used in the Redirect and UA Servers.

The main disadvantage of the SIP Servlet API is that servlets can be only implemented in Java and must run on a Java SIP Server. This implies that this programming method targets experienced Java developers only. The simple, however, may take advantage of some programming platform that would simplify the service creation. It is expected that combining the SIP Servlet API with JavaBeans may lead to a new service creation platform that will solve this problem.

The SIP Servlet API is the best candidate to be used for service creation on a SIP platform.
5. Parlay / OSA Service Creation Environment

5.1 Summary

Previous section presented the methods for implementing of multimedia services using SIP protocol designed by the Internet experts within IETF. At the same time as SIP was designed other consortia as Parlay and JAIN developed APIs to provide the access to the Core Network functionality. It was widely believed that applications will be the key to the success of future communications networks and that providing an infrastructure where applications can be easily created is extremely important. That is why APIs began to play a significant role in the concept of provisioning end-to-end multimedia services. Most of APIs developed by Parlay has been adopted by 3GPP, the standardization body responsible for the UMTS standards. The APIs has been created with the goal of defining a technology independent access to the network functions and can be used by third party application as well. In this section we will give an overview of the API-based network architecture, list already existing APIs for different services and briefly explain how the API-based services can be implemented.

5.2 Introduction

The new generation network concept that will allow applications to access the Core Network functionality is called the Open Service Architecture. It consists of the Service Network, where the applications reside, and the collection of the APIs that are implemented on the so-called Service Capability Servers (SCS). The already existing and standardized APIs are specified for a number of distributed technologies like CORBA and DCOM. SOAP API is planned to be developed in the near future. Those open technologies are used within the Service Network to allow the applications to communicate with the network using the Common Object Model. Applications will be able to get the access the all, so far hidden, information and capabilities of the network, by contacting the SCS located in the intermediate network layer, the Service Enabler Layer. The developers can use APIs to design a variety of services for 2G, 2.5G and 3G networks including multimedia services. A Parlay application needs to implement the API-based interface to be able to request services from the SCS. The current set of available is listed in the next section. The following figure depicts the architectural view of the service provisioning system.

![Parlay / OSA System Architecture](image)

Figure 25. Parlay / OSA System Architecture
5.3 Parlay communication principles

The Parlay / OSA APIs can provide the applications to access the Core Network functionality provided by switches, Home Location Registers (HLRs), GPRS Supports Nodes (GSNs), billing servers etc. The SCS will implement a standardized Parlay / OSA interface in order to interact with the application and network specific interface will map the communication details onto core network protocol like MAP, CAP or CAMEL. This situation is depicted in the figure below.

![Parlay call sequence Diagram](image)

Figure 26. Service Capability Server View (Stand-Alone SCS)

As already noticed, the applications communicates with the Parlay Gateway using Common Object Request Broker Architecture (CORBA) technology and the Parlay API. The example sequence diagram for communication between the Application Server and SCS is presented below.

![Parlay call sequence Diagram](image)

Figure 27. Parlay call sequence Diagram
At this moment the most Parlay APIs are specified in CORBA IDL. However, application developers are free to use any language of their choice to program the services, because CORBA allows the applications to communicate with one another no matter where they reside on the network and independent of which programming language they are written in. CORBA specifies a protocol called the Internet Inter-Orb Protocol (IIOP) that provides the functionality compared to Transaction Capabilities Application Part (TCAP). The Parlay Protocol Stack is depicted below.

![Parlay Protocol Stack](image)

**Figure 28.** Parlay Protocol Stack

### 5.4 Service development environment

Specific Software Development Kits can be developed and provided for easy, rapid and cost effective implementation of Parlay Client Applications and solutions. To make it even easier development toolkits may include a specific development environment platform to shield application developers from communications technologies like CORBA and APIs. The one choice would be CORBA application development to Parlay API and the other possibility would be, depending on the skills of the developer and the complexity of the application, a number of other programming interfaces including:

- Development of pure Java applications masking CORBA communication
- Component application development using an Integrated Development Environment based on Java Beans
- Simple eXtensible Markup Language XML script based application development
SDKs could be provided with a set of development components (Java beans), included in the development toolkits. The programmers can combine those software components as they see fit in the application and will be able to extend the existing set of beans with new ones as soon as new APIs come available. The Software Development Kit could also include SCS, Framework and Test Application simulators to give the developers ability to test their application before they can be physically deployed on the Application Server.

Using Parlay / OSA APIs developers are free to combine all existing APIs and integrate them in a single or distributed Parlay Client Application to provide desired functionality. Establishment of the simple Call will require for example the application to use User Status / Location API to locate the user then User Interaction API to collect the phone number digits and finally Call Control API to set up connection between the parties.

Extensibility of APIs functionality and flexibility of developing and testing Parlay-based applications gives developers the great opportunity to develop the services at low expenses and in extremely short time-to-market time.

5.5 Parlay API capabilities

The current set of APIs consists of two categories of interfaces, that both are APIs itself:

- Service Capability Server Interfaces. These offer applications access to a range of network capabilities and information.
- Framework Interfaces. These provide the infrastructure capabilities like authentication, SCS discovery and registration, fault management, etc.

The following APIs (also called Service Capability Features - SCF) has been developed by Parlay consortium so far and are available for application developers:

- General: contains the introduction and methodology used
- Common data: generic data, used in other parts.
- Framework: defining the infrastructure capabilities like authentication, SCF discovery, SCF registration, fault management, etc.
- Call Control, defining the call control family with capabilities ranging from setting up basic calls to conference calls.
- User Interaction: SCF to obtain information (digits) from the user, place announcements, send short text messages.
- User location / User status: SCF to obtain location and status information.
- Terminal capabilities: SCF to obtain the capabilities of an end-user terminal.
- Data session control: SCF to influence data sessions.
- Messaging: SCF for access to mailboxes
- Connectivity Management: SCF for mainly provisioned QoS.
- Account Management: SCF to access end-user accounts
- Charging: SCF to charge end-users for use of applications / data.
The Call Control API offers the most important capabilities and allows Parlay applications establishing of the multimedia sessions. The call control network service consist of three interface classes:

- Call manager. Management function for call-related issues
- Call. Methods to control a call
- Call Leg. Methods to control individual legs in a call

The programmers can use Call Manager and Call level interfaces to implement the following function:

- Initial trigger setting and event notification
- Basic call handling. That is, re-route call, start up call, perform screening to allow or disallow a call to continue, release call, deassign a call (removing the relation between the call and the application but leaving the call intact).
- Event monitoring. Information on network events.
- Setting of charge plan
- Information about the call. For example, call duration and the release cause.
- Call duration and charging supervision

The Leg level interface provides the following call control features:

- Create new call legs and call routing
- Attach leg to the call
- Detach leg from the call (put on hold)
- Release the call leg
- Monitor events
- Request information associated with the call leg

5.6 Standardization body

The early Parlay / OSA APIs inherited from Parlay APIs were developed jointly by 3GPP and ETSI because of the overlap in work. Latter, also Parlay has joined the community developing APIs for the third generation networks. Now the work is done jointly by the joint API standardization working group including 3GPP, ETSI and Parlay. There is also informal collaboration with JAIN to achieve as big as possible Developers Community. Currently 3GPP is also the body within the ETSI.

5.7 References

All interface specifications for the OSA APIs listed in the previous sections can be found on the website of European Telecommunication Standardization Institute (ETSI) at www.etsi.org or www.parlay.org.
5.8 Evaluation

The Parlay APIs are evaluated in terms of advantages and disadvantages. One advantage is that the Parlay APIs enable independent software vendors to write applications that provide heterogeneous services. This means that the service providers can use the APIs to implement services across wireless, IP-based and CS-based PSTN networks. This is possible because the APIs provide the sole access to the core network capabilities. The APIs hide the core network type and its signaling protocols.

Another closely related advantage is that the Parlay APIs give the service providers the freedom to implement the service without extensive knowledge of the core network technology that interacts with the Parlay application. The application developers have access to network services such as messaging, call routing and call control and can integrate those with applications and data outside the network.

Another positive aspect of Parlay as service programming platform is the granularity of the Parlay APIs. Application developers can start using a single Parlay API, but can also use all available APIs, combining them into one powerful application. This allows for services that mix voice and data communication.

Another advantage is the programming language independence. The main API specification are defined in the technology independent Unified Modeling Language (UML). This enables using the APIs via technologies such as COM, CORBA or JAVA. Many of commonly used programming languages like C++, SmallTalk and Java can be used for implementation of the services and support for XML and SOAP is coming soon.

A final advantage is that the Parlay APIs can be extended with additional not standardized proprietary APIs.

A disadvantage are the complexity and size of the Parlay / OSA specifications [16, 17]. It requires time from the developer to understand the APIs and use them effectively. Another drawback is the complicated format of Parlay data types and interfaces that may cause complications creating Parlay applications. However, compared to the IN service creation techniques, the service creation complexity and the time it takes to design an application has been drastically decreased. Additionally a Parlay application implementing a multimedia service must be multithreaded. This increases the complexity of the creation and of the testing process. Providing an Integrated Development Environment to simplify the service creation can alleviate this problem. Another solution is to provide SDK tools for rapid service development and testing.

Although this is often listed as advantage, another disadvantage is that Parlay APIs hide the network specific information from the application developers [14]. This decreases radically the control the application can have over the network functions like session establishment and consequently imposes limitation on the service creation. In some cases better results can be achieved if the service programmers can take full advantage of all capabilities of a specific network protocol. One solution to this problem is to define a new interface or to modify the existing one, such that some protocol specific functions are exposed. Applications using this approach of course loose the ability to operate the same over all Core Networks.

Summarizing, the Parlay APIs and their programming tools are mature and provide a great opportunity for programming services, especially over heterogeneous network with different access types. They allow for the combination of different network capabilities that are provided via a variety of APIs. They can be also extended with new APIs and capabilities that are not even standardized. They provide also high level of reliability, extensibility and flexibility. It is very likely that the Parlay APIs will become a standard in the areas of multimedia service creation and provisioning.
6. SIP versus Parlay / OSA Service Creation Environment

6.1 Summary

This chapter summarizes two previous chapters. As service creation possibilities has been examined and there is more clarity about the methods of programming the services there is a time for some comparison. This has been done in the form of a table, which lists in the columns Parlay API, SIP CGI, CPL and SIP Servlets API and in the rows different properties ordered according to the service creation lifecycle. The chapter is enclosed with conclusions about the applicability of presented methods for providing multimedia services in 3G network.

6.2 Comparison tables

The comparison has been made according to the service lifecycle. Every table groups the service creation properties that relate to different stages of the service creation and implementation process.

The service lifecycle has been divided onto the following steps:

- Construction and Testing
- Deployment and Activation
- Utilization and Execution
- Withdrawal
<table>
<thead>
<tr>
<th>Service Lifecycle</th>
<th>Property</th>
<th>Parlay / OSA APIs</th>
<th>SIP - CGI</th>
<th>SIP CPL</th>
<th>SIP Servlets API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction &amp; Testing</td>
<td>Target Developers Community</td>
<td>• Mainly targeted for 3rd party services creation.</td>
<td></td>
<td></td>
<td>Mainly targeted for Network Operators and Enterprise service providers.</td>
</tr>
<tr>
<td></td>
<td>Development Environment</td>
<td>• The applications can be written in any language because the Parlay / OSA interface is specified in a standardized IDL language.</td>
<td>• CGI application can be written in any programming language that supports access to environment variables like Perl, C, C++, VisualBasic, TLC and many others.</td>
<td>• CPL scripts are written in XML scripting language. The parsers that interpret them can be built in any of a number of languages such as C++, C, JavaScript, Tcl, and Python.</td>
<td>SIP Servlets are mainly written in Java and depend on the version of the Java Run-Time Environment.</td>
</tr>
<tr>
<td></td>
<td>Available Toolkits</td>
<td>• SDK tools (e.g. Java Beans or IDE) can be provided and used by a wide Developers Community.</td>
<td>• Existing scripts and libraries from HTTP and Internet Telephony can be used.</td>
<td>• CPL script can be easily created manually or generated automatically. Many XML graphical editors and XML parsers already exist.</td>
<td>Servlets can be created using Java Beans (JDK 1.2). The Servlets may also take advantage of many others existing APIs.</td>
</tr>
<tr>
<td></td>
<td>Complex or Simple Services</td>
<td>• A wide range of complex services can be implemented as the application can access most of the network functionalities provided by different available APIs.</td>
<td>• Many services can be created however CGI script runs only triggered by the network event. This restricts using CGI for implementing services like &quot;Click-To-Dial&quot;.</td>
<td>• As no external programs can be called from within the script and there is no support for recursion, CPL script is limited only to very simple services like call screening, time-of-day routing, call forwarding / redirection and call blocking.</td>
<td>SIP Servlet API allows the SIP Server to invoke different kinds of Servlets to execute the service and therefore a wide range of services can be created.</td>
</tr>
<tr>
<td></td>
<td>Application Testing</td>
<td>• A variety of tools simulating the network environment can be provided with the SDK.</td>
<td>• The CGI scripts can be tested on any server capable of processing CGI scripts before installing it on the Network Operator’s server.</td>
<td>• Most existing CPL Editors come up with extended development environment enabling of testing the script. The users can retrieve and modify the script at any latter time.</td>
<td>Possibly some simulation tools will be provided as soon as the first SIP Servlet API comes available.</td>
</tr>
</tbody>
</table>

1 Evaluation unit
## Deployment & Activation

<table>
<thead>
<tr>
<th>Service Lifecycle</th>
<th>Property</th>
<th>Parlay / OSA APIs</th>
<th>SIP - CGI</th>
<th>CPL</th>
<th>SIP Servlets API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Verification</td>
<td></td>
<td>The Network Operator must approve the application before it is being installed on the AS.</td>
<td>Server administrators must perform equivalent levels of security review before installing CGI scripts. If script is calling external programs the server should ensure that untrusted code could not be executed.</td>
<td>CPL script represents an acyclic graph and the server can create a graph object model that can be compared with the graph represented by the script and check the script for safety and estimate the service execution time. The syntax and semantic of the script can be verified by comparing it against an XML specification captured in the so-called Document Type Definition (DTD). Every script submitted to a CPL server should comply with this DTD that describes the XML syntax.</td>
<td>The server administrator can manually inspect the Servlet code or apply some code signing techniques to achieve an appropriate level of trust. Better approach would be to automatically analyze the Java bytecode to enforce restrictions on SIP Servlets that may contain loops or allocate to big amount of memory. This check could be performed during the upload or installation time.</td>
</tr>
<tr>
<td>Installation &amp; Configuration</td>
<td></td>
<td>The installation of an OSA application is accomplished by signing an agreement between the application provider and the owner of the OSA AS which will typically be the Network Operator. This results in creating a SLA that is used for activating and registering of the service in the network, which is done via the Framework.</td>
<td>Uploaded and configured by the Network Administrator. The SIP REGISTER method can be used for this purpose, but also other protocols as LDAP, ACAP, SNMP or HTTP file upload can be used.</td>
<td>As the scripts will be typically created and stored on the user's end devices they can be easily uploaded to the server during the registration time by using the SIP REGISTER method. The user can install the script on any server in the homes or visited network however, the method by which scripts are transmitted from a client to a server MUST be strongly authenticated.</td>
<td>SIP Servlets can be distributed as all Java classes as single compressed JAR files together with the Service Descriptor (XML DTD document) containing Servlet configuration information. (Servlet Descriptor). While the configuration of a servlet is specified, it is not yet well defined how this configuration is loaded into the servlets container.</td>
</tr>
<tr>
<td>Business Domain</td>
<td>An OSA application can reside on the Network Operator or 3rd party OSA Application Server.</td>
<td>CGI scripts reside on the servers owned by the Network Operator.</td>
<td>The CPL script is designed to be implemented on either network servers or user agent servers (UAS) which reside on the user's end-devices.</td>
<td></td>
<td>SIP Servlets are typically installed and execute in the network but can also be used on the user's terminal to implement some lightweight services.</td>
</tr>
<tr>
<td>Interface Extensibility</td>
<td>OSA architecture upon which the Parlay interface is built allows extension of the interface with a new standardized or not standardized proprietary APIs.</td>
<td>CGI is an interface and not a language and this makes it easy to extend and reapply to new protocols and technologies.</td>
<td>XML allows developers to create their own Document Type Definition (DTD) and effectively create new tag sets that can be used for multiple applications. XML itself is also being extended with several additional standards that add styles (e.g. XSL), linking, and referencing ability to the core XML set of capabilities. However CPL is limited in functionality because it must be safe for execution and is hardly extensible.</td>
<td>XML allows developers to create their own Document Type Definition (DTD) and effectively create new tag sets that can be used for multiple applications. XML itself is also being extended with several additional standards that add styles (e.g. XSL), linking, and referencing ability to the core XML set of capabilities. However CPL is limited in functionality because it must be safe for execution and is hardly extensible.</td>
<td>SIP Servlets API is build up from methods and interfaces that allows transparent mapping of the SIP messages from and to the server. Introduction of new SIP method or response would require only adding a new String constant to the existing set.</td>
</tr>
<tr>
<td>Application Portability</td>
<td>Because the OSA / Parlay APIs are a standardized interfaces the OSA application is easily portable between different networks.</td>
<td>CGI is a standardized Internet interface supported by many Web Servers. The Network Operator must provide CGI support on the network servers in order to support running of CGI scripts.</td>
<td>CPL script can be stored on the user's device and can be uploaded to any local server with CPL capabilities. Despite growing popularity of XML it can not be guaranteed that all networks servers will support CPL.</td>
<td></td>
<td>SIP Servlets API is to be a standardized interface portable between different networks and servers.</td>
</tr>
</tbody>
</table>
## Execution

**Utilization & Execution**

<table>
<thead>
<tr>
<th>Service Lifecycle</th>
<th>Property</th>
<th>Parlay / OSA APIs</th>
<th>SIP - CGI</th>
<th>SIP</th>
<th>SIP Servlets API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Level over Multimedia</td>
<td>OSA / Parlay application can have the control over the multimedia session during entire session duration time.</td>
<td></td>
<td>CGI script can be executed anytime the server receives a SIP message but the script maintains the state of the session only during session establishment. After the call has been setup the server deletes the states associated with that call and is not involved in the session anymore.</td>
<td>As with CGI the CPL script can be only used to implement the services that have impact on the session only during session establishment time.</td>
<td>Despite of other methods any servlet can subscribe itself to the network as Listener and can be notified about all events associated with a specific session. This way the Servlet can control the session during entire session duration time.</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance Aspects of Application</td>
<td>A Parlay / OSA application can be compiled as executable file and run very efficiently.</td>
<td></td>
<td>CGI application can be a script or an executable file and is optimized for that specific language. However CGI application does not run continuously and each time a SIP message arrives by the server a new process needs to be started. This makes CGI application less efficient and more resource consuming.</td>
<td>CPL script performance depends on the efficiency of the CPL Parser that interprets the script. Since a parser can be an executable file the CPL script can be run very efficiently. Most existing CPL servers are fully multi-threaded applications.</td>
<td>SIP Messages are passed to a class that runs within a JVM inside the SIP server and the same servlet can be simultaneously used by multiple services. As all Java codes the servlets can also be compiled for better performance.</td>
</tr>
<tr>
<td>Run-Time Environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Centric or Terminal</td>
<td>Parlay / OSA application can run on the network server or on the user end-devices.</td>
<td></td>
<td>The CGI script runs on the network server, which controls the execution of the service in a network centric manner.</td>
<td>CPL script can reside and execute on the network server or on the user's terminal thus both possibilities apply. One session establishment can require multiple different scripts distributed over the network to be executed but those constitute different services.</td>
<td>SIP Servlet API-based services can be executed in a network-centric or terminal-centric way. Normally one service will require interaction between multiple servlets.</td>
</tr>
<tr>
<td>Centric Service</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td>The Parlay / OSA application may access all common Internet resources as any web application does, but often the full access to those resources my be a subject to SLA and may depend on the network characteristics and configuration.</td>
<td></td>
<td>CGI is a normal program and can access many Internet and system resources like email, web, database and file store. This can be accomplished by using already existing APIs for different languages.</td>
<td>CPL allows easily services integration with web, email and chat services.</td>
<td></td>
</tr>
<tr>
<td>with Internet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recourses</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications Interaction</td>
<td>OSA architecture allows for some services to invoke multiple applications distributed over the network.</td>
<td></td>
<td>Only one script can run at time so no interaction is possible.</td>
<td>Script-to-script interaction is possible when a server invokes multiple scripts for a single call. This can happen when both the call originator and the call destination have scripts stored on a single server, when a script forwards a request to another address which also has a script or when an administrative script is specified as well as a user's individual script.</td>
<td>SIP Servlet API allows multiple servlets to process one incoming SIP message even if the servlets are distributed over the network. The Service Descriptor indicates whether the service is distributable or not.</td>
</tr>
</tbody>
</table>

Master Thesis
OSA / Parlay on a SIP Network

Ericsson Eurolabs, The Netherlands

Ericsson TU/e
## Service Property Parlay

### OSA APis Lifecycle

The service deinstallation is a Service Withdrawal Lifetime subject to SLA between the (Removal) Policy owner of the AS and the service provider. The Network Operator decides when to remove the CGI script. CPL-based service lifetime can be controlled by the End-User. Service Lifetime can be controlled by the Network Operator or can be subject to SLA if the 3rd parties provide the service.

<table>
<thead>
<tr>
<th>Service Lifecycle</th>
<th>Property</th>
<th>Parlay / OSA APIs</th>
<th>SIP - CGI</th>
<th>CPL</th>
<th>SIP Servlets API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawal (Removal)</td>
<td>Service Lifetime Policy</td>
<td>★ The service deinstallation is a subject to SLA between the owner of the AS and the service provider</td>
<td>★ The Network Operator decides when to remove the CGI script.</td>
<td>★ CPL-based service lifetime can be controlled by the End-User.</td>
<td>★ Service Lifetime can be controlled by the Network Operator or can be subject to SLA if the 3rd parties provide the service.</td>
</tr>
</tbody>
</table>
6.3 Conclusions

SIP has been chosen by the 3GPP as the protocol of choice for 3G networks, and the SCS as the gateway to the core networks could be implemented as SIP-enabled to provide enhanced facilities for service creation. This is still open issue, which of the service creation methods will be used or whether both possibilities can be taken into consideration.

It is very likely that both technologies will be involved in the programming and accessing of the multimedia services in the network. However, much discussion arises about the relationship between SIP and Parlay / OSA APIs on a SIP network, but this is the subject for discussion in the following chapters. As can be seen from the comparison table the Parlay APIs are very competitive as compared to SIP service creation tools and offer an easy development environment. Many telecommunications equipment vendors are very committed to Parlay. It has been designed to be technology independent, and to be implementable in many environments and network types. The main advantage of using Parlay APIs for programming the services is the variety of capabilities offered by various APIs and the possibility for the application to combine those with each other. It is allowed and may be very efficient and powerful to use User Interaction, Mobility or for example Connectivity API along with the Call Control API. Other interesting feature is the interaction between application and the call during the entire lifetime of the call. Other important advantages of Parlay SCE include:

- programming language independence
- existence of Integrated Development Environment
- easy testing and simulation
- extensibility even with propriety APIs and services
- standardized interface
- openness to 3rd party service providers
- easy integration with existing Internet technologies

Among SIP service creation tools there are two methods that are very similar with respect to provided capabilities and development environment. Those are SIP CGI and SIP Servlet API. Both are standardized and primarily targeted to the Network Operators but can also be used by 3rd parties. They are derived from already existing web application environments as HTTP CGI and Web Servlets API. Both technologies have been already for many years very successful in the area of Internet services. It seems that SIP CGI is more advanced technology. It comes with a mature RFC against SIP Servlet API drafts. In both cases the work is still in progress and it is not yet well defined how this tools could also support services provided by 3rd parties. The differences between SIP CGI and SIP Servlet API are that SIP CGI maintains the state of the session only during session establishment. CGI script runs only triggered by the session establishment network event. After the call has been setup the server deletes the states associated with that call. At that time the SIP Servlet API can still play an active role and be involved in the session. Disadvantageous is for SIP CGI that only one script can run at any time while the servlet-based service may engage many servlets that can sequentially produce different outputs combined into well-defined SIP server behavior.

CPL stands alone with its totally different target development community. It is primarily addressed to the common end-user and can be used for implementing very simple but useful services. It has no power to create complex and fancy services because it is restricted with running external programs and including iterative loops. Instead can be very easily verified at the installation or at the run time. It is considered to be safe and can be easily integrated with Internet services but it is hardly extensible. CPL has been widely used in the Internet Telephony area and is now taking a place in the multimedia service creation technology. There already exist CPL-based products on the communication market.
The future network can integrate all those technologies and it is very likely that SIP Servlets API and CPL-based services will be the best candidates along SIP service creation methods. Together with Parlay / OSA APIs they will form a powerful service programming platform for the new telecommunication network. However, for time being, as long as 3G architecture is not yet implemented, only Parlay APIs can provide a service creation and provisioning solutions in the existing 2G and 2.5G networks.
7. Service control in IMS subsystem via SIP

7.1 Summary

This chapter introduces in more details the IP Multimedia Subsystem and presents the work in progress and open issues on the IP MM architecture. The focus in this section will be on the IP MM service execution signaling during the session setup. The research to be made is to check the consistency of service triggering and service execution signaling based on SIP with that provided by OSA service architecture. The main issue is that SIP can be compared to ISUP from ISDN protocol and is totally different from today's session control protocols used for telecom services like CAP for example. It should be investigated whether SCS can provide a suitable mapping onto SIP as well. The input for this work consists of already existing and implemented services for the circuit switched network based on OSA interface. The question is whether SIP can be used together with OSA to control the service execution in the IP MM Core Network to provide the same services as for the circuit switched protocols. The issues to be addressed here and being discussed from here on include:

- Does SIP fulfill the requirements for the service control protocol to provide all required functionality in the IP MM subsystem?
- Is this possible to have SIP carry on the service control signaling from OSA AS to the Core Network and what are the limitations and inconsistencies in cooperation of those two signaling methods?
- How can SIP efficiently be mapped to OSA and vice versa?

The coverage area of the research is limited to the Generic Call Control (GCC) Parlay API. Other APIs like Multiparty Call Control (MPC) or User Interaction are for further study.

7.2 IMS Service Architecture

The figure below depicts an overall 3GPP view of how the "Vertical Service Provisioning" based on OSA interface could be realized in the network using the ISC interface.

![Figure 29. IP MM Service Architecture](image-url)

---

73
The ISC interface has been chosen to be SIP as specified in the RFC 2345bis specification of SIP [5] with additional enhancements introduced to support 3GPP's needs (see reference list).

The services can be executed on the SIP Application Server using methods like CGI, CPL or Java Servlets that has been introduced in the previous sections. The other possibility is to have the services being executed by a Parlay application residing on the OSA AS. Then OSA application accesses the network capabilities via the SCS. Until now SCS supports only services for the circuit switched domain. Enabling interface to the IP MM packet switched domain requires cooperation of SIP and OSA / Parlay methods and embraces a suitable protocol mapping between those two.

The point of interest is the signaling path between the S-CSCF, SCS and OSA AS. The SIP session control signaling toward the end user from the S-CSCF is not shown for simplicity reasons. The main goal is to validate 3GPP architecture in context of the IP MM service requirements for services execution. In the first stage only simple services in most general aspects will be investigated with respect to basic SIP messages and Generic Call Control API methods.

The SIP Application Server may include the Service Control Interaction Manager (SCIM) to support the services that require interaction between different applications. The S-CSCF, SIP Application Server and SCS implement the functionality of the SIP network entities. They can play the role of SIP User Agent, SIP Proxy and SIP Redirect Servers depending on the type of the service being executed and the current execution point. S-CSCF can additionally work as SIP Registrar Server. The picture below presents internal structure of S-CSCF, which handles independently the bi-directional SIP Call-Legs for incoming and outgoing signaling.

![Diagram](image)

**Figure 30.** Service Platform in the Proxy Mode
### 7.3 IMS Service Execution

The S-CSCF communicates with OSA Application Server (OSA AS) via the SCS which supports SIP on one side and OSA / Parlay interface on the other side. As specified in the [20] IP MM services are assumed to be triggered in the subscriber Home Network of the originating or the terminating party and can be executed in the Home Network or in the Visited Network of the triggered user. In the last case the serving S-CSCF will schedule the AS for execution and will interface external AS via Service Platform Gateway (SP GW). The exact information on, which AS should be scheduled for execution of the service is included in the Filter Criteria, stored at HSS and sent to S-CSCF during registration.

One of the requirements put on the ISC interface is filtering of the messages towards the AS based on AS specified criteria specified the Filter Criteria (FC). FC is a part of User Profile and is downloaded by the S-CSCF upon the user registration.

Filter Criteria (FC) is the information the S-CSCF receives from the HSS or from the AS that defines the relevant Service Points of Interest (SPIs) for a particular application. They select the subset of SIP requests received by the S-CSCF that should be sent / proxied to a particular application. SPIs are the points in the SIP signaling that may cause the S-CSCF to send / proxy the SIP message to a SIP AS, IM-SSF or OSA SCS. Filtering is done for SIP requests messages only and can be e.g. be based upon:

- The method of the SIP request.
- Whether the request was received in the originating or terminating case.
- A particular media type included in the SDP of a request.
- The presence/content of a particular SIP header.

Only the first SIP request is forwarded to the Service Logic (SIP Application Server or to SCS) when detected that some service should be executed. If Application Server wishes to remain in the signaling path it should insert its ID in the Record-Route header of the SIP request.

The Service Platform may utilize the following five basic modes of operation to support the service execution via the ISC interface.

1. Proxy mode
2. Redirect mode
3. User Agent mode
4. 3rd party controller mode
5. Not involved in the session

There are multiple modes allowed during one session and the current working model depends on the characteristics and nature of the service and required network elements interaction. In general S-CSCF, SIP AS and SCS implement one or more SIP Servers. It is however still open issue for which services, at which service execution points and under which conditions the different working flavors transitions are allowed to take place.
7.4 Working method

The IP MM will support a variety of services that can be provided by the network operator, 3rd party service provider or even by the end-users. The first attempt is to identify some problems around the service execution in the IP Multimedia Subsystem. The way of working is to look whiter OSA applications that could be deployed on the CS network could also directly work in the PS IMS network. The service execution signaling is presented below in the sequence diagrams and includes the following services:

- Call Barring
- Call Redirection
- Call Forwarding
- Prepaid Call

7.5 Assumptions

Symbolic interface names

This clear distinction between incoming and outgoing SIP legs may help to understand the functionality of S-CSCF addressing the issues with loop detection and prevention along the signaling path.

CSCF_IN: The interface between SIP terminal (UA Client A) and S-CSCF (SIP Server) in the originating network on which S-CSCF receives SIP requests from the terminal and sends back the responses.

CSCF_OUT: The interface between S-CSCF in originating network and the S-CSCF in the home network of the terminating party, on which S-CSCF sends requests to and receives responses from that network.

SCS-IN: The interface between the S-CSCF and SCS to send requests and receive responses.

SCS-OUT: The interface between the S-CSCF and SCS to receive requests and send responses.

![Figure 31. Auxiliary symbolic names for the S-CSCF interfaces](image-url)
Parlay APIs

All diagrams are based on a Generic Call Control API only and all method-calls between the OSA AS are SCS comes from that interface. This API has been chosen because it includes all basic methods that are sufficient to implement the set of selected services. The goal was to start with simple examples as the more complex APIs and services would make the investigation hard to proceed.

7.6 Results

7.6.1 Call Barring

In this example the call-terminating party is subscribed to the call barring service and the caller must provide the right pin to be able make a call to that user.

![Sequence diagram: Call Barring](image-url)

Remarks

a) It would be good having the originating party able to send the Pin together with its SDP in the first INVITE message to limit the traffic in the network. Maybe some CPL script could be downloaded and run on the originating terminal to provide the check of the pin before sending INVITE request.

b) It was assumed the OSA AS is not interested anymore in the call and issues deassignCall() request to SCS to indicate it is not interested any more in receiving other SIP messages. If application needs to be informed about the routing results it should remain in the signaling path to receive routeRes() back from SCS including information whether the call succeeded or not. After designing the Call SCS and S-CSCF in the user B home network remove theirs identities from the SIP message so the next messages will take another route in this session. This is also for the further study when and under which conditions is this allowed.
7.6.2 Call Redirection

In this example Service Platform acts as SIP Redirect Server by sending the 302 SIP response message indicating that the user being called has moved to other address. The originating party after receiving this message can choose to stop or to send again the INVITE message with modified address.

Remark: The question is: In which of the five modes should the Service Platform work in this example after sending the 302 SIP response? Does the AS require the SCS to send response to sendInfoReq() when the call proceeds or fails to proceed? In this case SCS and S-CSCF in the Home Network should remain in the signaling path but how? The originating UA Client will issue new SIP response with new Call-ID, but will the call really be routed to the new destination omitting the Home Network of the terminating user? The Home Network is the only place where the services are triggered in it is required to first request to be routed there, but in some cases like this it not necessary to do like this for the redirected call. Should the first S-CSCF to remember the previous transaction and associate it with a new one with different Call-ID and should not rout the request to the terminating Home Network?
7.6.3 Call Forwarding

7.6.3.1 Call Forwarding Busy

Here the called party is subscribed to the Call Forwarding service and while being busy the call is redirected to the voicemail. The 486 SIP response triggers invocation of the service that is like in all other cases executed in the Home Network.

![Sequence Diagram: Call Forwarding Busy](image)

**Figure 34.** Sequence Diagram: Call Forwarding Busy

Remarks

a) The application tends to setup the call and needs to be notified when the route has been established and wishes to remain in the signaling path. In case the last address is busy AS will redirect the calling party to the other address (probably voicemail) or will invoke user interaction service to ask him whether he or she wishes to do so.

b) Is this the right way the service should be triggered? Is this possible to have a user to be triggered on the SIP response? Maybe the transaction needs to be controlled already while receiving the SIP request to the terminating party. Then SCS remaining in the signaling path will receive the response indicating trigger conditions and the service should be invoked? This scenario presents the problem with the triggering mechanism. Should all SIP messages be checked for triggers or only requests? If requests only then the diagram above is incorrect.

This problem has been solved latter. The [25] defines that the services will be triggered only on reception of requests. So, in matter of fact the figure above presents wrong signaling path, as the service should be triggered in the Home Network every time a call request for the user subscribed to forwarding service is received. However, the necessary operations are performed after receiving response indicating that the user is already in call.
7.6.3.2 Call Forwarding No Answer

In this Call Forwarding scenario the call is being routed to the Home Network of the terminating party and after the timer expires routed on further towards the User B current location.

![Sequence Diagram: Call Forwarding No Answer](image)

**Figure 35. Sequence Diagram: Call Forwarding No Answer**

**Remark:** When the called party does not answer AS will probably try to contact some other addresses it has in its database associated with the user being called. If User B could nowhere be contacted the Service Platform will send 600 "Busy Everywhere" response. It would be better to attach to every request another address that should be tried for contact without having the call still coming back to the Home Network for service execution. Also a CPL document can be send in the body of the message to be run on the destination server.
7.6.3.3 Call Forwarding Unconditional (SCS in Proxy Mode)

In this example the Call Forwarding is unconditional. The SCS modifies the destination address and proxies SIP request to the next SIP server. It remains in the path to be notified if the call proceeds and choose another destination if necessary.

![Sequence Diagram: Call Forwarding Unconditional (SCS in Proxy Mode)](image)

**Figure 36. Sequence Diagram: Call Forwarding Unconditional (SCS in Proxy Mode)**
7.6.3.4 Call Forwarding Unconditional (SCS in B2BUA Mode)

The same as previous diagram with that difference that SCS is instructed by the Service Logic to issue a new SIP request with new Call-ID and this corresponds to the Back-To-Back-User-Agent mode (B2BUA). In previous case the `routeReq()` method included modified address and has been mapped on the same Call-ID as the SIP request had. There was only one SIP call leg in the network. Here a new SIP Call-ID is requested to be created resulting in a new SIP call leg. Consequently two different SIP call legs exist in the network.

![Sequence Diagram](image)

**Figure 37.** Sequence Diagram: Call Forwarding Unconditional (SCS in B2BUA Mode)

**Remarks:**

a) It should be clearer in which case the Service Platform should act as SIP Proxy or SIP User Agent during the session establishment.

b) Here again the big network bottleneck is encountered in case there are multiple addresses the call can be redirected to.
7.6.4 Prepaid Call

In prepaid call the application requests the network to be notified when some well-defined period of time expires in order to check the credits and decide whether the call can proceed. When the called party runs out of credit the OSA AS should instruct SCS to send an information to the caller that the call will be ended soon. If the terminal ignore the warning and does not issue the BYE request the application should terminate the call but sending a BYE request is not allowed. After the call has been terminated, in some unspecified way, AS asks the network about detailed information about the call by sending getCallInfoReq().

![Sequence Diagram: Prepaid call](image)

**Remarks:**

a) The application can instruct the SCS to issue the REFER method making the terminal to send BYE to the other party. In this case, however, this is violation of the 3GPP requirements prohibiting forcing UA to send such a request. UA must always present the routing request to the user and he or she can ignore the request and the call will not terminate. To solve this problem some other mechanisms should provide more reliable solution.

b) The Call and associated resources should be also released when the user out of the radio coverage area. SIP requests can not be sent anymore to or from terminals because there is no communication possible anymore. How to solve this problem?

c) Which network entity should report the bearer loss P-CSCF or MRF?
7.7 Open Issues

This study raised a number of questions.

7.7.1 Allowable Service Points of Interest (SPis)

As S-CSCF has its own Call-State-Model and looks for the match of the current call state with the information specified in the Filter Criteria for every incoming message of a specific user, it can be possible to trigger the services also on responses to the SIP requests and not only while receiving the first SIP request. It has of course much to do with the mechanism the services are triggered and the content of FC. The services are always triggered in the S-CSCF but should it happen on reception on requests only or on reception of responses as well?

Proposed solution

To preserve well-defined and unambiguous SIP message handling which would be fully coherent with the S-CSCF call state model definition it is suggested in [25] that services should be triggered only on reception of SIP request messages. In the case of a service that must be run on reception of a response the Service Logic should simply remain the session states until the response(s) is received. However, the beginning of the Session State tracking which is equivalent with the service triggering, does not imply the invocation of the service. The service is being invoked at appropriate Session State and on reception of an appropriate network event (response in this case).

7.7.2 Loop Detection

Does the S-CSCF implements two separate SIP servers to handle the incoming and outgoing SIP legs separately or does it include one SIP server together with additional logic to keep two different call state as given in Figure 30. If every SIP server must insert its own identity in the message being forwarded then S-CSCF receiving back the request previously forwarded to the SCS would cause SIP server in S-CSCF to detect the loop and tear down the session. This is the issue that has been discussed during one of the teleconference sessions of the IP Multimedia Task Force. Deeper investigation of the SIP functionality as protocol brought up significant clarification.

Loop Detection check: An element MUST check for forwarding loops before forwarding a request. If the request contains a Via header field value with A sent-by value that equals a value placed into previous requests by the proxy, the request has been forwarded by this element before. The request has either looped or is legitimately spiraling through the element. To determine if the request has looped, the element MUST perform the branch parameter calculation on this message and compare it to the parameter received in that Via field value. If the parameters match, the request has looped. If they differ, the request is spiraling, and processing continues. If a loop is detected, the element MUST return a 482 (Loop Detected) response.

Each SIP proxy MUST include a "branch" parameter (Section 22.40 of SIP RFC 2543) in the Via header. When the path of a request through one or more forking proxies is graphed, the result is a tree. The branch parameter identifies the "branch" each request was forwarded on. The branch parameter value MUST be unique for each client transaction to which the request is forwarded. In order to be able to both detect loops and associate responses with the corresponding request, the parameter SHOULD consist of two parts separable by the implementation.
The first part is used to detect loops and distinguish loops from spirals. The second is used to match responses to requests. Loop detection is performed by verifying that those fields having an impact on the routing decision have not changed. The value placed in that parts of the branch parameter SHOULD reflect all of those fields (which include any Proxy-Require and Proxy-Authorization headers). This is to ensure that if the request is routed back to the proxy and if one of those fields changes, it is treated as a spiral and not a loop.

7.7.3 SCS and CSCF operational behavior

What SIP network entities the S-CSCF and SCS should play and on which conditions and time stamps during the call setup they can change from one mode to another?

One solution would be to have S-CSCF collocated with the SCS and communicating via an internal API. This would preserve proper call related signaling and would not require from S-CSCF and SCS to change the working mode.

Another solution would be having S-CSCF and SCS stateful and tracking the call model separate for every SIP transaction. This would require that SCS or S-CSCF must be able to terminate SIP CallLeg on behalf of the end-user terminal and initiate its own dialog towards other network entities to provide required services and successfully complete the session establishment. This in turn implies flexibility in changing of the role playing by those network elements. If the session is being established for example to the user that has been set to perform redirection every time it is being contacted but based on the information accessed via SIP from other network resources then the SCS would terminate the first CallLeg from the initiation party and start its own dialog to obtain some forwarding criteria and then continue the first CallLeg to the final address. At the first moment SCS would act as the User Agent Server and then as a Proxy Server. So the role of the network entities should be not based on one SIP Call-ID only but should be based on the Parlay Call Leg ID and that may comprise multiple SIP Call-IDs. This would be the optimal solution for both SCS and S-CSCF.

7.8 Conclusions

The goal of this study was to determine how well the SIP protocol is able to play the role of the session control protocol in the PS IMS network. As the SCS already supports the access to the CS core network, it was reasonable to start looking at the simple services that are already implemented for the CAMEL / INAP networks. The validation of PS network support for those services has been done by analyzing the communication diagrams between the SIP-enabled terminal and the SCS with the S-CSCF there between.

This study has shown that the basic required functionality can be provided by Parlay / OSA APIs to operate on a SIP network. There is one main condition that the SCS and S-CSCF will provide efficient Parlay to OSA mapping what impose the use of heavy and advanced logic for interpreting of the Parlay messages and for maintaining the call state for each Parlay application initiated call as well. There may be a need to add some new methods to the Parlay interface for more advanced services but the generic set of services that have been investigated could be implemented.

Analyzing the sequence diagrams one issue arised. It was not well defined how the SCS and S-CSCF SIP should be assigned to the SIP network elements with respect to its functional behavior. The best solution is to allow the transitions of the working mode of SCS and S-CSCF. However this increases the complexity of the SCS implementation and the question may arise whether this will not become to heavy and whether this would degrade the network performance.
Final conclusion is that SIP can cooperate with Parlay / OSA, at least with respect to the basic services. This study, however, showed that there are some SIP-specific capabilities that seem to be not visible to the Parlay APIs. One example could be that currently APIs do not allow the Parlay application to control the media like media types or audio codecs. There is also a lack of support for session parameters negotiation capability and for specifying caller preferences. These issues should be taken into consideration in the adaptation of Parlay to SIP. I believe that SIP, as being the core network protocol, will have significant impact on the extension of Parlay / OSA APIs.
8. Parlay OSA enhancement

8.1 Summary

This chapter presents the investigation results on identifying unsupported features in the Parlay / OSA interface as compared to SIP functionality and provides some solution proposals for these problems. The next paragraph introduces shortly the concept of provisioning multimedia service in the IMS subsystem. Next to this the so far recognized problems and suggestions for improvements of the Parlay interface are listed. This work is primarily focused on the Multimedia Call Control API but covers also the basic type definitions specified in the Common Data Definitions API.

8.2 Parlay / OSA relation To SIP

There will be two different techniques to deploy services in the 3G network. The service providers can use these approaches to implement the services and offer them to the End-Users.

- In the first approach the end-user service is provided by the OSA application residing on the OSA AS in the Service Network. This can be achieved with Parlay / OSA APIs and the SCS providing the application access to the variety of the network functionality.
- Other choice is to have an application running on the SIP AS. Then the SIP service creation tools can be used to create required multimedia services. Most advanced SIP service creation methods include SIP CGI, Call Processing Language and SIP Servlets. In the case of CPL the End-User can play the role of the service developer.

To be able for OSA application to take fully advantage of the network functionality it is important that the interface between the application and the network is designed in a proper way. It should allow access to significant subset of network functions and provide required level of control and information to the application. The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardized interface, i.e. the OSA APIs. The API specification is contained in the 3GPP TS 29.198 series of specifications. The investigation of how OSA Multimedia Call Control Interface Class methods defined in 3GPP TS 29.198-4 can be mapped onto SIP methods raised a number of issues around visibility of the network functionality provided by SIP.

Because the Parlay / OSA interface is a standardized interface independent of the underlying network protocols and technologies it should be taken into consideration that modifying the interface has not as a goal to align it only to SIP but to implement functionality that applies also for all other network protocols. The main strategy is to hide as much of the underlying protocol complexity as possible from the OSA application allowing the interface to be merely modified and to remain as much as possible backwards compatible.
8.3 Parlay / OSA APIs Unsupported Features

The following list of issues has been identified so far and can be considered as subject for improvements in the Parlay / OSA interface.

1. Parlay versus SIP call models [32]

   The OSA application has limited view on the SIP call model (call leg visibility during the forking or forking visibility).
   - In some cases due to SIP forking an ordinary call may become a conference, when multiple end-points associated with the same user address will accept the call. It should be possible for the application to specify that no forking by the SCS is allowed or that the complexity of forking performed by the SCS should not be exposed to the application. Note that somewhere in the network the forking can still occur, downstream along the signaling path.
   - In other cases the application may want to have a control over forking in order for example to decide from which end point the call should be accepted. This may be required when OSA application originates the call and the originating party has specified restricting preferences for media, media compression methods or security levels for the call.
   - Which events in the core network are, as a minimum required to be reported to the application during the routing, in the middle of the call or during the session termination phase?

2. The visibility of the network bearer resources [37, 38, 40]

   The OSA application has no view on network resources used by the parties involved in the session like MRF or stream mixing capabilities (needed for Media => SDP). This problem could address also the issue around the identification of the source of the media stream on the bearer level (in case the application may wish to control the forking on its own). This problem comprises also issues around prioritization of the sequential forking. If the application (or SCS) is aware of the end-point network resources it can use this information to send requests in the order for example of the decreasing bandwidth owned by the terminal.

3. Support for transport of MIME types

   Currently Parlay / OSA interface doesn’t support the possibility to send an image to be displayed or a sound to be played to the call request that could be sent as attachments with SIP messages (support for transport of different MIME types). The session can be established as voice, video or data session but there is no support for customized way of presentation of the incoming call.

4. Media control and visibility [32]

   - Bandwidth limitations: early media aggregation when forking and OSA single medium

     A callLeg may automatically own a network resources when created but if request has been forked in the network then more then one end-point can accept the call. This can result in multiple media streams that will probably exceed the bandwidth of the media channel assigned to the callLeg.
Early media handling: attachment of media channels to the call (direct, on hold or early media modification) [37, 38, 40]

Due to the possibility of putting the call on hold or immediate attachment of the media to the call leg the interface should enable exact specification of whether one of those two variants should apply for the call being established. The possibility for denial of the early media should be addressed as well.

5. End-point addressing [16, 17]
   - The Parlay interface needs to support SIP addresses
   - Because of the variety of SIP addresses and associations of SIP addresses with different types of end-devices it may be desirable for the application to specify some routing path constrains along with the address.
   - The address in SIP can also specify the service to be invoked.

6. Mapping issues
   - Mapping of call id and leg id.
   - Mapping request methods (OPTIONS, REFER). It would be nice to have a general (not SIP specific) support for interrogating the network for its capabilities, checking user availability or requesting the SIP UA to issue some request to the other UA on behalf of the application. Very important thing is also creating headers of the SIP request message. This may be required that the application specifies which headers should the message include for example when the SCS wants to remain in the SIP signaling path the via header needs to be modified.
   - Normally when forking at the SCS or S-CSCF level occurs those entities should track and control the Branch parameter in the Via headers to be able to distinguish between the spirals and the loops. But in some cases the application may need also to control the branch parameter for example when it is controlling the forking by itself.
   - The negotiation capabilities for media, codecs and terminal capabilities should be addressed as well.
   - Mapping response codes: particular interesting are 180, 182, 183 and 603 response codes. This includes the mapping of the SIP responses received by the SCS like routing or address failures reports and also the mapping of the Parlay methods sent to the SCS and to be translated onto SIP response sent to the network. Also responses like 300 multiple choice or 380 alternative services should be addressed when the SCS needs to present possible options to the application before proceeding and application needs to make some decisions or contact the originating party.
   - Ability for the application to specify the media formats and codecs to be used and QoS.
8.4 Research Problems

The following problems has been selected for investigation:

1. **SIP address support.** Parlay *TpAddress* data type needs to handle SIP addresses. A SIP address consists of a single string prefixed by "sip:" and followed by text formatted similar to an email address, for example "sip:nick.edwards@bt.com". It can be also represented as an E164 number or an Internet Telephony user name (used by the gateways to route VoIP calls). Clearly some SIP addresses which represent other address types, e.g. E164 are already supported, but an address such as "sip:nick.edwards@bt.com" has no obvious Parlay analogue. So, it is currently unclear how SIP addresses should be handled in Parlay.

2. **Call Leg control.** The forking mechanism in SIP may result in that call can be setup from one caller to multiple destinations (callee’s). This can happen if a user is registered at more than one location and the service location lookup returned more than one address. The application may accept only a single destination address as result of the route request, however any route request may cause the call to be established to multiple destinations. It should be clarified how the call should be handled in this situation.

3. **Handling of media description.** In the Multimedia Call Control Service (MMCCS) the handling of media (SDP) at application initiated calls is left unspecified. There are also other situations when it may be required that the application needs to specify the session parameters on more detailed level. Also the current lack of support for certain media types shows the need to provide a flexible interface capability for extending the set of supported media types in a convenient way. This could be addressed by solving the following research questions:

   - How the terminal capability related information can be managed in the network so that it can be easily accessed by the authorized network entities or by the Parlay application?
   - How Parlay application could request and obtain that information?
   - How Parlay application could specify the session description parameters that could be mapped on the signaling message in the core network and determine the media type used in the session?

Also there are a number of routing options and conditions along the signaling path that could be set by the application and have an impact on the routing decisions in the network. As flexibility of the Parlay interface is an important issue all those problems should be solved.

4. **Session parameters negotiation capabilities.** In many situations it may be required to be aware or involved in the negotiation of some session parameters like compression or encryption algorithms that could be valid entire terminal registration time. Sip has build-in negotiation capability but it applies only for single session. The new SIP extensions, the NEGOTIATE method, is being proposed to solve this problem. In other cases there may be a need to negotiate the required bandwidth SDP port numbers or other session parameters and it could be valuable to solve all those problems by means of extending Parlay interface with an appropriate method.
8.4 SIP End-Point Addressing

8.4.1 SIP Address Type

This problem has already been addressed in [16, 17].
SIP uses Uniform Resource Identifiers (URIs) a Request-URI addressing scheme, which can be:

- a general URL as defined by the RFC 2396
  http://www.mediaserver.com/
- a SIP URL as defined by the RFC 2543bis-5
  sip: alice@gateway.com
  sip: +31-161-242-128@gateway.com
- a non-SIP format like that specified in the RFC 2806
  tel:+358-555-1234567@gateway.com

See [Appendix 3] for detailed description of those standards.

8.5.1 Problem Description

Parlay interface should support the SIP URL addressing scheme in order to enable efficient routing of the signaling messages in the network and to enhance the services identifying and service triggering methods.

8.5.1 Proposed Solution

In order to enable for OSA application to make a call to SIP enabled end-point devices the OSA / Parlay interface needs to support the SIP addressing scheme. Here follows the short description of the address related data types.

TpAddress

The address type TpAddress specifies Sequence of Data Elements data type that consists of sequence of the following data types:

- TpAddressPlan [Plan]
- TpString [AddrString, SubAddressString, Name]
- TpAddressPresentation [Presentation]
- TpAddressScreening [Screening]

AddrString parameter is of type TpString and represents a byte String. It defines an actual address information and is formatted according to the structure defined in TpAddressPlan data type.
As AddrString depends on the Plan parameter, the TpAddressPlan needs to be extended with the P_ADDRESS_PLAN_SIP type element in the following way:

**Suggested changes to TpAddressPlan:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESS_PLAN_SIP</td>
<td>12</td>
<td>SIP URL</td>
</tr>
</tbody>
</table>

**TpAddressPlan**

This has also an impact on the definition of the TpString data type and requires following changes:

<table>
<thead>
<tr>
<th>Address Plan</th>
<th>AddrString Format Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESS_PLAN_SIP</td>
<td>Address structure as specified in the RFC 2543bis-5: sip:user:password@host:port;url-parameters?headers or one of URLs as specified in the RFC 2806: phone-url = phone-scheme &quot;:&quot; phone-subscriber fax-url = fax-scheme &quot;:&quot; fax-subscriber modem-url = modem-scheme &quot;:&quot; remote-host</td>
<td>RFC 2543bis-5 examples: sip:user=host.com sip:user@1.255.255.1 sip:user*@host.com sip:<a href="mailto:1234@gateway.com">1234@gateway.com</a> sip:+1-123-456-7891:<a href="mailto:1234@gateway.com">1234@gateway.com</a>;user=phone sip:+1-123-456-7891:<a href="mailto:1234@host.com">1234@host.com</a>;user=ip sip:user;day=<a href="mailto:tuesday@host.com">tuesday@host.com</a> sip:host.com;method=REGISTER?to=user RFC 2806 examples: tel:+358-555-1234567 fax:+358.555.1234567 modem:+3585551234567;type=v32b?7e1;type=v110</td>
</tr>
</tbody>
</table>

**TpAddressRange**

TpAddressRange represents the same type as TpAddress with that difference that wildcards in the address string are allowed. In compliance with Parlay specifications the wildcards can be placed only at the beginning or at the end of the AddrString. As already mentioned in SIP URL the wildcards can be also placed between the "sip:" and "@" characters and modifying of TpAddressRange can cause potential problems because SIP URL allows the wildcards to be placed also between the "sip:" and "@" strings.

8.5.1 Open Issues

1. The Request-URI can contain URL address as defined in the RFC 2906 like:

   "sip:+1-123-456-7891:1234@host.com;user=ip"
   or
   "tel:+358-555-1234567".

The 3GPP requirements say that E.164 format public user identities must not be used for routing within the IM CN Subsystem, and session requests based upon E.164 format public user identities will require conversion into SIP URL format for internal IM CN Subsystem usage [20].
This means that this type of non-SIP URL addresses can be used in SIP INVITE method as Request-URI only if all outbound calls are handled by a proxy server that can translate the request URL to a SIP URL of a gateway server. Then Request-URI header is rewritten but the To header can still remain the same. To determine the next-hop SIP server the Telephony Routing over IP protocol (TRIP) can be used. The question is whether application needs to use this type of URL while being able to access the circuit switched network in more convenient way using the already existing methods.

2. Other problem is that SIP URL can contain wildcards, which are unfortunately not allowed in AddrString parameter. The special type that does not restrict the occurrence of the wildcards is the TpAddressRange data type but it does restrict the place where wildcards can be put in the AddrString. SIP URL also allows semicolon and other special characters in the user field like "user;day=tuesday".

3. The TpAddressRange data type should allow placement of the wildcards also in the middle of the AddrString. Because the AddrString, which represents SIP URL can also contain a wildcards and has a structure of TpAddressPlan it is questionable whether TpAddressRange type really needs to apply for SIP URL.
8.6  Call Leg Control

8.6.1  Summary

According to the Parlay call model the application may be aware of only single originating and terminating callLeg. However every user may have multiple terminal registered in the network. These could be end-devices capable of handling speech, video, data, high-quality audio, games or other types of media. Due to the SIP capability of forking the request to all possibly multiple addresses currently associated with a specific user, it can happen that more then one end-point would accept the call. Lets consider a voice-based session where the user will register three different terminals like mobile and fixed phones and the PC and all those end-devices will accept the call because they all support speech-based communication. In this case it should be well defined how SCS, receiving multiple responses from the network accepting the call, should behave and, if required, how to present the progress of the call to the application. In fact SCS may accept only one end-point and the signalling path to that end-point needs to be associated with just one already existing Parlay callLeg.

8.6.2  Parlay versus SIP Call Models

The Parlay call model assumes two types of callLegs to be assigned to the call object: originating and terminating callLegs. However when trying to create e.g. the terminating callLeg, the call request can be forked in the core network due to the SIP forking capabilities. This could lead to situation in which multiple end-points can accept the call what can result in multiple SIP callLegs to be created. The Parlay application can accept only one callLeg as result of a single request and attach it to the call and some selection / filtering criteria should be applied to allow mapping of the Parlay callLeg onto only one of possibly many available connection paths. The choosing of the SIP callLeg can be left up to the Application of can be done by the SCS and should be based on some selection criteria obtained from the originating party or could be based on the application preferences. Selection criteria can be specified by the end-user and captured in the service subscriber profile or send together with the request or decision of which selection criteria should be applied can be left up to the application or to the network.

There are two OSA-related scenarios when the application requests a call to be routed to one destination address and due to SIP forking multiple end-points accept the call:

- Application initiated call with two participants. In this case the two Parlay callLegs needs to be created and routed in the network. The SCS must provide appropriate Parlay callLegSessionID to SIP Call-ID mapping.
  SCS is working in B2BUA or 3rd Party Controller mode. Then the two Parlay callLegSessionID need to be mapped onto the two SIP Call-ID (one Call-ID for each SIP callLeg).
- Network initiated call. Here originating SIP callLeg already exist (SCS creates the corresponding Parlay callLeg and assigns the callLegSessionID to it, probably mapped from the SIP Call-ID) and only one (the terminating) Parlay callLeg needs to be created and routed.
  SCS can be working in Proxy mode and then two Parlay callLegSessionIDs needs to be associated with one SIP Call-ID. If the application requires that SCS has more control over the call then it would work in a B2BUA or 3rd Party Controller mode. In this case the Parlay callLegSessionID to SIP Call-ID mapping is one-to-one mapping (two Parlay callLegs mapped onto two SIP callLegs)

95
The two main decisions should be made to avoid the ambiguity during setting up a call that has been forked. Those address the following issues:

- **Where decision about whether the forking is allowed or not should be made?**
  Shall this decision be up to SCS or to the application?

- **If forking is allowed should it be visible and controlled by the OSA application or not?**
  SCS can select one of the possible SIP callLegs and make it visible to the application or should it present the result of the forking to the application and let it choose one of the SIP callLegs?

### 8.6.3 Assumptions

- The forking is done by S-CSCF and S-CSCF will forward all responses to SCS.
- If forking occurs on the level deeper then S-CSCF and multiple terminal will accept the call then all responses will be forwarded to the SCS.
- All call identities and callLeg identities are created / assigned by the SCS.
- As application is interested in only one connection (actually in a certain media type) and only one responding end-point needs to be selected the complexity of doing this is left up to the SCS and not to S-CSCF. Alternatively this choice can be left up to the application itself.
- Error reporting are not covered here.
- All issues are concentrated around S-CSCF, SCS and OSA Application in the home network of a user for which OSA Application runs a service.
- The SIP RFC 2543bis-05 states: "Multiple 2xx responses may arrive at the UAC for a single INVITE request due to a forking proxy" and it is assumed that no forking proxy will cancel still outstanding requests when receiving the first 2xx response from one of the end-points.
- Application can accept only one SIP callLeg that will be mapped on the Parlay callLeg as result of a single route request.
- Every SIP-enabled end-point always includes a valid SDP packet in the response to the SIP INVITE request even if the request contained no session description.
- The term "forking" denotes here "parallel-forking". The sequential forking is much less complex and do not introduces any problems.
- No media attachment / detachment issues are covered here.
8.6.4 Proposed Solutions

8.6.4.1 Forking never allowed in the network.

There are two possible solutions:

a) SCS can always add a Request-Disposition header with the value "No-Forking" to every SIP request. Then if every SIP server in the network support this header, then the forking will never happen. See the SIP RFC 2543bis-05 to find out how to create SIP Request-Disposition header. This solution, as opposed to the next proposal, requires that all servers in the signaling path support the Request-Disposition header, which is defined for SIP as optional. The second disadvantage is that if the call will be routed to the terminal that does not fulfill the session description requirements then the call establishment error will be reported to the application.

b) The application can request user location and user status related information from the network before trying to route the call to the terminating party. Then a new method needs to be added to one of Parlay APIs to discover the supported media types for every of currently registered user terminals. If application knows where the user is available at that moment and knows exactly what terminals the user has registered to the network then it would directly choose one of the possible end-point that exactly matches required session description parameters like media type, bandwidth and media codec. Then the application will put this session description information in the SDP in the outgoing SIP message. The SCS can additionally put the Request-Disposition header in the SIP INVITE message but if anyway the call will be forked then all other SIP UAs should respond with "415 Unsupported Media Type" or "406 Not Acceptable" messages because only the one selected terminal can accept the specified session description criteria. This issue addresses the SDP mapping and will be not further discussed here.

8.6.4.2 Application decides whether the forking is allowed or not.

Here if application decides to allow forking, the forking can be controlled by the application or by the SCS.

Two implementation approaches are suggested:

a) Application can specify that no forking is allowed by setting some Boolean value in the route request. This solution requires adding a Boolean parameter to all route requests. The SCS will add a Request-Disposition header to every outgoing SIP message. If the forking is allowed (Boolean parameter set to true) then the application takes control over the forking. In this situation the SCS will never set the SIP Request-Disposition header and every SIP forking proxy downstream in the signaling path will be allowed to fork a SIP request if needed. The Parlay interface could be additionally modified to allow the presentation of the forking results one by one to the application. The existing event notification methods could not be used for this purpose as those are related to the new incoming call and have an effect of creating a new call. This implies the necessity for new method that would create a new CallLeg and present the reference to it to the Application. This method could be based on the PartyJoined
event in the conference call. The Multi Media API could as well inherit for this purpose an existing method from Multi Party API but proposed solution would be better because of solving two problems at ones. The method would only report new connection path and the fact of creation of a new CallLeg associated with it but leave the attachment of this CallLeg up to the Application.

All SIP connections would be established on-hold and the application will need to make a choice from the number of possible connection paths. SCS should report the incoming SIP callLeg as soon as it receives the first response from the network. The application decides then which CallLeg from the set of reported CallLegs it wants to accept by specifying the CallLegs that should be destroyed. The attachMedia () could also be used to explicitly attach the callLeg object to the call and setup the media stream on the chosen CallLeg. Release () or deassign () methods could be used by the application to destroy other existing CallLegs.

b) In this scenario the same rules applies as above. The application uses the Boolean parameter in the route request to switch on or off the forking but the control over the forking and taking decision on selecting a SIP callLeg is left up to the SCS. Then SCS can use some processing instructions / criteria associated with a specific user or service or if not available apply it own rules for the call.

8.6.4.3 Forking always allowed.

This solution boils down to having SCS taking decision whether the forking is allowed or not and eventually controlling the forking. The forking, if necessary, is done transparently to the application and a SIP callLeg is selected according to some filtering / priority criteria available to the network (SCS) as proposed in 1.2.b

This approach assumes the existence of some kind of filtering criteria available to the SCS in order to be able to choose one of the possible SIP callLegs. The method for obtaining this information and managing user preferences related data is outside of the scope of the problem discussed in this section and will not be covered here in details. A possible solution is that the SCS can obtain that information from HSS or other database on request. In further discussion we will assume existence of such an auxiliary information. The resulting behavior of the SCS will be applying the selection criteria, selecting the right SIP callLeg and making it visible to the application.

8.6.4.4 What is the preferred method ?

The first scenario is the easiest one but definitely not the best one. It is very likely that users will have multiple terminals registered in the network and forking can considerably enhance the call session establishment and is a desired functionality. The question is also whether the new method in Call Control API in order to support obtaining information about user preferences regarding preferred terminal type would be efficiently mapped onto SIP OPTIONS method and provide detailed terminal capabilities description. However, due to forking problems with OPTIONS, the empty INVITE method could also be used to discover all required information.

The second proposal would require adding one Boolean parameter to all routing related methods and a new value in the TpCallMonitoreMode data type. Also a new method newCallLegCreated () for presenting the results of the forking to the application should be defined.
The third option would require no changes in the routing methods towards the SCS as all incoming from the network events could be handled with already existing call event notification. This is probably the simplest solution but it puts the complexity in the user/service related data management. However it would require no Parlay interface extensions. The forking would then be always allowed and handled by the SCS transparently to the application. The fact is that if the application would have to make decision on selecting one of the possible SIP callLegs it would be based on some information that is better available to the SCS then to the application itself. If even the application will have some information over user preferences obtained during the subscription to the service it would not be the real-time and up-to-date information. The availability of the user and currently registered addresses would need to be still discovered every time the call is being made. So this solution seems to be more reliable.

To preserve the best flexibility of Parlay interface the best solution is to have the forking being presented to the application, which in turn could select one of SIP CallLegs obtained from the SCS. The Parlay interface would require additional method for presentation of the CallLeg being created. New created CallLegs will be automatically attached to the Call and selecting of one right one could be done by using already existing release() method on all other CallLegs. This would mean for the SCS that the CallLeg not being destroyed has been chosen by the Application.

8.6.4.5 Motivation for the proposed solution

The user may wish always to have its call being forked e.g. for recording purposes so that it is important to allow the forking in the network. Also forking provides the user search capabilities of the network and reduces the session establishment time if parallel forking is used. Disabling of the forking will require that the application or SCS must always know all addresses associated with a specific user at each call establishment attempt. Then if one call fails to be established the application will try sequentially all other possibilities what will lead to increased traffic in the network and considerably delayed session establishment time. So the first option can be discarded.

From remaining two, the first one seems to be the best solution. It provides valuable flexibility to the interface so the application can fully control the forking. Although it implies some changes to the API, those changes are minimal and can be reused for PartyJoined method.
8.6.5 Proposed Implementation

8.6.5.1 New method in the Terminal Capabilities and User Profile API Interface should be added to discover the media supported by the terminal.

```plaintext
mediaReportRequest (appMedia : in IpAppUserMediaRef,
user : in TpAddress,
assignmentId : out TpSessionIDRef) : TpResult
```

In 1.1.b the application may use `locationReportReq()` method to discover all possible user locations then `statusReportRequest()` to check the status of the user and finally the `mediaReportRequest()` to discover the media types for every terminal registered by the user.

**Parameters**

- `appMedia`: in `IpAppUserMediaRef`
  Specifies the application interface for callbacks from the Multi Media service.
- `users`: in `TpAddressSet`
  Specifies the user(s) for which the supported media shall be reported.
- `assignmentId`: out `TpSessionIDRef`
  Specifies the assignment ID of the media-report request.

*Define new `IpAppUserMediaRef` type or check if one of the existing types associated with media could be used.*

8.6.5.2 In 8.6.4.2.a and 8.6.4.2.b The Boolean parameter needs to be added to all routing requests.

This solution has impact on the `createAndRouteCallLegReq()` and `routeReq()` route request methods:

- If `createAndRouteCallLegReq()` will be used for request the proper monitoring mode should be set to enable forking results notification. This method requests creation and routing of a new CallLeg. After successful routing the CallLeg will be attached to the Call and no explicit `attachMedia()` operation is needed. This request can be used anytime because is independent of other methods and includes event report subscription requests in it. The `createAndRouteCallLegReq()` operation is performed on the Call object.

```plaintext
createAndRouteCallLegReq (   
callSessionID : in TpSessionID,
  eventsRequested : in TpCallEventRequestSet^2,
  targetAddress : in TpAddress,
  originatingAddress : in TpAddress,
  applInfo : in TpCallAppInfoSet,
  appLegInterface : in IpAppCallLegRef
  ForkAllowed : TpBoolean) : TpCallLegIdentifier
```

^2 See Appendix
At the same time `TpCallMonitorMode` as being a data element in the `TpCallEventRequest` and `TpMediaStreamRequest` types should be extended with

\[ \text{P\_CALL\_MONITOR\_MODE\_NOTIFY\_FORK} \]

and additionally `TpCallEventType` should be extended with

\[ \text{P\_CALL\_EVENT\_FORKED} \]

**Note 1:** `P\_CALL\_MONITOR\_MODE\_NOTIFY\_FORK` can be used only with conjunction with `ForkAllowed` set to `true`.

**Note 2:** The SCS can see from the received SIP response that the call has been forked and can discover the number of possible responses applying the methods presented in Annex 1. If routing response is received in the monitor mode set to `P\_CALL\_MONITOR\_MODE\_NOTIFY\_FORK` then SCS will specify the `P\_CALL\_EVENT\_FORKED` parameter in the event report and will indicate to the application that multiple responses are possible.

- The `routeReq()` method from the Multi Media API can also be used on the CallLeg object but it must be proceed first with the `createCallLeg()` method proceeded with the `eventReportReq()` method that set triggers on the network events.

```plaintext
callLegSessionID : in TpSessionID,
targetAddress : in TpAddress,
originatingAddress : in TpAddress,
appInfo : in TpCallAppInfoSet,
connectionProperties : in TpCallLegConnectionProperties
forkingAllowed : TpBoolean)
```

**Note:** `P\_CALL\_MONITOR\_MODE\_DO\_NOT\_NOTIFY\_FORK` is unnecessary because it would be equivalent to already existing `P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR`.

The `routeReq()` method could be proceeded with the following event subscription method:

```plaintext
callLegSessionID : in TpSessionID,

eventsRequested : in TpCallEventRequestSet)
```

and then the following methods could be used to report the forking related events:

```plaintext
callLegSessionID : in TpSessionID,

eventInfo : in TpCallEventInfo)
```

The following methods could be used by the application to destroy callLegs.

```plaintext
callLegSessionID : in TpSessionID,

cause : in TpReleaseCause)
```
This method requests the release of the call object and associated objects. If this method is used with only one callLeg associated with the call then also the Call will be terminated in the network but requested call-related events will still be sent to the application.

```
defassign(callLegSessionID: in TpSessionID): void
```

This method requests to tear the relationship between the application and the callLeg and associated objects be deleted. It leaves the callLeg in progress, however, it purges the specified callLeg object so that the application has no further control of the callLeg processing. All event reports are disabled.

**Note:** It may be required the `TpCallNotificationInfo` and `TpCallEventInfo` needs to be modified to better support the forking result event notification.

Finally the new method for presentation the 200 OK SIP response and creation of a new callLeg would look like:

```
newCallLegCreated (callSessionID: in TpSessionID, callLeg: in TpCallLegIdentifierRef, eventInfo: in TpEventInfo, appLegInterface: in TpAppCallRef): void
```

This method would report to the Application the successful routing of one of possibly multiple connection paths resulting in creation of a new callLeg. This callLeg needs still to be explicitly attached by the Application to the call. This is different from the `eventReportRes()` which should be used to report successful routing request that has not been forked.

This new method could make the Application aware of possible multiple connection paths to be returned. Depending on which method has been used for route request the rules of applying `newCallLegCreated()` method or `eventReportNotification()` should be defined. The route request method will always create a callLeg object. Then if the request has been forked and the 300 Multiple Choices SIP response has been reported by network before 200 OK was received from the first SIP branch, the SCS would use normal response method associates with the route request operation and set the `_P_CALL_EVENT_FORKED` parameter. Also the number of SIP branches discovered from the 300 Multiple Choices response could be presented to the Application. This will make the Application aware of multiple responses that could be received and will be immediately followed by the `newCallLegCreated()` method presenting the first SIP callLeg. Then the application can decide whether to wait or use available callLeg. After making decision the old terminating callLeg should be destroyed.

Alternatively the method `eventReportRes()` could be used to present the result of forking to the application. It should be modified to allow notification of new created callLeg.

**8.6.5.3** No changes to the Parlay interface required. The appropriate solutions for managing the auxiliary processing instructions to deal with forking by SCS and methods for obtaining are for further study.
8.7 Session Description Control

8.7.1 Introduction

This section addresses the following issues that are further described in more details:

- How the terminal capability related information can be managed in the network so that it can be easily accessed by the authorized network entities or by the Parlay application?
- How Parlay application could request and obtain that information?
- How Parlay application could specify the session description parameters that could be used in the session?
- How application can specify certain request handling directives for the message routing by proxies and redirect servers in the network.

All those problems relate to the session establishment procedure when application request the network to route the call to the destination party and also address the issues around specifying the session description parameters.

In SIP this is the normal case that the User Agent Client (UAC) issuing an INVITE request posses the media resource(s) and includes SDP in the session invitation. After receiving 200 OK response from the User Agent Server (UAS) residing on the other terminal UAC will send ACK message starting communication. The invited party can also initiate the media stream immediately after receiving the INVITE message from the originating terminal if that message includes the session parameters description. This is so-called Early-Media transmission. **Support for early media is important for interoperability with the PSTN.** Additionally three way handshake call establishment can take up to ten seconds before the media stream can be send!

On the other hand when INVITE message include no SDP then no media stream can be sent even after receiving the ACK message because the characteristics of the session like media type and RTP port numbers are not known. This is the way the session can be established on-hold. Dependent on the service type the OSA Application may need to have the possibility to implement the call according to both scenarios and the Parlay interface should provide this flexibility.

In the Multimedia Call Control Service (MMCCS) the handling of media (SDP) at the application-initiated calls is left unspecified. Also the terminal on the network may issue a call request towards the application that contains no SDP. This can happened when the SIP or Parlay application will execute on the terminal and will issue an empty INVITE message to the other party causing the S-CSCF of that party being triggered for some Parlay service. Then the request will be forwarded to the SCS and further to the application which will receive a call request with no SDP. This is unclear how Parlay application should behave in this situation. This section presents solution to this problem.

Considering the routing of the message in the network there are a number of routing decisions that the user originating the call or the application may be interested in to control. The examples may include:

- whether to proxy or redirect the request
- which URLs to proxy or redirect to
- whether to fork or not
- whether to search recursively or not
- whether to search in parallel or sequentially

103
The directives of the administrator are embedded in the policy of the server and the preferences of the call terminating party can be expressed through a CPL script running on the terminating server. However there is no possibility for the call originating party or network entity to specify its preferences regarding message processing in the network.

For example, the requestor might want to speak to a specific user, but want to reach them only at work, because the call is a business call or the requestor might want to reach a user, but not their voicemail, since it is important that the caller talk to the called party. There should be the possibility to have a proxy in the network making a particular routing decisions based on the preferences of the caller.

In SIP for example the Request- Disposition header describes desired server behavior and can be used to instruct the server to proxy or redirect the request message, to specify whether sequential or parallel search is desired or to prohibit the forking in the network. Accept-Contact and Reject-Contact headers can be used to specify the caller preferences about the addresses he or she would like the request to be routed to. In SIP these types of preferences can appear in any request, not just in the INVITE message.

The other issue is to have some database base on the network where the user profile and the terminal capabilities can be stored every time the terminal registers to the network. For example, there could be a mobility parameter in the terminal profile, which indicates whether the UA is fixed, or mobile. The terminal profile could also include the parameter describing the type of the terminal like "voicemail" or "PC" that could be put in the request messages or responses to indicate that the terminal is a voicemail server or a computer.

As flexibility of the Parlay interface is an important issue all those problems should be solved. In the next section several possible scenarios are being discussed:

8.7.2 SDP Discussion

Network originated call

- The application receives from the network a call request that includes media specifications and can use it to request the routing of the terminating callLeg. This is the scenario that raises no problems.
- There is a network-originated call but INVITE message mapped onto the Parlay event notification method, did not include any SDP. The Parlay application can try to discover the media description supported by the originating terminal, but this would probably make no sense since the reason the terminal did not include that information in the call request may be that it does not posses any media related resources. In this case the one choice for the application would be to send an empty INVITE and establish the call on hold or apply some default session parameters and request from the network resources conforming those specifications.

Application initiated call

- If application initiate the call between two parties then it has no information related to the media types supported by both end-user terminals and the call can be established on-hold. The session parameters are then discovered from the SIP endpoint responses. Other choice is to request those information from the network database where it was stored during the registration process.
Note that a sending a media stream from the media server to a SIP terminal is equivalent to the SIP call between two parties as media server is a normal SIP User Agent Server (UAS).

The other question is when is would or not be required that the Early Media stream must used during the call setup. This requirement that SDP must be used in the first INVITE message when application request the call being routed in the network can possibly depend on the service type.

**SDP Not Required**

- If it is not required that the media must be attached immediately after routing the callLeg then no information about the call media type needs to be specified in the initial call request and can be obtained at the latter time from the terminal response. The SCS can send to the first party an empty INVITE message with no SDP. The terminal of party A will put its SDP in the response message. Then the INVITE to the second party can contain the SDP of the first party received this way. After receiving response and SDP from the party B the new INVITE is sent to party A containing the session description parameters of party B. SCS could also choose to sent two empty INVITE messages to both parties simultaneously and after obtaining SDP from both terminals it would re-INVITE them with SDP being intersection of both session description parameters. In this case the SCS can be working in 3rd Party Controller mode.

The short description is presented in the [Appendix 11] based on the draft *Third Party Call Control in SIP* [35].

**SDP Required**

- If the media must be attached to the call immediately after routing the SIP callLeg then the application or SCS should possess sufficient information about the media types to be used in the call before the request is sent to the network. This implies that the range of media types supported by the call-originating terminal should be known in order to create SDP session description message. On example could be the communication between critical medical devices located in two different hospitals or emergency calls.

One solution is as already said to let the SCS to reserve some default resources and put some default media description in the outgoing INVITE. Other option is to obtain those informations from some database on the network where the terminal related informations are stored for example at the registration time.

Those informations may include:

- media types
- encryption algorithms
- preferred bit rate
- preferred codec
- required bandwidth
- port numbers for media stream
It is also desired that the application can specify the preferred media types or routing related call parameters as well. On the other hand the application should be able also to request and obtain the terminal related informations from the network.

8.7.3 Proposed solutions

Assumptions

- All User Agent Clients (UAC) in the network always put its session description parameters (SDP) in the response to the INVITE message (even if it includes no SDP).

8.7.3.1 Specifying session-related en routing parameters

The interface can support specifying these parameters or may remain untouched.

This solution implies changes in the Parlay API

- The first approach lets the application to explicitly specify session description details. Putting detailed media description in the Parlay interface increases the complexity of the Parlay interface but provides the application more control over the session description parameters.

The Multimedia Call Control API supports the basic media types like voice, video and data. The TpCallAppInfoSet data type gives the possibility to specify the type of session that should be established in the route request methods. This allows specifying a number of Bearer-Services like speech, 3.1 kHz audio or video and a number of Tele-Services like telephony, fax, telex and videotext. This could be however insufficient in the service-provisioning environment that requires a variety of new different services to be supported. The high quality audio and video, which strongly depends on, the available bandwidth and the codec type available for use by the end-point and by the network raises the question whether the MPCCS can be used for the application initiated calls. There the media specification is a part of the session initiation message (INVITE with SDP).

The session description parameters could be included in the Service Properties but this would need to apply to all calls initiated from the SCS and would be not a good solution. The other possibility for application initiated calls is that the OSA SCS or S-CSCF would not posses any media resource function (MRF), i.e. its UAC will have to generate an INVITE with no SDP as already said. However, how could it work and what if some services would require media types that are supported by the network but not by Parlay API? Some new Parlay method would solve this problem and provide extensibility to provide support for new media types and other session-related parameters.

These are proposed solutions:

Combining the subject worked out in the previous section about application controlled call forking in the network with controlling the SDP in the INVITE messages it is suggested to create a new data type of type “Sequence of Data Elements” allowing specification of the media related parameters and routing related parameters. Not being focus on the SIP protocol only it seems very valuable if the application would specify some conditions for the routing of the call request and instruct the network entities how to process the signaling messages to complete the session establishment.
The new `TpRequestOptions` data type could be defined and added to the route request methods. This is the proposed implementation of the `TpRequestOptions` data type.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDP</td>
<td><code>TpSessionDescription</code></td>
</tr>
<tr>
<td>RoutingDispositions</td>
<td><code>TpRoutingDispositions</code></td>
</tr>
</tbody>
</table>

The `TpSessionDescription` data type could be used to specify the media type and other session parameters to be used in the call. `TpRoutingDispositions` can be the way of instructing the network about the processing of signaling messages and could be applicable to set for example the Header fields that are additional named attributes providing additional information about a message.

This could be the definition of the `TpSessionDescription` data type.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaType</td>
<td><code>TpMediaType</code></td>
</tr>
<tr>
<td>TransportProtocol</td>
<td><code>TpTransportProtocol</code></td>
</tr>
<tr>
<td>Bandwidth</td>
<td><code>TpBandwidthRange</code></td>
</tr>
<tr>
<td>PortNumber</td>
<td><code>TpInt32</code></td>
</tr>
<tr>
<td>Codec</td>
<td><code>TpCodecSet</code></td>
</tr>
<tr>
<td>ipAddress</td>
<td><code>TpAddress</code></td>
</tr>
</tbody>
</table>

The same way the `TpCodecSet` data type would be defined and an example of the implementation on the language level would be:

```c
typedef struct TpCodecSet {
    TpInt32 Number; [size_is(Number)] TpMedia Set[];
} TpCodecSet;

typedef struct {
    TpString Manufacturer;
    TpCompressionType CompressionType;
    TpString CodecName;
    TpBitrate Bitrate;
    TpInt32 BufferSize;
} TpCodec;
```

`TpRoutingDispositions` data type definition

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignalingParameters</td>
<td><code>TpSignalingParameters</code></td>
</tr>
<tr>
<td>ReportForking</td>
<td><code>TpBoolean</code></td>
</tr>
<tr>
<td>ForkingType</td>
<td><code>TpForkingTypeSet</code></td>
</tr>
<tr>
<td>ForwardingAllowed</td>
<td><code>TpBoolean</code></td>
</tr>
<tr>
<td>ReportForward</td>
<td><code>TpBoolean</code></td>
</tr>
</tbody>
</table>
MaxNumberOfHops | TpInt32
CallPresentationAllowed | TpBoolean

The ReportForking parameter would be set to *true* to instruct the SIP Server to report the call forking event in the network. The parameter would be used in connection indicate whether the SCS or CSCF want remain in the signalling path or not.

ForkingType can be used to specify the forking type that needs to be used in the network. TpForkingTypeSet is a set of data elements of type TP\text{ForkingType}. The TpForkingType data type is of type Tp\text{String} and contains two values {Parallel, Sequential}.

ForwardingAllowed can specify whether call forwarding is allowed or not. On the other hand ReportForward could be used to request the forward event done in the Network to be reported to the Application.

The MaxNumberOfHops would cause setting of the Max-Forwards SIP header and the CallPresentationAllowed parameter would specify whether the name of the caller could be presented to the callee.

### SigalingParameters data type definition

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessingDispositions</td>
<td>Tp\text{ProcessingDispositionsSet}</td>
</tr>
<tr>
<td>RequireFeatures</td>
<td>Tp\text{String}</td>
</tr>
<tr>
<td>SupportedFeature</td>
<td>Tp\text{String}</td>
</tr>
<tr>
<td>SetRecordRoute</td>
<td>Tp\text{Boolean}</td>
</tr>
</tbody>
</table>

The ProcessingDisposition could be made up from several data elements of type Tp\text{String} and used for example to prohibit the call request from being forked in the network. The FORKINGNOTALLOWED value from that set could be used to instruct the SCS to insert the Request-Disposition header with value "NoForking" so that request will never be forked in the network.

SetRecordRoute could be used to indicate that the Application wishes to remain in the signaling path of subsequent messages. The SCS will then add a Record-Route header to the outgoing INVITE message.

### The solution having no impact on the Parlay interface.

- In the second case the relevant information and session parameters description could be automatically associated with a specific service or user and requested by the SCS automatically at the time the call is being made. An XML-based document would be probably a good candidate.

#### 8.7.3.2 Obtaining user profile and/or terminal capabilities

The application providing a service to the well-known end-user that is subscribed to this service, can take advantage of the user-profile information to get awareness about the media type that the user can potentially posses.
During the subscription to any service that will require the application to initiate the call on behalf of the user the user can always specify one or more end-point devices, which can be automatically, associated with certain media types. Customized media descriptions by the user should also be possible. This information can be stored in the subscription profile and can be associated with the service. Then if the application is about to make a call it knows the media type to be used by the end-point. In case the user specified to use only one end-point device the application can use this information to route call to that user's terminal. If multiple terminals of different types have been registered by the end-user, he or she may has been specified the preferences about in which order they wish to be contacted. This information should be then passed together along with the route request to the SCS and used for prioritization of the sequential forking.

This user specific information can be stored:

- at any database accessible by the application in the Service Network
- can be downloaded e.g. from the HSS or other database.

Note that in those cases the application will have some information that is not necessarily up-to-date and real-time information.

The other solution guarantees more recent and actual information.

If there will be the possibility for the application to discover the terminal capability and resources possessed by a terminal it would use it to specify the detailed session description parameters in the route request.

The method proposed in the previous section to be added to the User Profile and Terminal Capability API could be used.

8.7.3.3 Managing the terminal capabilities and user profile informations in the network

Both proposals suggest uploading of these informations during the terminal registration to the network.

- One solution would give the user the possibility to upload the CPL script in the normal REGISTER message. Then the S-CSCF that function as a Registrar server would issue an OPTIONS request towards the terminal to discover the terminal capabilities. This information could be stored in some database for example in the HSS. Only terminal related informations would be obtained this way but it would be desired also to request some user and service related informations. This is addressed by the second option.
- The second possibility is to have the terminal issue two or more different REGISTER messages to the different network entities and with different contents. The first message could include CPL script and the second the terminal capabilities or user profile. The following two drafts SIP Caller Preferences and Callee Capabilities [36] and Supporting Mobility for Multimedia with SIP [34] describes in details the implementation concepts.
8.8 Session Parameters Negotiation Capabilities

8.8.1 Problem Description

a) In the network initiated call during the session establishment time it can happen that the media types supported by the terminating party doesn't match that specified by the originating party. Also in the case of two party application initiated call where application sends an empty INVITE message to receive SDP from both terminals in the responses it can happen that there will be no intersection of the supported media types. That would mean that application might need tear down current call and try a number of times with other media types until match would be found. Instead some negotiation dialog would be started to agree on some media types and complete the call that is already in progress.

For this reason some negotiation capability in the Parlay interface would be useful. If the session could not be established because of the mismatch of supported media types the appropriate method would still enable the application to negotiate and to agree on some other parameters. This would prevent SCS from reporting error to the application.

b) In many situations it may be required to negotiate compression or encryption algorithms that could be valid entire terminal registration time. Sip has build-in negotiation capability but it applies only for single session. The new SIP extension, the NEGOTIATE method, is being proposed to solve this problem on a per registration basis.

c) In other cases there may be a need to negotiate other session parameters. The examples of session parameters that may cause the session establishment to fail but may still be negotiated may include:

- media types
- encryption algorithms
- compression algorithms (codecs)
- code book size
- message integrity mechanisms
  - minimum / maximum bit rate
  - minimum bandwidth
  - QoS parameters
- RTP port numbers
- the URI to which the request should be routed to if there are multiple choices
- whether to proxy or redirect the request
  - whether to fork or not
  - whether to search recursively or not
  - whether to search in parallel or sequentially

Actually all the parameters suggested to be specified by the application in the route request methods described in the previous section could apply.

In SIP for example the INVITE method describes the parameters for initiation of a particular session. Once the session is over, the negotiated settings (such as the RTP profile used, in the case of SDP) are invalidated for future re-use. It would be inefficient to have to renegotiate parameters that are the same from session to session, or have to transmit large quantities of persistent data (such as a compression code book) each time.
All those problems could be solved by means of extending the Parlay interface with an appropriate Parlay method implementing the negotiation capability.

A good working example would be the scenario from the previous section about the application callLeg control. If the forking proxy is about to send a request to the multiple destinations its could issue a 300 Multiple Choices response towards the SCS. According to the problems with interpretation of the SIP RFC it is not clear whether the proxy must issue this response. This should be required from the implementation of SIP proxy that it supports this feature as mandatory. As 3GPP already adopted many of new SIP extensions beyond SIP RFC, this should not be the problem. From 300 Multiple Choices response SCS can discover the set the addresses currently registered to the user and make use of the negotiation interface to present those choices to the application. Application could then select one of possible destination addresses probably requesting first the terminal capabilities of every terminal associated with a given address. Compared with application callLeg control from the previous scenario this brings the following enhancement:

- No multiple SIP callLegs must be setup towards the SCS
- No time wasting by the application with waiting for forked responses

This example shows the need for new Parlay method that could be mapped on to the NEGOTIATE SIP method to let the forking proxy know to which address to forward the request.

It is believed that also the following routing errors reported with Warning header could be negotiated:

- Incompatible network protocol
- Incompatible network address formats
- Incompatible transport protocol
- Incompatible bandwidth units
- Media type not available
- Incompatible media format
- Session description parameter not understood
- Multicast not available
- Insufficient bandwidth
- Unicast not available

Also the following responses indicating failure could be subject for session parameters negotiation:

- 380 Alternative Service (The call was not successful, but alternative services are possible)
- 415 Unsupported Media Type
- 420 Bad Extension supported unsupported methods (protocol extension)
- 503 Service Unavailable
- 505 Version Not Supported (SIP protocol version)
- 485 Ambiguous (The callee address provided in the request was ambiguous. The response MAY contain a listing of possible unambiguous addresses in Contact headers)
- 484 Address Incomplete
8.8.2 Network and Service Requirements

3GPP requirements [20] says it must be possible to negotiate:

- Compression
- QoS
- Bearers

This boils down to the following issues:

a) In establishing a SIP session, it must be possible for an application to request that the resources needed for bearer establishment are successfully allocated before the destination user is alerted.

b) In establishing a SIP session, it must be possible for a terminating application to allow the destination user to participate in determining which bearers shall be established.

c) Successful bearer establishment must include the completion of any required end-to-end QoS signaling, negotiation and resource allocation.

8.8.3 Proposed Solution and Implementation

No new Parlay method is required. If the terminal to which the call was routed does not support certain media types specified in the routeReq() method no routeErr() should be reported to the application. The routing results could be reported to the application by already existing routeRes() method. However, TpCallReport data type should be extended with the event types conforming the TpRequestOptions data type to enable the reporting of possible options to successfully complete call establishment. TpCallAdditionalReportInfo data type could be extended with appropriate event types to report those network events.

routeRes (callSessionID: in TpSessionID
   eventReport: in TpCallReport
   callLegSessionID: in TpSessionID )

Figure 39. Sequence Diagram: Media Negotiation
9. Conclusions

The main goal of this Master Thesis was to investigate whether Parlay / OSA architecture with its APIs can be applied to a SIP network. To address this issue the following questions must be answered:

1. What is OSA?
2. What is SIP?
3. What are the methods for programming services in both environments?
4. How do they compete with each other?
5. How Parlay / OSA and SIP architectures relate to each other?
6. What are the main issues in combining those two?
7. How to solve these issues?
8. What to do next?

Open System Architecture (OSA) provides for the applications in the Service Network an open and standardized interface to the Core Network resources. OSA is based on the set of Parlay APIs that has been adopted by the telecommunication community in the context of 3GPP. The access to the network is guaranteed by the so-called Service Capability Server that offers those APIs. SCS expose to the Parlay applications the capabilities of the network in a safe and controlled manner.

In the Core Network of 3G UMTS the Session Initiation Protocol (SIP) has been chosen as the signaling protocol for establishing end-to-end multimedia connections. SIP is being standardized within IETF. It supports setup, modification and tear down of the session but has limited capabilities to control an ongoing session. However, SIP is extendable and many extensions are being proposed. SIP can be compared to ISUP / ISDN and in that matter it is very different from other session control protocols currently being used for the telecom services. With relation to the protocols currently used in the 2G networks like CAP for example, SIP differs significantly in that it is simple, text-based and provides the separation of the call setup signaling from the control of the media channels what allows better use of the network resources.

Three major service creation methods recommended by IETF have been investigated:

- SIP Servlets
- Call Processing Language (CPL)
- Common Gateway Interface for SIP (SIP CGI)

The major differences between those are as follows:

- CGI offers limited service functionality. It is stable and has been around for a while but seems not to take off. The reason may be that despite of many programmers familiar with CGI, it is more and more seen as an old fashion technology. CGI SIP comes with a mature RFC but there is still no implementation to found on the technology market.
- CPL offers even more restricted service functionality, but is available. The key benefit of CPL is that CPL scripts are small XML-based text documents that can be easily validated at the server and run safely. CPL targets directly the end-user and allows creation of very simple services. CPL scripts can be stored on the user's terminal and uploaded to the network during the registration time. CPL specifications and implementing are also quite mature and CPL is actually as the only one ready to be used for end-user service creation.
- SIP Servlets offer the richest functionality. It is currently being standardized. This is the most attractive option for use in the 3G service creation platform. Services are composed from building blocks, the so-called servlets. The Servlet API provides the interaction between those components that constitute a service. This interface support also extension of the service platform with a new servlets.

With respect to the Parlay / OSA service creation environment it can be stated that Parlay API provide a perfect application creation and service provisioning solutions. They are applicable not only to 3G, but also to 2G and 2.5G networks. The Parlay APIs and their programming tools are mature and provide a great opportunity for programming services, especially over heterogeneous network with different access types. They allow for the combination of different network capabilities that are provided via a variety of APIs. They can be also extended with new APIs and capabilities that are not even standardized. Parlay APIs provide also very high level of reliability, extensibility and flexibility and for this reason they are slowly becoming a standard in the areas of multimedia service creation and provisioning.

Comparing Parlay / OSA APIs an SIP servlets API the statement can be made that both service creation methods offer a great opportunity to the service application developers. An important difference is that for SIP Servlets API service creation and provisioning platform no security infrastructure has yet been addressed. The SIP Servlets API based architecture can take advantage of the standard Java security model but will this be sufficient? The security issue together with support for 3rd party service providers seems to be a significant problem in the SIP environment and no solution has been found yet. However, combining this SIP-based architecture with Parlay / OSA architecture would alleviate this problem. The Parlay / OSA architecture can be combined with that of SIP and add the valuable security level that is actually missing in the SIP environment. So both architectures can complement each other in service provisioning.

Summarizing relation of SIP and Parlay / OSA service architectures the statement can be made that SIP-based service creation platform allows easier migration from existing architecture to the new one based on SIP while Parlay requires SCS and Parlay AS. On the other hand Parlay / OSA easily supports 3rd parties and is usable not only for 3G networks which in fact are not implemented yet but also for 2G and 2.5 G networks. This however is not true for the SIP-based service creation methods that can only be used in the 3G networks. Here ones again the advantages of Parlay / OSA APIs meet the requirements of the nowadays service development environment.

The study in the area of interworking capability of SIP and Parlay has brought up some weaknesses of Parlay as compared to SIP. The main idea of Parlay interfaces is to hide the network specific informations from the application developers. However this decrease radically the control the application can have on the session establishment. Also the Network Operator may wish to expose the SIP-based network functionality on the Parlay service APIs. There are a variety of SIP specific services that seem not to be accessible through the Parlay API. The effort to alleviate this problem resulted in a number of suggestions how Parlay / OSA could be enhanced with respect to those issues. A slightly modification of Parlay interface could improve the mapping onto SIP without adding much complexity to the APIs. At the same time the more functionality of SIP and its valuable capabilities could be exposed to the application developers without significant limitations. Some solutions have been found and proposed for those issues, involving the adaptation of OSA.
Finally it can be stated that SIP can already now cooperate with Parlay / OSA, at least with respect to the basic multimedia services like Call Barring, Call Forwarding, Call Redirection and Prepaid Call. There is some missing functionality in SIP with respect to 3GPP requirements and also OSA / Parlay APIs exhibit some lack of supported features as compared to the SIP. So there are still several issues to be resolved, including the precise mapping by the SCS between OSA events and SIP request methods and responses. The standardization of SIP and Parlay APIs progress rapidly and this problems will be certainly addressed in the next standards release.

The main conclusion is that combining the Parlay / OSA architecture and the SIP architecture can be done and makes sense. The Parlay API is already rich enough to support many of the capabilities provided by a SIP core network. However it is sure that SIP Core Networks will continue to have a significant impact onto the future of the Parlay APIs and the service provisioning architecture.
10 References

2. OPEN SERVICE ARCHITECTURE

[1] Open Service Architecture: Concept and Standards
Ard-Jan Moerdijk, Lucas Klostermann
IN and 3G Mobile, Computer Communications Journal, 2001

Stefan Gartner
Ericsson Developers Zone, January 2001

[3] Parlay User Status Simulator
Aleksander Bordjiasevic

3GPP TS 23.228, 2001-06

3. SIP NETWORK

[5] SIP, SESSION INITIATION PROTOCOL
J. ROSENBERG, H. SCHULZRINNE, G. CAMARILLO,
A. JOHNSTON, J. PETERSON, R. SPARKS, M. HANDLEY, E. SCHOOLER.
DRAFT-ietf-sip-rfc2543bis-05.txt, OCTOBER 2001

Thomas Holmlund, Ulf Stenskog
MASTER THESIS, April 2000

4. SIP SERVICE CREATION

[7] XML for creating 3G Services (CPL)
Jose Costa Requena, Laboratory of Telecommunications Technology,
Helsinki University of Technology

[8] CPL: A Language for User Control of Internet Telephony Services
Lennox/Schulzrinne
IPTEL WG, draft-ietf-iptel-cpl-04, November 2001

[9] The Call Processing Language: User Control of Internet Telephony Services
Jonathan Lennox,
February 28, 2000

[10] Call Processing Language Framework and Requirements
J. Lennox, H. Schulzrinne
Network Working Group, Columbia University, May 2000
5. **PARLAY / OSA SERVICE CREATION ENVIRONMENT**

[12] Open Service Architecture: Concept and Standards  
Ard-Jan Moerdijk, Lucas Klostermann  
IN and 3G Mobile, Computer Communications Journal, 2001

[13] The Ericsson Service Network: a “melting pot” for mobile Internet service creation and delivery  
Roger Staton  
SN_article_for_Ericsson_Review_draft_1_010107, January 2001

[14] Open Service Access: Advantages and Opportunities in service provisioning on 3G Mobile Networks  
Michal Waldken, Nick Edwards, David Foster (BT), Milan Jankovic, Borislaw Odadzic (Community of Yugoslav PTT), Gaute Nygreen, Geir Gylteryd (Telenor), Corrado Moiso, Sergio M. Tegnon (Telecom Italia Lab), Brendan de Bruin,  
Eric Prigent (Broadcom)  
Eurescom, September 2001

6. **PARLAY / OSA VERSUS SIP SERVICE CREATION ENVIRONMENT**

Stavros Papastavrou, Panos Chrysanthis, George Samaras, Evaggelia Pitaura

[16] Parlay and SIP  
Nick Edwards, CEG1, BT Adastral Parkn, November, 2000

[17] SIP Support in Parlay  
Nick Edwards  
Meeting 10, Antwerp, March 2001

[18] Siplets: Java-based service programming for IP telephony  
W. van Leekwijk, D. Brouns  
Alcatel Telecommunications, 2nd Quarter 2000

Roch H. Glitho  
Ericsson Research Canada, August 2000

7. **SERVICE CONTROL IN IMS SUBSYSTEM VIA SIP**

[20] 3GPP requirements on SIP  
draft-garcia-sipping-3gpp-reqs-00, November 2001
[21] Reliability of Provisional Responses in SIP
J. Rosenberg, H. Schulzrinne
draft-ietf-sip-100rel-04.txt, September 2001, work in progress.

[22] SIP Caller Preferences and Callee Capabilities
Schulzrinne/Rosenber, from Columbia U./dynamicsoft, SIP WG
draft-ietf-sip-callerprefs-05.txt November 21, 2001

[23] IP Multimedia Call Control Protocol based on SIP and SDP (Release 5)
3GPP TS 24.229, October 2001

[24] Signaling flows for the IP multimedia call control based on SIP and SDP
3GPP TS 24.228, October 2001

[25] IP Multimedia (IM) Session Handling;
IP Multimedia (IM) call model (Release 5)
3GPP TS 23.218, 2001-10

[26] CR to 23.218: Information flows for IMS service examples: Call Forwarding Service
Control Scenarios
3GPP TSG CN WG1 Meeting, October 2001

[27] SIP Call Flow Examples
Alan Johnston, Steve Donovan, Robert Sparks, Chris Cunningham, Jonathan
Rosenberg, Kevin Summers, Henning Schulzrinne
draft-ietf-sip-call-flows-05, June 2001

[28] SIP Service Examples
Alan Johnston, Robert Sparks, Chris Cunningham, Steve Donovan, Kevin Summers
draft-ietf-sip-service-examples-02, June 2001

[29] 3GPP IP Multimedia CN Subsystem - Session establishment flows
K. Drage & M. Orsic (Lucent Technologies), W. Marshall, J. O/EHare (Motorola),
R. Zwart (AT&T), S. Olsen (Ericsson), B. Bertenyi (Nokia),
J. Bharatia (Nortel Networks)
draft-drage-3gpp-sip-establish-00, February 2001

[30] 3GPP IP Multimedia CN Subsystem - Registration flows
K. Drage, M. Orsic, Lucent Technologies, W. Marshall, R. Zwart, AT&T, S. Olsen,
Ericsson, J. O/EHare, Motorola, G. Bajko, Nokia, J. Bharatia, Nortel Networks
draft-drage-3gpp-sip-registration-00, February 2001

[31] Reliability of Provisional Responses in SIP
J. Rosenberg, H. Schulzrinne, SIP WG
draft-ietf-sip-100rel-0, September 19, 2001

8. PARLAY OSA ENHANCEMENT

[32] OSA API MPCCS: SIP mapping open issues
Jorgen Dysf, Kindy Sylla
Meeting 15, Cancun, Mexico, November 2001
[33] OSA AND IPMM REPORT, 3GPP AND THE ISC MODEL
Jørgen Dyst, November 2001

[34] Supporting Mobility for Multimedia with SIP
H. Schulzrinne, S. Baba, N. Nakajima, Y. Shobatake, M. Tauil,
J-C. Chen, A. Dutta, F. Vakil
draft-itsumo-sipping-mobility-multimedia-01.txt, July 2001

[35] Third Party Call Control in SIP
Rosenberg/Peterson/Schulzrinne/Camarillo, SIP WG
Draft-rosenberg-sip-3pcc-02, March 2, 2001

[36] SIP Caller Preferences and Callee Capabilities
Schulzrinne/Rosenberg, SIP WG
draft-ietf-sip-callerprefs-05, November 21, 2001

[37] SIP Early Media
Jonathan Rosenberg, SIP WG
draft-rosenberg-sip-early-media-00, July 13, 2001

[38] Early Media Issues and Scenarios
Sanjoy Sen, Jayshree Bharatia, Chris Hogg, Francois Audet, NORTEL NETWORKS
draft-sen-sip-earlymedia-00, July 13, 2001

[39] The SIP Negotiate Method
Sriram Parameswar, Brian Stucker, Nortel Networks
draft-spbs-sipping-negotiate-00, August 2001

[40] SIP Forked Media
Mahesh Shankar, Cisco Systems
draft-shankar-sip-forked-media-00, February 2001

[41] Control of Service Context using SIP Request-URI
B. Campbell, R. Sparks, Dynamicsoft
RFC 3087, April 2001

[42] Third Party Call Control for Resource Management
F. Haerens, Alcatel
draft-haerens-sip-3pcc-00, February 2001
In order to make this graduation project successfully proceed it has been divided into several phases. It will help to keep planning and realization of tasks under control regarding the time it should take and will improve the methods used to achieve the integration of entire project. Each phase contains a number of activities that have to be done at an appropriate time. The following table specifies the planning and realised results. The whole graduation project was divided into the following phases and activities:

<table>
<thead>
<tr>
<th>Research Activity Description</th>
<th>Duration</th>
<th>Realization Date</th>
<th>PROGRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Planned</td>
<td>Realized</td>
</tr>
<tr>
<td>Study on specifications and requirements for 3GPP</td>
<td>1 month</td>
<td>23.04 - 23.05</td>
<td>23.04 - 23.05</td>
</tr>
<tr>
<td>Writing the first part of the Master Thesis work</td>
<td>1 week</td>
<td>23.05 - 02.06</td>
<td>23.05 - 02.06</td>
</tr>
<tr>
<td>SIP protocol and SIP network architecture</td>
<td>2 weeks</td>
<td>08.06 - 23.06</td>
<td>08.06 - 28.06</td>
</tr>
<tr>
<td>Writing SIP introduction chapter</td>
<td>1 week</td>
<td>23.06 - 29.06</td>
<td>28.06 - 04.07</td>
</tr>
<tr>
<td>SIP development environment</td>
<td>1 week</td>
<td>29.06 - 07.07</td>
<td>04.07 - 13.07</td>
</tr>
<tr>
<td>Writing SIP service creation chapter</td>
<td>1 week</td>
<td>13.07 - 20.07</td>
<td>13.07 - 20.07</td>
</tr>
<tr>
<td>Parlay / OSA development environment</td>
<td>3 days</td>
<td>20.07 - 23.07</td>
<td>20.07 - 28.07</td>
</tr>
<tr>
<td>Writing Parlay / OSA service creation chapter</td>
<td>3 days</td>
<td>23.07 - 27.07</td>
<td>28.07 - 30.07</td>
</tr>
<tr>
<td>Review of SIP and OSA service creation</td>
<td>1 week</td>
<td>01.08 - 08.08</td>
<td>01.08 - 08.08</td>
</tr>
<tr>
<td>Preparing slides for presentation</td>
<td>2 weeks</td>
<td>08.08 - 23.08</td>
<td>08.08 - 05.09</td>
</tr>
<tr>
<td>Sick</td>
<td></td>
<td></td>
<td>05.09 - 15.08</td>
</tr>
<tr>
<td>Vacation</td>
<td>2 weeks</td>
<td>15.09 - 29.09</td>
<td>17.09 - 29.09</td>
</tr>
<tr>
<td>Parlay / OSA and SIP in IMS subsystem</td>
<td>2 weeks</td>
<td>30.09 - 15.10</td>
<td>30.09 - 15.10</td>
</tr>
<tr>
<td>Sequence Diagrams for GCC-based IP MM services</td>
<td>1 week</td>
<td>15.10 - 21.10</td>
<td>15.10 - 21.10</td>
</tr>
<tr>
<td>Writing report on SIP and OSA interworking in IMS</td>
<td>3 days</td>
<td>21.10 - 24.10</td>
<td>21.10 - 24.10</td>
</tr>
<tr>
<td>Sequence diagrams for MPC and other APIs</td>
<td>3 days</td>
<td>25.10 - 28.10</td>
<td>skipped</td>
</tr>
<tr>
<td>OSA enhancement issues</td>
<td>1.5 month</td>
<td>28.10 - 14.12</td>
<td>06.11 - 14.12</td>
</tr>
<tr>
<td>Outstanding examination (Licht en Geluid)</td>
<td>2 weeks</td>
<td>November</td>
<td>08.12 - 20.12</td>
</tr>
<tr>
<td>Biblioteekpracticum</td>
<td>1 week</td>
<td>Januari</td>
<td>03.01 - 08.01</td>
</tr>
<tr>
<td>Merging the entire Master Thesis document</td>
<td>3 weeks</td>
<td>02.01 - 21.01</td>
<td>07.01 - 22.01</td>
</tr>
<tr>
<td>Master Thesis work training and presentation</td>
<td>2 weeks</td>
<td>21.01 - 30.01</td>
<td>In progress</td>
</tr>
</tbody>
</table>

**Figure 40. Planning Table**
Appendix 2

Ericsson Eurolab in the Netherlands

Ericsson is the leader in the so-called New Telecom World providing innovative communication solutions, which combine telecommunication and datacommunication technologies with an optimal mobility and interaction for the end user. With 100,000 employees in more than 140 countries all around the world, Ericsson makes the communication simpler facing the high requirements of the network operators, service providers and enterprises and satisfying millions of customers.

Ericsson is a Swedish company with the base in Stockholm. It is active in the Netherlands since 1893. In 1920 the Ericsson Telefoon Maatschappij (ETM) was founded and that was the first Ericsson-daughter outside of Sweden. From October 2000 all R&D activities in the Netherlands are united into one R&D Unit, named Ericsson EuroLabs Netherlands (ELN). ELN will develop several technologies, products and services, varying from W-CDMA base stations and Bluetooth products to innovative Internet applications, Intelligent Networks, Charging, Billing and Accounting Systems and Announcement Systems. The activities take place at the present companies Ericsson Business Mobile Networks, located in Enschede, Emmen and Hoofddorp and at the R&D units of Ericsson Telecommunication in Rijen.

I was assigned to the Application Solution Center and made part of the IP Multimedia Task Force responsible for development of the IP Multimedia System in 3G network.
Appendix 3  
SIP Address Format

SIP uses a Request-URI addressing scheme, which can be:
- a general URL as defined by the RFC 2396
- a SIP URL as defined by the RFC 2543bis-5.
- a non-SIP format like that specified in the RFC 2806.

RFC 2396. The standardized RFC 2396 scheme is already supported by the Parlay. Examples:

ftp://ftp.is.co.za/rfc/rfc1808.txt
gopher://spinaltap.micro.umn.edu/00/Weather/California/Los%20Angeles
http://www.math.uio.no/faq/compression-faq/part1.html
mailto:mduerst@ifi.unizh.ch
news:comp.infosystems.www.serversunix
telnet://melvyl.ucop.edu/

RFC 2543bis-5. SIP Request-URI as specified in the RFC 2543bis-5 indicates the user or service to which SIP request is being addressed. This will be usually a SIP URL which identifies a communications resource. Examples of communications resources include:
- a user of an online service
- an appearance on a multiline phone
- a mailbox on a messaging system
- a PSTN phone number at a gateway service
- a group (such as "sales" or "helpdesk") in an organization

The general form of SIP URL is: **sip: user: password@host: port; url-parameters ? headers**

**User:** User field is optional and if used must be followed by a "@" sign. SIP URL without "@" sign refers to a domain. If the host (like Internet Telephony gateway) being addressed is capable of processing telephone numbers, a telephone-subscriber field defined in RFC 2806 may be used in user field.

**Password:** It is not recommended to use the password for authentication because this field is not encoded and sent as text exposes contained information to the network entities.

**Host:** This field contains the domain name or the numeric IP address (IPv4 or IPv6).

**Port:** This is the IP port number where the request is to be sent.

**URL-Parameters:** This field can indicate the following:
- **transport:** The transport protocol which can be UDP, TCP, TLS and SCTP.
- **maddr:** The server address to be contacted in the numeric form superior to that indicated in the host field. It can be used to force the message to take the route via this server.
- **ttl:** Time-To-Live for the SIP request used only with conjunction with multicast maddr address and UDP transport.
- **User and method parameters.** If the user field indicates a real telephone number the user parameter should contain "phone" string.
Example: **sip:+31-161-242-128:1234@gateway.com;user=phone**
- Method parameter can specify the method to which the URL can be mapped.

**Headers:** This parameter can describe the subject and body headers of the SIP request.
RFC 2806. Request – URI can also have a non-SIP format like that specified in the RFC 2806. This RFC defines URL (Uniform Resource Locator) schemes like “tel”, “fax” and “modem” for specifying the location of a terminal in the phone network and the connection types (modes of operation) that can be used to connect to that entity.

Digital phone networks distinguish between voice, fax and data calls. To be able to successfully connect to, say, a fax machine, the caller may have to specify that a fax call is being made. Otherwise the call might be routed to the voice number of the subscriber. In this sense, the call type is an integral part of the ‘location’ of the target resource.

Examples:

tel:+358-555-1234567. This URL points to a phone number capable of receiving voice calls
fax:+358.555.1234567. This above URL describes a phone number which can receive fax calls.
modem:+3585551234567;type=v32b??7e1;type=v110. This phone number belongs to an entity which is able to receive data calls. The local entity may opt to use either a ITU-T V.32bis modem (or a faster one, which is compatible with V.32bis), using settings of 7 data bits, even parity and one stop bit, or an ISDN connection using ITU-T V.110 protocol.
Appendix 4  Parlay Basic Data Types

**TpString**
Defines a Byte string, comprising length and data. The length must be at least a 16 bit integer.

**Sequence of Data Elements**
This describes a sequence of data types. This may be defined as a structure (for example, in C++) or simply a sequence of data elements within a structure. C++ example for *TpAddress*:

```c++
typedef struct {
    TpAddressPlan  Plan;
    TpString       AddrString;
    TpString       Name;
    TpAddressPresentation.....Presentation;
    ....TpAddressScreening.......Screening;
    ....TpString..................SubAddressString;
} TpAddress;
```

**Numbered Set of Data Elements**
This describes a data type which comprises an integer which indicates the total number of data elements in the set (the *number* part), and an *unordered* set of data elements (the *data* part). Set data types do not contain duplicate data elements. C++ example for *TpAddressSet*:

```c++
typedef struct TpAddressSet {
    TInt32 Number; [size_is(Number)] TpAddress Set[];
} TpAddressSet;
```

**TpResult**
Defines the Sequence of Data Elements that specify the result of a method call. All methods in the Parlay APIs return a result of type *TpResult*.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResultType</td>
<td>TpResultType</td>
</tr>
<tr>
<td>ResultFacility</td>
<td>TpResultFacility</td>
</tr>
<tr>
<td>ResultInfo</td>
<td>TpResultInfo</td>
</tr>
</tbody>
</table>

**TpResultType**
Defines whether the method was successful or not.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_RESULT_FAILURE</td>
<td>0</td>
<td>Method failed</td>
</tr>
<tr>
<td>P_RESULT_SUCCESS</td>
<td>1</td>
<td>Method was successful</td>
</tr>
</tbody>
</table>

**TpResultFacility**
Defines the facility code of a result. In phase 2 of the Parlay APIs, only *P_RESULT_FACILITY_UNDEFINED* must be used.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_RESULT_FACILITY_UNDEFINED</td>
<td>0</td>
<td>Undefined</td>
</tr>
</tbody>
</table>
**TpResultlnfo**
Defines further information relating to the result of the method, such as error codes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_INVALID_DOMAIN_ID</td>
<td>0001h</td>
<td>Invalid client ID</td>
</tr>
<tr>
<td>P_INVALID_SERVICE_ID</td>
<td>0006h</td>
<td>Invalid service ID</td>
</tr>
<tr>
<td>P_INVALID_EVENT_TYPE</td>
<td>0007h</td>
<td>Invalid event type</td>
</tr>
<tr>
<td>P_SERVICE_NOT_ENABLED</td>
<td>0008h</td>
<td>The service ID does not correspond to a service that has been enabled</td>
</tr>
<tr>
<td>P_INVALID_SERVICE_TOKEN</td>
<td>0013h</td>
<td>The service token has not been issued, or it has expired.</td>
</tr>
<tr>
<td>P_USER_NOT_SUBSCRIBED</td>
<td>0030h</td>
<td>An application is unauthorised to access information and request services with regards to users that are not subscribed to the application.</td>
</tr>
<tr>
<td>P_ILLEGAL_SERVICE_TYPE</td>
<td>0033h</td>
<td>The specified name is not a valid Service Type name</td>
</tr>
<tr>
<td>P_UNKNOWN_SERVICE_TYPE</td>
<td>0034h</td>
<td>The specified Service Type name is valid, but the Framework does not currently support it.</td>
</tr>
<tr>
<td>P_GCCS_INVALID_ADDRESS</td>
<td>0103h</td>
<td>Invalid address specified</td>
</tr>
<tr>
<td>P_GCCS_INVALID_CRITERIA</td>
<td>0104h</td>
<td>Invalid criteria specified</td>
</tr>
<tr>
<td>P_GMS_INVALID_MAILBOX</td>
<td>0200h</td>
<td>Invalid mailbox number</td>
</tr>
<tr>
<td>P_GMS_INVALID_AUTHENTICATION_INFO</td>
<td>0201h</td>
<td>Invalid authentication information</td>
</tr>
<tr>
<td>P_GUIS_INVALID_CRITERIA</td>
<td>0300h</td>
<td>Invalid criteria specified</td>
</tr>
<tr>
<td>P_GUIS_ILLEGAL_ID</td>
<td>0301h</td>
<td>Information id specified is invalid</td>
</tr>
</tbody>
</table>
Appendix 5  Relevant Parlay Address Types

AddrString
Defines the sequence of data elements that specify an address.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>TpAddressPlan</td>
</tr>
<tr>
<td>AddrString</td>
<td>TpString</td>
</tr>
<tr>
<td>Name</td>
<td>TpString</td>
</tr>
<tr>
<td>Presentation</td>
<td>TpAddressPresentation</td>
</tr>
<tr>
<td>Screening</td>
<td>TpAddressScreening</td>
</tr>
<tr>
<td>SubAddressString</td>
<td>TpString</td>
</tr>
</tbody>
</table>

The AddrString defines the actual address information and the structure of the string depends on the Plan. The following table gives an overview of the format of the AddrString for the different address plans.

<table>
<thead>
<tr>
<th>Address Plan</th>
<th>AddrString Format Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESS_PLAN_NOT_PRESENT</td>
<td>Not applicable</td>
<td></td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_UNDEFINED</td>
<td>Not applicable</td>
<td></td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_IP</td>
<td>For IPv4 the dotted quad notation is used. Also for IPv6 the dotted notation is used. The address can optionally be followed by a port number separated by a colon.</td>
<td>&quot;127.0.0.1:42&quot;</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_MULTICAST</td>
<td>An IPv4 class D address or IPv6 equivalent in dotted notation.</td>
<td>&quot;224.0.0.0&quot;</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_UNICAST</td>
<td>A non multicast or broadcast IP address in dotted notation.</td>
<td>&quot;127.0.0.1&quot;</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_MULTICAST</td>
<td>An international number without the international access code, including the country code and excluding the leading zero of the area code.</td>
<td>&quot;13161249111&quot;</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_AESA</td>
<td>The ATM End System Address in binary format (40 bytes)</td>
<td>01234567890abcdef01234567890ab cdef01234567</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_URL</td>
<td>A uniform resource locator as defined in IETF RFC 1738</td>
<td>&quot;<a href="http://www.parlay.org">http://www.parlay.org</a>&quot;</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_NSAP</td>
<td>The binary representation of the Network Service Access Point</td>
<td>490001a00040010420</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_SMTP</td>
<td>An e-mail address as specified in IETF RFC22</td>
<td>&quot;<a href="mailto:webmaster@parlay.org">webmaster@parlay.org</a>&quot;</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_MSMAIL</td>
<td>Identical to P_ADDRESS_PLAN_SMTP</td>
<td>&quot;<a href="mailto:john.doe@hiitech.com">john.doe@hiitech.com</a>&quot;</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_X400</td>
<td>The X400 address structured as a set of attribute value pairs separated by semicolons.</td>
<td>&quot;C=nl;ADMD=value;O=parlay;S=Doe;I=S;G=John&quot;</td>
</tr>
</tbody>
</table>

TpAddressPresentation
Defines whether an address can be presented to an end user.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESS_PRESENTATION_UNDEFINED</td>
<td>0</td>
<td>Undefined</td>
</tr>
<tr>
<td>P_ADDRESS_PRESENTATION_ALLOWED</td>
<td>1</td>
<td>Presentation Allowed</td>
</tr>
<tr>
<td>P_ADDRESS_PRESENTATION_RESTRICTED</td>
<td>2</td>
<td>Presentation Restricted</td>
</tr>
<tr>
<td>P_ADDRESS_PRESENTATION_ADDRESS_NOT_AVAILABLE</td>
<td>3</td>
<td>Address not available for presentation</td>
</tr>
</tbody>
</table>
**TpAddressScreening**

Defines whether an address can be presented to an end user.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESS_SCREENING_UNDEFINED</td>
<td>0</td>
<td>Undefined</td>
</tr>
<tr>
<td>P_ADDRESS_SCREENING_USER_VERIFIED_PASSED</td>
<td>1</td>
<td>User provided address verified and passed</td>
</tr>
<tr>
<td>P_ADDRESS_SCREENING_USER_NOT_VERIFIED</td>
<td>2</td>
<td>User not verified</td>
</tr>
<tr>
<td>P_ADDRESS_SCREENING_USER_VERIFIED_FAILED</td>
<td>3</td>
<td>User provided address verified and failed</td>
</tr>
<tr>
<td>P_ADDRESS_SCREENING_NETWORK</td>
<td>4</td>
<td>Network provided address (Note that even though the application may provide the address to the gateway, from the end-user point of view it is still regarded as a network provided address)</td>
</tr>
</tbody>
</table>

**TpAddressPlan**

Defines the address plan (or numbering plan) used. It is also used to indicate whether an address is actually defined in a TpAddress data element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESS_PLAN_NOT_PRESENT</td>
<td>0</td>
<td>No Address Present</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_UNDEFINED</td>
<td>1</td>
<td>Undefined</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_IP</td>
<td>2</td>
<td>IP</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_MULTICAST</td>
<td>3</td>
<td>Multicast</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_UNICAST</td>
<td>4</td>
<td>Unicast</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_E164</td>
<td>5</td>
<td>E.164</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_AESE</td>
<td>6</td>
<td>AESE</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_URL</td>
<td>7</td>
<td>URL</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_NSAP</td>
<td>8</td>
<td>NSAP</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_SMTPE</td>
<td>9</td>
<td>SMTP</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_SMTPM</td>
<td>10</td>
<td>Microsoft Mail</td>
</tr>
<tr>
<td>P_ADDRESS_PLAN_X400</td>
<td>11</td>
<td>X.400</td>
</tr>
</tbody>
</table>

For the case where the P_ADDRESS_PLAN_NOT_PRESENT is indicated, the rest of the information in the TpAddress is not valid.

**TpAddressError**

Defines the reasons why an address is invalid.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ADDRESS_INVALID_UNDEFINED</td>
<td>0</td>
<td>Undefined error</td>
</tr>
<tr>
<td>P_ADDRESS_INVALID_MISSING</td>
<td>1</td>
<td>Mandatory address not present</td>
</tr>
<tr>
<td>P_ADDRESS_INVALID_MISSING_ELEMENT</td>
<td>2</td>
<td>Mandatory address element not present</td>
</tr>
<tr>
<td>P_ADDRESS_INVALID_OUT_OF_RANGE</td>
<td>3</td>
<td>Address is outside of the valid range</td>
</tr>
<tr>
<td>P_ADDRESS_INVALID_INCOMPLETE</td>
<td>4</td>
<td>Address is incomplete</td>
</tr>
<tr>
<td>P_ADDRESS_INVALID_CANNOT_DECIDE</td>
<td>5</td>
<td>Address cannot be decoded</td>
</tr>
</tbody>
</table>
TpAddressRange
This type is identical to TpAddress with the difference that the AddrString can contain wildcards. Two wildcards are allowed: * which matches zero or more characters and ? which matches exactly one character. The wildcards are only allowed at the end or at the beginning of the AddrString.
Some examples for E164 addresses:
- "123" matches specifies number.
- "123*" matches all numbers starting with 123 (including 123 itself)
- "123???" matches all numbers starting with 123 and at least 5 digits long
- "123??*" matches all numbers starting with 123 and exactly 6 digits long

For e-mail style addresses, the wildcards are allowed at the beginning of the AddrString:
- "*@parlay.org" matches all email addresses in the parlay.org domain

The following address ranges are illegal:
- 1?3
- 1*3
- ?123*

Legal occurrences of the ‘*’ and ‘?’ characters in AddrString should be escaped by a ‘\’ character. To specify a ‘\ character ‘\’ must be used.

TpURL
This data type is identical to a TpString and contains a URL address. The usage of this type is distinct from TpAddress, which can also hold a URL. The latter contains a user address which can be specified in many ways: IP, e-mail, URL etc. On the other hand, the TpURL type does not hold the address of a user and always represents a URL. This type is used in user interaction and defines the URL of the test or stream to be sent to an end-user. It is therefore inappropriate to use a general address here.

TpCallNotificationInfo
Defines the Sequence of Data Elements that specify the information returned to the application in a Call notification report.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallNotificationReportScope</td>
<td>TpCallNotificationReportScope</td>
<td>Defines the scope of the notification report.</td>
</tr>
<tr>
<td>CallAppInfo</td>
<td>TpCallAppInfoSet</td>
<td>Contains additional call info.</td>
</tr>
<tr>
<td>CallEventInfo</td>
<td>TpCallEventInfo</td>
<td>Contains the event which is reported.</td>
</tr>
</tbody>
</table>

TpCallEventInfo
Defines the Sequence of Data Elements that specify the event report specific information.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
<th>Sequence Element Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallEventType</td>
<td>TpCallEventType</td>
<td>AdditionalCallEventInfo</td>
<td>TpCallAdditionalEventInfo</td>
<td>CallMonitorMode</td>
<td>TpCallMonitorMode</td>
<td>CallEventTime</td>
<td>TpDateAndTime</td>
<td></td>
</tr>
</tbody>
</table>

TpCallEventRequestSet
Defines a Numbered Set of Data Elements of TpCallEventRequest.
**TpCallEventRequest**
Defines the Sequence of Data Elements that specify the criteria relating to call report requests.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallEventType</td>
<td>TpCallEventType</td>
</tr>
<tr>
<td>AdditionalCallEventCriteria</td>
<td>TpAdditionalCallEventCriteria</td>
</tr>
<tr>
<td>CallMonitorMode</td>
<td>TpCallMonitorMode</td>
</tr>
</tbody>
</table>

**TpCallNotificationRequest**
Defines the Sequence of Data Elements that specify the criteria for an event notification.

<table>
<thead>
<tr>
<th>Sequence Element Name</th>
<th>Sequence Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallNotificationScope</td>
<td>TpCallNotificationScope</td>
<td>Defines the scope of the notification request.</td>
</tr>
<tr>
<td>CallEventsRequested</td>
<td>TpCallEventRequestSet</td>
<td>Defines the events which are requested</td>
</tr>
</tbody>
</table>

**TpCallMonitorMode**

<table>
<thead>
<tr>
<th>Field Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
<td>P_CALL_MONITOR_MODE_DO_NOT_MONITOR</td>
</tr>
<tr>
<td>static int</td>
<td>P_CALL_MONITOR_MODE_INTERRUPT</td>
</tr>
<tr>
<td>static int</td>
<td>P_CALL_MONITOR_MODE_NOTIFY</td>
</tr>
<tr>
<td>static TpCallMonitorMode</td>
<td>P_CALL_MONITOR_MODE_DO_NOT_MONITOR</td>
</tr>
<tr>
<td>static TpCallMonitorMode</td>
<td>P_CALL_MONITOR_MODE_INTERRUPT</td>
</tr>
<tr>
<td>static TpCallMonitorMode</td>
<td>P_CALL_MONITOR_MODE_NOTIFY</td>
</tr>
</tbody>
</table>

**TpCallEventType**
Defines a specific call event report type.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_CALL_EVENT_UNDEFINED</td>
<td>0</td>
<td>Undefined</td>
</tr>
<tr>
<td>P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT</td>
<td>1</td>
<td>An originating call attempt takes place (e.g. Off-hook event).</td>
</tr>
<tr>
<td>P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED</td>
<td>2</td>
<td>An originating call attempt is authorised</td>
</tr>
<tr>
<td>P_CALL_EVENT_ADDRESS_COLLECTED</td>
<td>3</td>
<td>The destination address has been collected.</td>
</tr>
<tr>
<td>P_CALL_EVENT_ADDRESS_ANALYSED</td>
<td>4</td>
<td>The destination address has been analysed.</td>
</tr>
<tr>
<td>P_CALL_EVENT_ORIGINATING_SERVICE_CODE</td>
<td>5</td>
<td>Mid-call originating service code received.</td>
</tr>
<tr>
<td>P_CALL_EVENT_ORIGINATING_RELEASE</td>
<td>6</td>
<td>A originating call leg is released</td>
</tr>
<tr>
<td>P_CALL_EVENT_TERMINATING_CALL_ATTEMPT</td>
<td>7</td>
<td>A terminating call attempt takes place</td>
</tr>
<tr>
<td>P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED</td>
<td>8</td>
<td>A terminating call is authorized</td>
</tr>
<tr>
<td>P_CALL_EVENT_ALERTING</td>
<td>9</td>
<td>Call is alerting at the call party.</td>
</tr>
<tr>
<td>P_CALL_EVENT_ANSWER</td>
<td>10</td>
<td>Call answered at address.</td>
</tr>
<tr>
<td>P_CALL_EVENT_TERMINATING_RELEASE</td>
<td>11</td>
<td>A terminating call leg is released or the call could not be routed.</td>
</tr>
<tr>
<td>P_CALL_EVENT_REDIRECTED</td>
<td>12</td>
<td>Call redirected to new address: an indication from the network that the call has been redirected to a new address (no events disarmed as a result of this).</td>
</tr>
<tr>
<td>P_CALL_EVENT_TERMINATING_SERVICE_CODE</td>
<td>13</td>
<td>Mid call terminating service code received.</td>
</tr>
<tr>
<td>P_CALL_EVENT_QUEDUED</td>
<td>14</td>
<td>The Call Event has been queued. (no events are disarmed as a result of this)</td>
</tr>
</tbody>
</table>
Appendix 6 Relevant Parlay Routing Methods

MULTI-PARTY CALL CONTROL SERVICE INTERFACE. The multi-party call interface class represents the interface to the multi-party call Service Capability Feature. It provides a structure to allow simple and complex call behaviour.

createAndRouteCallLegReq (callSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet, targetAddress : in TpAddress, originatingAddress : in TpAddress, applInfo : in TpCallApplInfoSet, appLegInterface : in IpAppCallLegRef) : TpCallLegIdentifier

This method is an asynchronous method used to request the creation of a new Call Leg and the setup of a connection to the indicated address.

CALL LEG SERVICE INTERFACE. The call manager interface class provides the management functions to the multi-party call Service Capability Features. The application programmer can use this interface to create call objects and to enable or disable call-related event notifications.


This method is an asynchronous method used to request routing of the call leg to the remote party indicated by the target address.

getLastRedirectedAddress (callLegSessionID : in TpSessionID) : TpAddress

This method is sent by the application to the leg to get the last address the leg has been redirected to.

release (callSessionID : in TpSessionID, cause : in TpReleaseCause) : void

This method requests the release of the call object and associated objects. The call will also be terminated in the network. If the application requested reports to be sent at the end of the call (e.g., by means of getlnfoReq) these reports will still be sent to the application.

Deassign (callLegSessionID : in TpSessionID) : void

This method requests that the relationship between the application and the call leg and associated objects be de-assigned. It leaves the call leg in progress, however, it purges the specified call leg object so that the application has no further control of call leg processing. If a call leg is de-assigned that has event reports or call leg information reports requested, then these reports will be disabled and any related information discarded.

The application should not release or deassign the call leg when received a callLegEnded() or callEnded(). This operation continues processing of the call leg.
attachMedia (callLegSessionID : in TpSessionID) : void

This method requests that the call leg be attached to its call object. This will allow transmission on all associated bearer connections or media streams to and from other parties in the call. The call leg must be in the connected state for this method to complete successfully.
Appendix 7  SIP Forking

What is Forking?

The user can own several SIP-enabled devices like a number of fixed (wireline) and / or mobile phones and one or more PC’s. The user can register all those addresses by issuing a single REGISTER message with multiple “contact” headers or by registering each address separately. The S-CSCF has an access to location services and can store those informations probably in the HSS or somewhere else in a database. The S-CSCF can play the role of SIP Registrar server and SIP Redirect server as well. If the S-CSCF in the home network assigned to that user receives a call request with a destination address corresponding to one of those registered by this specific user, it will fork the request to all possible locations retrieved from the database (SIP Location server). Forking can still occur downstream for every forked request. The decision whether to use parallel or sequential forking is up to the server configuration and hence to the network operator but can be also specified by the application in the routing request. A search type called from within a service can be specified by the service itself. It is also very likely that the parallel en / or sequential search functionality will become one of the end-user services. Then the following scenario will apply: ask the network to fork and report results. Depending on the need of the application the SCS can return one or more addresses to the application or it will report an error: no user could be contacted.

The application, SIP Proxy or UA itself can also decide to initiate the fork independent of the network events, when running a service. This can result in a parallel forking which can be graphically presented as a “star”. This situation is specific for conference or multiparty calls. In this case the application again can entirely control the fork or send a list of contacts and ask the forking as a service from the network.

When the forking can occur in the SIP network?

- user has registered multiple addresses
- looking for the server supporting required SIP extension (OPTION forking)
- user uploads (updates) a CPL script to multiple servers where the script (service) should run.
- splitting the media stream for tracking / recording purposes
- forking of SUBSCRIBE request (here a UA issuing SUBSCRIBE can receive NOTIFY requests with the FROM field that differs from the TO field from the SUBSCRIBE response, because the subscription can be accepted by multiple nodes. Whether the SUBSCRIBE request may be forked or not is specified in the event package. It should also specify whether merging of multiple NOTIFY is required to form one single state notification)
- “workgroup call pickup”

How to distinguish multiple responses of the forked-message?

In SIP network when request has been forked and multiple 200 OK responses arrive at the UAC or the forking proxy, each response is distinguished by the tag parameter in the To header field, and each represents a distinct dialog, with a distinct dialog identifier. Every proxy must also insert a Via header field into the copy before the existing Via header fields. When forking the Via header MUST include a “branch” parameter When the path of a request through one or more forking proxies is graphed, the result is a tree. The branch parameter identifies the “branch” each request was forwarded on. The branch parameter value MUST be unique for each client transaction to which the request is forwarded (check rfc2543bis-05 for details on branch parameter formatting).
How to restrict forking in the network?

The Request-Disposition header field specifies caller preferences for how a proxy or redirect server should process a request. It is not processed by user agents. Its value is a list of tokens, each of which specifies a particular feature. When the caller specifies a feature, the server SHOULD treat it as a hint, not as a requirement and MAY ignore the feature request. The header field has the following syntax:

```
Request-Disposition = ( "Request-Disposition" | "d" ) ;"1# (proxy-feature | cancel-feature | fork-feature | recurse-feature | parallel-feature | queue-feature)
```

- **proxy-feature:** This feature indicates whether the caller would like each server to proxy or redirect. If the server is incapable of performing the requested feature, it SHOULD ignore the feature request.
- **cancel-feature:** This feature indicates whether the caller would like each proxy server to send a CANCEL request downstream in response to a 200 OK from the downstream server (which is the normal mode of operation, making it somewhat redundant), or whether this function should be left to the caller. If a proxy receives a request with this parameter set to "no-cancel", it SHOULD NOT CANCEL any outstanding branches on receipt of a 2xx. However, it would still send CANCEL on any outstanding branches on receipt of a 6xx.
- **fork-feature:** This feature indicates whether a proxy should fork a request, or proxy to only a single address. If the server is requested not to fork, the server SHOULD proxy the request to the "best" address (generally the one with the highest q value). The feature is ignored if "redirect" has been requested.
- **recurse-feature:** This feature indicates whether a proxy server receiving a 300-class response should send requests to the addresses listed in the response (i.e., recurse), or forward the list of addresses upstream towards the caller. The feature is ignored if "redirect" has been requested.
- **parallel-feature:** For a forking proxy server, this feature indicates whether the caller would like the proxy server to proxy the request to all known addresses at once, or go through them sequentially, contacting the next address only after it has received a non-200 or non-600 final response for the previous one. The feature is ignored if "redirect" has been requested.
- **queue-feature:** If the called party is temporarily unreachable, e.g., because it is in another call, the caller can indicate that it wants to have its call queued rather than rejected immediately. If the call is queued, the server returns "182 Queued". Example: Request-Disposition: proxy, recurse, parallel

**Request-Disposition Processing**

If the request contains a Request-Disposition header, the server SHOULD execute the behaviors described by the tokens, unless it has local policy configured to direct it otherwise.

**Interactions with CPL**

When the called party has a Call Processing Language (CPL) script present, feature interactions are introduced. CPL addresses this by allowing the CPL script to control whether caller preferences are applied to the location list or not. CPL also allows the called party to discard certain rules from the caller preferences before their application.
How to report forking to the call originating party?

The INVITE is redirected

If the UAS decides to redirect the call, a 3xx response is sent. A 300 (Multiple Choices), 301 (Moved Permanently) or 302 (Moved Temporarily) response SHOULD contain a Contact header field containing URIs of new addresses to be tried. The response is passed to the INVITE server transaction, which will deal with its retransmissions.

**Response 300 Multiple Choices**

The address in the request resolved to several choices, each with its own specific location, and the user (or user agent) can select a preferred communication end point and redirect its request to that location.

The response MAY include a message body containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate, if allowed by the Accept request header.

The choices SHOULD also be listed as Contact fields (Section 22.10). Unlike HTTP, the SIP response MAY contain several Contact fields or a list of addresses in a Contact field. User agents MAY use the Contact header field value for automatic redirection or MAY ask the user to confirm a choice. However, this specification does not define any standard for such automatic selection.

This status response is appropriate if the callee can be reached at several different locations and the server cannot or prefers not to proxy the request.

**This method could be used to let SCS know how many branches are created.**
Appendix 8  Related SIP Methods

NEGOTIATE

There is a need to negotiate a multitude of parameters, settings, and algorithms when setting up sessions using Session Initiated Protocol (SIP). While SIP itself provides mechanisms for negotiation of these parameters on a per-session basis through the use of the INVITE method, it does not provide a ready mechanism for meta-session negotiation. The closest mechanism provided is the REGISTER method, however, this method is directed towards the registrar alone, and cannot be used to conduct negotiation of parameters between any two arbitrary SIP nodes.

Examples of parameters that may need to be negotiated (and thus, the ready impetus for providing a simple mechanism to handle this) include: compression algorithms, code book size, message integrity mechanisms, encryption algorithms, etc. Many of these parameters are not always eligible for use in an INVITE method, for two reasons:

- The INVITE method describes the parameters for initiation of a particular session. Once the session is over, the negotiated settings (such as the RTP profile used, in the case of SDP) are invalidated for future re-use. It would be inefficient to have to renegotiate parameters that are the same from session to session, or have to transmit large quantities of persistent data (such as a code book) each time.
- Many meta-session applications (and therefore their attendant negotiable parameters) are best utilized if they can be applied for the first message of a session. An example of this would be header compression. If the INVITE were compressed, then the header that identifies the type of compression in use would also be compressed, and therefore unintelligible (assuming no shim mechanism). This document seeks to solve these problems by introducing a SIP extension that allows for meta-session parameters to be negotiated in a generic manner. This negotiation would take place prior to session establishment, between any two SIP entities (User Agents, Proxies etc.).

The process of registration entails sending a REGISTER message to a special type of UAS known as a registrar. The registrar acts as a front end to the location service for a domain, reading and writing mappings based on the contents of the REGISTER messages. This location service will then be consulted by a proxy server that is responsible for routing requests for that domain.

SIP does not mandate a particular mechanism for implementing the location service. The only requirement is that a registrar for some domain MUST be capable of reading and writing data to the location service, and a proxy for that domain MUST be capable of reading that same data. A registrar MAY be co-located with a particular SIP proxy server for the same domain, allowing usage of an in memory database for the location service. Usage of a shared database is another implementation choice.

OPTIONS

The SIP method OPTIONS allows a client to query another client or server as to its capabilities. This allows a client to discover information about the methods, content types, extensions, codecs etc. supported without actually "ringing" the other party. For example, before a client inserts a Require header field into an INVITE listing an option that it is not certain the destination UAS supports, the client can query the destination UAS with an OPTIONS to see if this option is returned in a Supported header field.
Example OPTIONS request:

OPTIONS sip:carol@chicago.com SIP/2.0
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=23411513a6
Via: SIP/2.0/UDP 10.1.3.3:5060
To: <sip:carol@chicago.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 63104 OPTIONS
Contact: <sip:alice@10.1.3.3>
Accept: application/sdp
Contact-Length: 0

Example OPTIONS response:

SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=23411513a6
Via: SIP/2.0/UDP 10.1.3.3:5060
To: <sip:carol@chicago.com>;tag=93810874
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 63104 OPTIONS
Contact: <sip:carol@10.3.6.6>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE
Accept: application/sdp
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
Content-Type: application/sdp
Contact-Length: 274

v=0
o=carol 28908764872 28908764872 IN IP4 10.3.6.6
s=
u=0
i=0
m=audio 0 RTP/AVP 0 1 3 99
a=rtpmap:0 PCMU/8000
a=rtpmap:1 1016/8000
a=rtpmap:3 GSM/8000
a=rtpmap:99 SX7300/8000
m=video 0 RTP/AVP 31 34
a=rtpmap:31 H261/90000
a=rtpmap:34 H263/90000

OPTIONS request can be used to determine the basic state of a UAS, which can be an indication of whether the UAC will accept an INVITE request.

Note that this use of OPTIONS has limitations due to the differences in proxy handling of OPTIONS and INVITE requests. While a forked INVITE can result in multiple 200 OK responses being returned, a forked OPTIONS will only result in a single 200 OK response, since it is treated by proxies using the non-INVITE handling.
Appendix 9  SIP Loop Detection

If the request has no tag in the To the UAS core checks ongoing transactions. If the To, From, Call-ID, CSeq exactly match (including tags) those of any request received previously, but the branch-ID in the topmost Via is different from those received previously, the UAS core SHOULD generate a 482 (Loop detected) response and pass it to the server transaction.

The same request that was generated by the UAC has arrived to the UAS more than once following different paths. The UAS processes the request that was received first and responds with 482 (Loop detected) to the rest of them.

The proxy MUST include a "branch" parameter (Section 22.40) in the Via header. When the path of a request through one or more forking proxies is graphed, the result is a tree. The branch parameter identifies the "branch" each request was forwarded on. The branch parameter value MUST be unique for each client transaction to which the request is forwarded. The precise format of the branch token is implementation-defined. In order to be able to both detect loops and associate responses with the corresponding request, the parameter SHOULD consist of two parts separable by the implementation. The first part is used to detect loops and distinguish loops from spirals. The second is used to match responses to requests. Loop detection is performed by verifying that those fields having an impact on the routing decision have not changed. The value placed in the this part of the branch parameter SHOULD reflect all of those fields (which include any Proxy-Require and Proxy-Authorization headers). This is to ensure that if the request is routed back to the proxy, and one of those fields changes, it is treated as a spiral and not a loop (Section 3). A common way to create this value is to compute a cryptographic hash of the To, From, Call-ID header fields, the Request-URI of the request received (before translation) and the sequence number from the CSeq header field, in addition to any Proxy-Require and Proxy-Authorization fields that may be present. The algorithm used to compute the hash is implementation-dependent, but MD5 [23], expressed in hexadecimal, is a reasonable choice. (Note that base64 is not permissible for a token.) In order to correctly match responses to requests (Section 17.1.3), the value SHOULD also contain a part that is a globally unique function of of the branch on which this request will be forwarded. One example is a hash of a sequence number, local IP address and request-URI of the request. For example: 7a83e5750418bce23d5106b4c06cc632.1 The "branch" parameter MUST depend on all information used for routing decisions, including the incoming request-URI and any header values affecting the routing choices. This is necessary to distinguish looped requests from requests whose routing parameters have changed before returning to this server. Note that the request method MUST NOT be included in the calculation of the branch parameter. In particular, CANCEL and ACK requests MUST have the same branch value as the corresponding request they cancel or acknowledge. The branch parameter is used in correlating those requests at server handling them (see Section 17.2.3 and 9.2 in SIP RFC 2543bis).
Appendix 10  3GPP General Objectives

Terminal capabilities and end-user behavior

- **The capabilities of the terminal** have impact on the SDP description in the SIP session flows, since different terminals may support different media types (such as video, audio, application or data) and may have implemented different set of codecs for audio and video. Note that the capabilities of the terminal may change when an external device, such as a video camera is attached to the terminal.

- **The configuration of the terminal** changes the capabilities of the terminal. This can be done by attaching external devices or possibly by a user setting of certain parameters or profiles in the terminal.

- **The preferences of the destination user** may depend on who is originating the session and on the situation. Cost, associated with the session, may also be another factor, i.e. depending on time of the day or day of the week etc. Due to this reason the user may want to accept or reject certain media components.

- **The available resources in the network** play an important role, as certain media streams, consuming high bandwidth, may be denied. Therefore, before the user is alerted that the session set up is successful, it is assumed that the network has guaranteed and has reserved the needed resources for one or several media streams of the session. This does not preclude the possibility for the user to indicate his/her preferences regarding the session also after the alerting, in which case the initial resource reservations may have to be modified.

- **End-to-end quality of service** may be provided by using a variety of mechanisms, including guaranteed end-to-end QoS and best effort. The network may not be able to guarantee the requested end-to-end QoS. This may be the case when the user is establishing sessions through the public Internet. On the other hand, certain sessions, with the agreement of the initiating and terminating endpoints, should have the right to go through even without having the requested QoS guarantee.

Depending on operator policy, the S-CSCF may forward the SIP request or response to another SIP server located within an ISP domain outside of the IM CN subsystem.
Appendix 11 Call Establishment On-Hold

**Controller originated two party SIP call** (see draft-rosenberg-sip-3pcc-03.txt)

In the following scenario the session establishment is controlled by an controller (which should be seen to be the SCS working in the 3rd Party Controller mode or B2BUA mode) and initially the first SIP callLeg is set on-hold because of absence of the SDP session description in the INVITE message. The controller receives the response of the terminal with contained media descriptions and sends re-INVITE message to the other party. Note that SDP A1 may vary from the final SDP A2 sent by the UA of user A.

![Call Establishment On-Hold Diagram](image)

**Figure 41.** SIP call controlled by a 3rd party controller

This solution has two major advantages:

- It is not required that the controller knows the media that will be used by the participants.
- The first device receiving the second INVITE can perform an intersection of its own SDP (sent in the first message) and the SDP obtained from the other end-point.
Appendix 12  Wireless Application Protocol (WAP)

1. Summary

Service creation in mobile environment has always been rather complicated and time-consuming task. As soon as web services on the Internet became successful the search for methods to implement similar services in the wireless network has started. This brought up the Wireless Application Protocol (WAP) which introduced the concept of the Internet as a service platform. WAP is a result of continuous work to define an industry wide specification for developing applications that operate over wireless communication networks. It enables the operators and manufacturers to meet the challenges in advanced and differentiated service creation on the wireless market. In this section we discuss WAP with special focus on the Wireless Telephony Application (WTA) UA and WTA Interface (WTAI) components.

2. Introduction

The Internet has proven to be an easy and efficient way of delivering services to millions of users in the fixed networks. It was quickly realized, that the opportunity of creating wireless services on a global basis will attract networks operators as well as third party service providers. To avoid incompatibilities in the service delivery methods and enable successful co-operation between companies, in December 1997 WAP Forum has been found. Ericsson, Motorola, Nokia and Unwired Planet took the initiative to start a rapid creation of a standard for making advanced services. The first WAP 1.0 specifications have been released in April 1998 followed up by other releases. The latest version of WAP 2.0, published in August 2001, integrates Internet protocol stack including TCP, TLS and HTTP and provides support for high-speed air interface technologies like General Packet Radio Service (GPRS) and 3rd Generation (3G) cellular.

New WAP also provides a rich application environment that enables delivery of information and interactive services to all types of wireless devices. WAP 2.0 is compatible with WAP 1.0 and comes with new version of Wireless Markup Language (WML2). WML2 is based on eXtensible HyperText Markup Language (XML) and supports Cascading Style Sheets (CSS) for elements positioning. This section concentrates on the Wireless Telephony Application (WTA) framework and the WTA user-agent.
Figure 42. WAP Protocol Stacks

WAP Application Environment (WAE) also called WAP Browser offers the following features:

- **Wireless Telephony Application.** An application framework for telephony services.
- **WAP Push.** A mechanism to send the content to WAP terminal.
- **User Agent Profile (UAProf).** Defines the terminal capabilities and user preferences.
- **External Functionality Interface (EFI).** Extends WAE with external functionalities.
- **Persistent Storage Interface.** Enables storage of the services on the terminal.
- **Data Synchronization.**
- **Multimedia Messaging Service (MMS).** Delivery of different type of content to the terminal.
- **Provisioning.** Provides WAP client with necessary network information.
- **Pictogram.** This service permits the use of tiny images.

The WTA user-agent is an extension to the standard WML user-agent with the addition of capabilities for interfacing with mobile network services available to a mobile telephony device, e.g. setting up and receiving phone calls.
The WTA framework extends WAE framework by adding the following features:

- **Wireless Telephony Application Interface (WTAI)**. An interface from WML and WMLScript to a specific set of local, telephony related, functions in the WAP client.
- **Network event handling**. This means that events originating from the mobile network could be detected by the WTA user-agent and actions in response to the events could be defined.
- **Repository**, which is a storage container, used by the WTA user-agent, that persistently stores content that executes WTA services. The purpose of the repository is to fulfill the real-time requirements that are placed on the execution of WTA services.
- **WTA service indication**. Provides a means to notify a client that an external asynchronous event has occurred and indicate the service that can be loaded in order to react to that event.

### 3. Architecture

A WAP terminal enhanced with WTA functionality can be viewed as a fully integrated Internet and voice service platform. WTA User Agent (WTA UA) runs the service referenced by the URI that is stored locally or obtained from the WTA Server. The services can be also initiated after receiving an event from the network or by accepting Service Indication (SI) message send from the server on IP domain (Push mechanism). The figure below presents the WTA UA communication with the WTA Server and possible methods to start executing the service.

![WTA Service Architecture and service invocation methods](image)

**Figure 43.** WTA Service Architecture and service invocation methods

1a – Access to the URI via Repository  
1b – Access to the URI via WTA Server  
2 – Receiving the URI via SI (Push)  
3 – Network event transformed by the WTAI onto the WTA event
The WTA server can be thought of as a web server delivering content requested by a client. Like an Internet web browser, a WTA user-agent uses URLs to reference content on the WTA server. A URL can also be used to reference an application on a web server (e.g. a CGI script) that is executed when it is referenced. Such applications can be programmed to perform a wide range of tasks, for example generate dynamic content and interact with external entities. By referencing applications on a WTA server it is possible to create services that use URLs to interact with the mobile network (e.g. an IN-node) and other entities like a voice mail system. The access to the extended telephony-related functionality of the IN-nodes can be controlled by the application running on the WTA Server and sending the service invitation towards the WTA UA on the terminal. Thus, the concept of referencing applications on a WTA server provides a simple but yet powerful model for how to seamlessly integrate services in e.g. the mobile network with services executing locally in the WAP client.

A WTA service can invoke WTAl-functions that enable access to local functions in the mobile client. Since such functions make it possible to set up calls and access the users local phonebook, it must be ensured that only authorized WTA services are allowed to execute.

WTA services are separated from common WAE services by using predefined port numbers by the WAP gateway and can be accessed only during a WTA session. WTA session is a Wireless Datagram Protocol (WDP) session that uses secure dedicated WDP port number. WTA UA is allowed to retrieve content only from WTA Service Provider, which has been approved for access to a trusted gateway. The WAP gateway verifies that the providers of WTA content are authorized as depicted in the figure below.

![Figure 44. The task of the WAP Gateway](image)

A WTA session can be started only from the terminal by requesting a new WSP session and establishing a Context between the WAP Gateway and the WTA UA. A WTA user-agent can have one or many WTA sessions simultaneously, e.g. one WTA session can be used for service execution and another session can be used to receive pushed Service Indications (SI). However, there is only one common context (a event and state binding between currently executing application on WTA UA and the WTA Server) allowed between the WAP gateway and the WTA user-agent. A new WTA context is initialized whenever the WTA user-agent is started. The context is used to manage the user-agent’s various states.
WAP 2.0 based communication does not require a WAP proxy to translate between the protocols because new version of Wireless Markup Language has built-in support for HTTP version 1.1. Proxy however can optimize the communications process between the WAP Client and Web Server and may offer mobile service enhancements, such as location, privacy, and presence based services. In addition, a WAP proxy is necessary to offer Push functionality and plays a similar role as HTTP proxy but with mobile network-based optimization. Every time the Push is about to being initialized the Push Initiator (PI) can query the WAP proxy for terminal specific information that are cached by the proxy every time the terminal establishes a WAP session.

The WTA framework supports Wireless Telephony Applications that interface with the in-device telephony related functions and the network telephony infrastructure. The internal architecture of WAP client in relation to WTA is presented below.

As already indicated Service Indications (SI) are used to enable the WTA server to notify the end-user about new content to be retrieved from the WTA server. A Service Indication may relate to messaging applications (such as voice mail or e-mail), but also to events in the mobile network.
4. WTAI Capabilities

In addition to the new application environment and the increased capability of the micro-browser, WAP 2.0 also supports other features to improve the user experience. These features expand the capabilities of the wireless devices and improve the ability to deliver useful applications and services. The WTA user-agent is an extension to the standard WML user-agent with the addition of capabilities for interfacing with mobile network services available to a mobile telephony device, like setting up and receiving phone calls.

The specifics of the Wireless Telephony Applications are introduced in the form of an interface. The WAP WTA Interface (WTAI) features provide the means to create Telephony Applications using a WTA UA with the appropriate WTAI functions libraries.

The WTAI features are partitioned into a collection of WTAI function libraries. The type of function and its availability determines where the different functions are specified. The WTAI function libraries are accessible from WMLScript using the scripting function libraries. Some WTAI functions are also accessible from WML using URIs. These functions may initiate an interaction between the mobile and the network. WTAI functions are accessed using the WTAI URI scheme and follow CGI-like invocation mechanism. WTAI functions pass all parameters as type string.

WTA Interface is divided into three categories depending on type of function:

- **Common Network Functions.** The most common features that are available in all networks. They are only accessible from the WTA user-agent. Examples of functions are call setup and answer incoming call.
- **Network Specific Functions.** Features that are only available in certain types of networks. Operator-specific features may also reside in this set.
- **Public Functions.** Simple features that are available to third party applications executing using the standard WAE user-agent. These functions support only simple telephony functions like setting up the Calls.

Only the Network Common and the Public Functions are defined in the WTAI specification and those are described below. All functions have identities within each library and are associated with a certain set of supported permission types and events that may occur as a direct result of function invocation.

As previously mentioned the WTAI library functions are accessible from WMLScript and some of them as well from WML using dedicated WTAI URI encoding scheme. From WMLScript a specific function is called within WMLScript code by referencing WTAI function library together with the actual function name and parameters like:

```
WTAVoiceCall.setup(number, mode)
```

Then the service developer can write a WMLScript function and call it from within WML like presented below. This WML Card allows the mobile user to order the desired type of food.

```
<wml:card>
  <go href="myScript.wmls#CallFood('${foodNumber}')"/>
  <select wml:name="foodNumber">
    <option value="5556789">Pizza</option>
    <option value="5553344">Sandwich</option>
  </select>
</wml:card>
```
The *myScript.wmls* includes then a direct call to some Library Function like
*WTAVoiceCall.setup(12345678, true)*, which will be executed by the WTA Server after receiving
appropriate event from the terminal.

Within WML code the WTAI URI library identifier can be used to identify the library. WTAI
functions are named using URI's. URI follows the following format:

```
wtai://library/function;parameter!result
```

where:

- library - The function type.
- function - Function identifier
- parameter - Function parameters
- result - An optional name of the variable that will be set in the
  WTA UA context as a result of the function call.

Then invocation of WTA function from URI would look like:

```
<wml:card>
  <a href="wtai://wp/rnc;5551212">Call Now</a>
</wml:card>
```

WTA services can place, receive, and terminate voice calls. WTA UA may support multiple
simultaneous voice calls or the service can be limited to only one voice call at a time. WTA
Interface is also capable of detecting network events end information and presenting them to the
WTA UA like specific information about incoming voice calls. Each information field has a name
and value. A field value may be retrieved using its field name. The following fields are defined:

- Number. A phone-number of the called / calling party
- Status. Indicates the recent state of the call.
- Mode. Indicates the coupling between the established call and the current WTA
  context. *DROP* indicates that the call state is tightly coupled with the WTA context
  and *KEEP* indicates no coupling at all.
- Name. The name of the other party.
- Duration. Specifies duration of the call in seconds.

### Public WTAI Library

<table>
<thead>
<tr>
<th>Library Function</th>
<th>URI Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTAPublic</td>
<td><em>wtai://mc;number!/result</em></td>
</tr>
<tr>
<td>WTA Events</td>
<td></td>
</tr>
</tbody>
</table>

There are no WTA events associated with this function library

### WMLScript Functions

| MakeCall(number)      | wtai://wp/mc;number!/result |
| sendDTMF(dtmf)        | wtai://wp/sd;dtmf!/result   |
| addPBEntry(number, name) | wtai://wp/ap;number;name!/result |
### Network Common WTA! Libraries

<table>
<thead>
<tr>
<th>Library Function</th>
<th>WTA Events</th>
<th>WMLScript Functions</th>
<th>URI Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTAVoiceCall</td>
<td>IncomingCall, CallCleared, CallConnected, OutgoingCall, CallAlerting, DTMFSent</td>
<td>setup(number, mode)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>accept(callHandle, mode)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>release(callHandle)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sendDTMF(callHandle, dtmf)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>callStatus(callHandle, field)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>list(returnFirst)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Library Function</th>
<th>WTA Events</th>
<th>WMLScript Functions</th>
<th>URI Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTANetText</td>
<td>MassageSendStatus, IncomingMessage</td>
<td>send(address, text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>list(returnFirst, messageType)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>remove(msgHandle)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>getFieldValue(msgHandle, field)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>markAsRead(msgHandle)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Library Function</th>
<th>WTA Events</th>
<th>WMLScript Functions</th>
<th>URI Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTAPhoneBook</td>
<td>There are no WTA events associated with this function library</td>
<td>write(index, number, name)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>search(field, value)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>remove(index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>getFieldValue(index, field)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>change(index, field, newValue)</td>
<td></td>
</tr>
</tbody>
</table>
### WMLScript Functions

**setIndicator(type, newState)**

The `type` parameter is an integer and represents the following events:

- 0 = Incoming Speech Call
- 1 = Incoming Data Call
- 2 = Incoming Fax Call
- 3 = Call Waiting
- 4 = Text Message
- 5 = Voice Mail Message
- 6 = Fax Message
- 7 = Email Message.

**endContext()**

**getProtection()**

**setProtection(mode)**

**URI Functions**

- `wtai://ms/ec`

## 5. WTA Services

The WAP Programming Model, closely aligned with the Web Programming Model, uses the Pull Model, (which is where the client requests content from the server). However, WAP also extends the Web architecture by adding telephony support with WTA and enabling a Push Model, where a server can proactively send content to the client.

In addition to supporting presentation services, similar to HTML, WML adds support for events and variables. This gives developers the tools they need to develop browser-based applications that go beyond simple viewable documents for mobile users. WML’s accompanying client-side scripting language, WMLScript, provides for additional intelligence and control over presentation. To improve the efficiency of transmission and client implementation for the handling of both WML and WMLScript, WAE supports tokenisation of WML1 and the compilation of WMLScript before the gateway sends the content to the device.
WTA services are created using WML2 and WMLScript. The previously used WTA-WML content format is deprecated and should be transformed into WML2. From a WMLScript, telephony functions can be accessed through the Wireless Telephony Applications Interface (WTAI). WTAI also provides access to telephony functions from WML2 by using URIs. URIs form a unifying naming model to identify features independently of the internal structure of the device and the mobile network. The WTA services reside on the WTA server. The client addresses WTA services by using URLs. Examples of WTA services include:

- **Extended set of user options for handling incoming calls (Incoming Call Selection):**
  The service is started when an incoming call is detected in the client. A menu with user options is presented to the user. Examples of options could be:
  - Accept call
  - Redirect to voice mail
  - Redirect to another subscriber
  - Send special message to caller

- **Voice mail:** The user is notified that she has new voice mails, and retrieves a list of them from the server. The list is presented on the client's display. When a certain voice mail has been selected, the server sets up a call to the client and the user listens to the selected voice mail.

- **Call subscriber from message list or log:** When a list of voice, fax or e-mails or any kind of call log is displayed the user has the option of calling the originator of a selected entry in the list or log.

The detailed set of WTA functions is as follows:
- setting up the Calls
- answering or rejecting the Calls
- placing Calls on a hold
- redirecting the Calls
- Sending and receiving network messages and getting information about those messages.
- manipulating the device's Phonebook
- accessing the device's call history. The WTA user agent may provide access to the following call logs:
  - Dialed Call Log. History of outgoing voice calls
  - Missed Call Log. History of incoming voice calls that were not answered
  - Received Call Log. History of incoming voice calls that were answered
- Accessing to the following logical indicators:
  - Incoming Speech Call
  - Incoming Data Call
  - Incoming Fax Call
  - Call Waiting
  - Text Message
  - Voice Mail Message
  - Fax Message
  - Email Message

A logical indicator may be manifested as an LED, as an icon on the display, as a number on the display, or as a unique audible signal.

Repository
The repository is a persistent storage module within the mobile terminal that may be used to eliminate the need for network access when loading and executing frequently used WTA services. The repository also addresses the issue of how a WTA service developer ensures that time-critical WTA events are handled in a timely manner. WTA service developer can pre-program the device with content using the depository and improve the response time for a WTA service.

The repository can be accessed:

- As result of the WTA event generated after receiving the event from the network and being associated with a specific channel
- Directly by end-user via channels menu representing allowable services in case the service has been.
- The SI or content retrieved from the WTA server can contain the URL to the content, which already has been stored in the repository. On receiving such a reference the WTA UA will check if the content indicated in URL has been already retrieved en if it still exists in the repository.

- The repository contains a set of channels and resources.
- Resources are data that have been downloaded with WSP (e.g. a WML2), and are stored along with their meta-data (e.g. content type and the HTTP 1.1 entity-tag) and location (URL).
- A channel is a resource that contains a set of links to resources. Channels have an identity and freshness.

![Diagram of Repository](image)

**Figure 46. WTA Repository**

Channels can be pushed into the repository or loaded into the repository whenever the user-agent retrieves them. Channels are uniquely identified by the channelid attribute value.

The Channel Media Type contains the resource element that specifies a resource that is contained in a channel. The location of the resource is described by means of the URL that is specified in the first resource element in a channel. It identifies the content to be invoked when the channel is referenced by some means.

This is an example of a simple WTA service specified using the WTA Channel.
Channels can be loaded into the repository using any standard content transfer mechanism that is suitable to use with the specific type of network and bearer. Channel download methods may include:

- Returning the channel as part of a response to a standard URL request (GET or POST method).
- Pushing the channel to the device either directly or using a Service Indication (SI).

Figure 7. Successful Channel Installation
6. Standardization body

WAP Forum: [www.wapforum.org](http://www.wapforum.org)

7. Examples

This example illustrates the Incoming Call Selection service

846. Incoming Call Selection service

Here the network receives an incoming call and sends the message to the mobile subscriber. In the terminal the WTAI generates an IncomingMessage event and consults the repository to find a propitious channel associated with this event. The channel provides the URL to the "Incoming Call Selection" service stored in the repository. In this case the repository contains required content to run the service and returns it to the WTA UA.

Then the context is created and code is loaded into that context and service starts executing. The service gives the user possibility to choose between several options which can be:

- Accept the call
- Redirect the call
- Reject the call

Then user accepts the call and the WTAI function "WTAVoiceCall.accept" is invoked. Then WTAI function is invoked to connect to the mobile network, which returns acknowledgement of the connection and then A result code indicating the outcome of the call is generated internally in the phone. A speech path between the mobile network and the client is established.
And the next example illustrates the Push service invocation.

![Diagram of the Push service invocation process]

**Figure 49.** Push service invocation

1. The Voice Mail System notifies the WTA server that there are new voice mails. A list of them is also sent to the WTA server.
2. The WTA server creates new service content based on the list received from the voice mail system. The content is stored on the server, and its URL is included in a Service Indication that will be pushed to the client. The Service Indication’s message could read: “You have 4 new voice mails”.
3. The WAP gateway sends the Service Indication to the client using push.
4. The user is notified about the Service Indication by a message delivered with the Service Indication. The user chooses to accept the Service Indication.
5. A WSP “Get” request is sent to the WAP gateway (URL provided by the Service Indication).
6. The WAP gateway makes a WSP/HTTP conversion.
7. The WTA server returns the earlier created voice mail service.
8. The WAP gateway makes an HTTP/WSP conversion.
9. The voice mail service is now executing in the client. The user is presented with a list of voice mails originating from the Voice Mail System (a WML2 “Select List” created in step 2). The user selects a certain voice mail to listen to.
10. Another WSP "Get" request is sent to the WAP gateway. The requested deck identifies the selected voice mail.
11. The WAP gateway makes a WSP/HTTP conversion.
12. The WTA server returns the requested deck. The deck only contains one card with a single WML2 "go href" task. The URL is automatically called when the card is executed and it refers to a card in the earlier downloaded voice mail content which binds the incoming call event (wtaev-cc/ic) so that the subsequent call from the Voice Mail System will be answered automatically. Now, also the WTA server is informed about which voice mail the user has chosen to retrieve.
13. The WAP gateway makes a HTTP/WSP conversion.
14. The incoming call event (wtaev-cc/ic) is temporarily bound so that the call from the Voice Mail System will be answered automatically. In order to avoid that the voice mail service answers a call from someone else than the voice mail system, the calling party's phone number (callerld parameter of the wtaev-cc/ic event) is preferably checked.
15. The WTA server instructs the Voice Mail System to play the selected voice mail.
16. The Voice Mail System instructs the mobile network to set up a call to the client.
17. The mobile network sets up a call to the client.
18. The client answers the call automatically due to content loaded in steps 12 to 14.
19. The mobile network informs the Voice Mail System that the client has accepted the call.
20. Acknowledgements are sent to the client and the voice mail system.
21. A speech path is established between the Voice Mail System and the client, and the message is played.
Appendix 13  PINT / SPIRITS Service Architectures

1. Summary

Now a day the telephony is a voice centric service in the fixed PSTN / ISDN and mobile GSM networks. Also on the Web, the technologies like Voice over IP (VoIP) make speech-based web services available to the wide range of Internet users. As in the next generation networks the web services are moving towards the UMTS terminals the need for integration of IP and circuit switched networks arise. This section deals with the creation and service architecture of so-called hybrid services that allow inter-networking across IP-based and circuit-switched networks. PINT and SPIRITS are examples of such services. This section presents the following:

- The PINT / SPIRITS service architecture.
- The type of PINT / SPIRITS services.
- PINT / SPIRITS Gateway
- A static and dynamic working scenario.
- Implementation methods

2. Introduction

The service architecture for provisioning speech-based services in the fixed world is called the Intelligent Network (IN) and in wireless world, Customized Application for Mobile Network Enhanced Logic (CAMEL). In IN the Intelligent Network Application Protocol (INAP) and in GSM the Camel Application Protocol (CAP) serve for invocation and handling of traditional telephony services and are used between network entities that communicate using different protocol. This implies necessity for network logic to translate between those different protocols. CAP and INAP in both service architectures relay on ISDN User-Part (ISUP) which is a peer-to-peer call control signaling protocol and provides a similar to SIP functionality for establishing speech-based connections. Also the next generation mobile networks will contain mixed GSM circuit switched and GPRS packet switched networks in its IP Multimedia Subsystem (IP MM) domain.

On the other side the considerable and still growing number of Internet users along with many PSTN clients has created a demand for a new class of services, which can enable interconnection between IP and circuit switched domains and take advantages of both technologies. Successful inter-networking of the Internet and Global Switched Telephone Network (GSTN) should enable integration of PSTN services with those offered by the Internet through the World-Wide Web. Fortunately, in converged fixed and mobile network the service architecture is going to be access and terminal independent because IP will be the transport layer platform for not voice only but for all end-to-end multimedia services. PINT / SPIRITS architecture is meant to be IP-based what would allow both SIP servers and SIP enabled end-user terminals to manage calls in a wide range of networks, including circuit switched networks, without requiring a big changes to existing protocols and network nodes. Some new methods and a gateway on the border between IP and PSTN network has been defined to support PINT and SPIRITS services. It is understood that Intelligent Network systems, private PBXs, cellular phone networks and the ISDN can all be used to deliver PINT services. Also, the request for service might come from within a private IP network that is disconnected from the whole Internet. The PINT / SPIRITS architecture standardization in IETF are driven by Lucent, AT&T, Nortel, Siemens and Telia.
3. PINT

PINT is proposed extension to SIP protocol (also referred to as SIP+) and the work on it is done by PINT WG of IETF. For the IN part of PINT the ITU-T is responsible. SIP Servers and SIP UAs supporting SIP+ extensions are also called PINT clients. PINT architecture enables access to GSTN services from the Internet. PINT services always involve two separate networks: an IP network to request the placement of a call, and a GSTN telephone network to execute the actual call. The PINT Gateway that relays PINT requests to the GSTN appears to the SIP network as a SIP User Agent Server. The PINT Milestone Services include:

- Request to Call
- Request to Fax Content and Request to Fax back
- Request to Speak / Send / Play Content

Despite of using SIP methods in PINT service execution session the media is transported over the telephone system while in a SIP session the data is sent over IP network. There are three new SIP methods proposed in RFC 2848 that would provide the user with some additional information about the status of the call. Those, along with other changes to SIP are listed below.

- **SUBSCRIBE.** This request indicates that a user wishes to receive information about the status of the session (Session Monitoring).
- **UNSUBSCRIBE.** Used to unsubscribe from the service.
- **NOTIFY.** This message will be send during the period a user is subscribed to the service as a result of any change in the status of the service session (Status Indication).
- **Warning Headers.** To signal unsupported PINT features.
- **Require Headers.** To signal the mandatory PINT extensions to SIP towards the PINT Server.
- **Use of user portion of SIP URL to indicate the type of Service.**
  - R2C: Request-to-Call
  - R2F: Request-to-Fax
  - R2HC: Request-to-Hear-Content

A PINT URL can be used within Request-URI or in To and From header in PINT request.

The defined for PINT extensions of SIP affect also Session Description Protocol (SDP) for which SIP is the transport protocol. Here are the most significant changes to SDP:

- New TN (Telephone Network) parameter for SDP connection field.
- New SDP media types: text, image and application.
- New transport protocol keywords along with associated format and attribute types: voice, fax and pager.

The possible PINT interaction with the PSTN network include:

- Create a Call
- Destroy a Call
- Connect a Terminal to a Call
- Disconnect a Terminal to a Call
- Play some file data to a connected Terminal
- Check and Monitor Status of a Request, Call or Terminal
- Send a bill

The PINT Service Protocol is specified in RFC 2848.
3. SPIRITS

The SPIRIT has been developed within SPIRIT Working Group (WG) of IETF and deals with the implementation of services that are initiated in the PSTN towards terminals attached on IP network. Typical examples of SPIRITS services are:

- Internet Call Waiting (ICW)
- Internet Caller-ID Delivery
- Internet Call Forwarding

Those services take advantage of the following SPIRITS functionality:

- Redirect Calls
- Announce a pending Call
- Filter incoming Calls
- Select termination point for incoming Call

In Internet Call Waiting a PC user which is attached to the Internet via a dial-up connection can be notified of an incoming phone call, without the need for an additional phone line. He can also specify on the fly the treatment of the incoming call while being notified as follows:

- Accept the Call
  - Terminate the Internet connection and accept PSTN Call
  - Accept the Call by VoIP
- Reject the Call
- Forward the incoming call to another phone number
- Redirect the incoming Call to Voice Mail
- Play recorded message to the calling party and disconnect incoming Call

Other possibility is to define default action the network will take in case of an incoming Call. In this situation the Call will not be presented to the User but a recorded service processing log file will provide the User with all information about the call date and time and calling party number and can be viewed at any latter time.

Internet Caller-ID Delivery and Internet Call Forwarding are a part of ICW as described above if the User is being called while connected to the Internet over a single telephone line. If it is not the case, in the Internet Call Forwarding service PC will act as an auxiliary device to which incoming call is directed first.

The service subscription to the SPIRITS services can be done over the phone, by posting a request by mail or via the Internet. After this, subscriber will receive the SPIRITS software by post or will download it from the Web. After that the User can register to SPIRITS service session using the web browser and can specify whether the incoming Call should be presented to him / her or handled automatically.

The SCF in the SPIRITS network interacts with the SPIRITS Gateway and PINT Server in IP domain through the SPIRITS Client and can be physically located in a stand-alone general-purpose Service Control Point (SCP) node or in the specialized network element called Service Node (SN). The SCF also controls the switches where the Service Switching Function (SSF) resides. SSF interacts with SCF for recognition and handling of IN services.

The SPIRITS Service Protocol is described in RFC 2995.
3. PINT / SPIRITS gateway

The PINT/SPIRITS gateways could be new innovative bridging products, which may be seen as a natural evolution of the IP Service Switching Function (IPSSF). PINT gateway might have a true telephone interface or it might be connected via some other protocol or API to the Executive System in PSTN. PINT work is more advanced than SPIRITS, however both architectures can be seen as part of a common architecture of IP/PSTN service integration. The general idea behind the architecture is to create services as if all communication was based on IP and all clients and servers were SIP enabled. Because, of course, it's not the case in existing telecommunication networks, a new type of network element, the Service Control Gateways (SCG), has been proposed to hide the true situation from the services. SCGs convert network-specific call control signaling to PINT / SPIRITS SIP-like messages and vice versa.

![Diagram of Service Control Gateway]

**Figure 50.** The Service Control Gateway as interface to SCF / SSF functions

A SCG behaves as a regular SIP User Agent (UA) towards the services and as a network-specific service control node in the network where the call is being set up. SCGs handle protocol conversions, but address translation, such as telephone number to SIP URL, are handled by a regular SIP Servers in order to keep the SCG as simple as possible.

A PINT GW acts like a SIP UA server towards the SIP network (terminating end-point), and as an SSF towards the IN/Camel network. Basically, the PINT GW implements a bridging function that allows SIP client-servers to invoke certain telephone services from an IP network. The other implementation could be to implement the PINT GW, to act as a SCP towards the IN/Camel network, controlling the SSF.

A complementary bridging function is the SPIRITS GW, which acts like a SCP towards the IN/Camel network and as a SIP UA Client towards the SIP network. Alternatively, the SPIRITS GW could be implemented to act as an SSF towards the IN/Camel network.
4. PINT / SPIRITS service network architecture

Inter-networking of the Internet and PSTN, based on open well-defined interfaces, will promote interoperability of both the networks and systems built by different vendors. In the overall PINT / SPIRITS architecture SPIRITS Client, which can be collocated with SCF, receives PSTN requests from SCF and sends back the responses. PINT servers receive PINT requests from the PINT clients residing on the user IP-host and relay them to the PSTN for execution. SPIRITS Server terminates PSTN requests and is responsible for all interactions between Subscriber and SPIRITS Gateway like incoming call notification or specifying the Call treatment. A conventional Number Portability implementation in a mobile Circuit Switched Network (CSN) uses INAP messages to carry number queries to a network-internal database application. One of the INAP messages, that carries the number query, is converted to a SIP INVITE message by the SCG and is then forwarded to the SIP Redirect Server. The next figure depicts the PINT / SPIRITS architecture and the crucial role of the Service Gateway (SG) which also handles so-called Number Portability.

![PINT / SPIRITS service network architecture](image-url)

**Figure 51.** PINT / SPIRITS service network architecture
In the recently proposed architecture SPIRITS Gateway is collocated with the PINT Client as presented below. The figure indicated also the most significant interfaces.

3. How it works?

IN or Soft Switch network node within PSTN domain initiates a request for service towards SPIRIT gateway where SPIRIT Client constructs and issues SPIRIT request and sends it to the SPIRIT Server on the IP domain. This server can have also its mirror on the user terminal, which will usually include a PINT client as well used for registration and initiation of PINT services. The most likely scenario is that a Lite version of the SPIRITS Server will be downloaded in the form of JAVA applet after successful user registration to the SPIRITS services or the appropriate software will be installed on the user machine after subscription. The following figure depicts the registration process to SPIRITS services.

![Diagram of SPIRITS Gateway architecture](image-url)

**Figure 53. Registration to SPIRITS services**

![Registration process](image-url)

**Figure 52. Latest architectural proposals**
Every time the user logs on onto IP domain he / her may register to PINT / SPIRITS Services, the PINT Client will issue then the Subscribe message towards the PINT Gateway. Subscription will end after user logs off or any moment he / her decide to break subscription explicitly. This PINT Call sequence diagram presents the use of SUBSCRIBE and NOTIFY PINT methods.

Figure 54. PINT Call

5. How to implement?

As PINT / SPIRITS services use SIP as signaling protocol, they also use the same service creation methods, which include CPL, CGI and JAVA Servlets as depicted in figure 7. But use of SIP is not mandatory and also others protocols can be used to handle PINT / SPIRITS requests along the network. Due to the high number of available web browsers and servers it seems very likely that some PINT systems will use HTML/HTTP as a front-end for registration and invoking of its services. The PINT request would be transported by means of HTTP request towards the web server associated with a PINT Client. The browsers and web servers on the Internet would be upgraded to support the SIP URLs, which can be of various types like email-like SIP addresses, H323 or E.164 telephone numbers. Then an appropriate HTTP-to-PINT API would be used to convert from HTTP onto PINT request.
6. Conclusions

Both protocols have been developed taking care of, that future extensions to PINT and SPIRITS could develop along with Internet conferencing. Therefore they using SIP to establish the association between participants in the session and SDP to describe the media to be exchanged. However SIP messages are routed based on the Request-URI and To headers which in PINT can include sensitive information. In this case the native SIP encryption which applies only to the body of the message can not protect the information about the user and transport or network layer protection mechanisms should be used. In PINT the REGISTER message is used to register for the service while in SIP it is used for user registration.