

https://dspace.org/

Extended call control telecommunications web service

Vannucci, David Emmanuele

2011-11-02
http://hdl.handle.net/10539/10678

Downloaded from a DSpace repository. https://dspace.org/

EXTENDED CALL CONTROL TELECOMMUNICATIONS WEB SERVICE

David Emmanuele Vannucci

A thesis submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Doctor of Philosophy.

Johannesburg, 2010

Declaration

I declare that this thesis is my own, unaided work, other than where specifically acknowledged. It is being submitted for the degree of Doctor of Philosophy in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

Signed this _____ day of _____ 2010

_____

David Emmanuele Vannucci

Abstract

Internet based call control Web services enable telecommunications network operators to offer Web developers a simplified method of controlling telecommunication resources. Web Services that expose telecommunication networks to third parties are highly abstracted. This abstraction allows Web developers to create applications that provide call control functionality without detailed knowledge of the underlying network. Functionality offered by network operators is usually of a simple nature, and does not provide developers with advanced call control functionality, similar to that found in operator services.

Advanced call control requires the Web application to have detailed knowledge of the state of the telecommunication resources. In this research an Extended Call Control call model and Extended Call Control Web service are developed and demonstrated to provide Web applications with this knowledge.

To develop the Extended Call Control call model existing telecommunication call models were analysed for components suitable for Web control. The Extended Call Control Web service was developed using advanced call control use cases. The proof of concept successfully demonstrates the use of the Extended Call Control Web service and the value of the Extended Call Control call model in proving asynchronous Web based advanced call control of telecommunications resources.

This research has developed a novel call model for Web based call control of telecommunications networks. The Extended Call Control call model and API fulfils a fundamental requirement for Web based advanced call control, namely knowledge of the state of the underlying network and asynchronous control of those resources. This research facilitates the development of advanced Web applications controlling telecommunications calls within the network which previously was limited by the knowledge of the network state. Telecommunication service applications can be moved from tightly coupled systems within the operators network to Web based applications within third party domains such as a Internet based virtual private branch exchange or call centre.

Acknowledgements

I would like to thank the following people:
Prof Hu Hanrahan, my supervisor, for his guidance and support.
Prof Rex Van Olst, head of the Center for Telecommunications Access and Services, for his help and support.
Dr Rolan Christian and Ryan van den Bergh, colleagues, for their advice and encouragement.
Dr Megan Russell, colleague, for her encouragement and support.
Warren Hodgkinson, colleague, for his software and AsteriskNOW advice.

This work was performed in the Wits University Convergence Lab, Centre for Telecommunications Access and Services, at the University of the Witwatersrand, Johannesburg. The Centre is funded by Telkom SA Limited, Nokia-Siemens Networks, Telsaf Data, and the Department of Trade and Industry's THRIP programme. Their financial support is greatly appreciated.

Contents

Declaration i

Abstract

Acknowledgements i

Contents ii

List of Figures vi

List of Tables ix

Acronyms x

Formatting Conventions xiii

1 Introduction 1

1.1 Research objectives 3

1.2 Thesis Outline 5

2 Stateful Web Services 7

2.1 Web services usefulness 8

2.2 Architecture 9

2.3 Stateful Web Services 10

2.4 Synchronous and Asynchronous Stateful Web services 13

2.5 Maintaining state in a Web server 15

2.6 Web Service Resource Framework 16

2.7 Conclusion 19

3 Call Control Call Models 21

3.1 Call Model Background 23

3.1.1 First and Third-Party Call Control 23

3.1.2 Full and Half-Call Models . . . 24
3.1.3 Symmetric and Asymmetric call models . . . 26
3.2 The Intelligent Network . . . 26
3.2.1 IN Service Creation . . . 30
3.2.2 Conclusions . . . 31
3.3 JTAPI . . . 32
3.3.1 Distributed JTAPI . . . 37
3.3.2 Conclusions . . . 38
3.4 JCC/JCAT . . . 39
3.4.1 Java Call Control . . . 40
3.4.2 Relation of JCC/JCAT API to Parlay API . . . 40
3.4.3 JCC Finite state Machines . . . 41
3.4.4 Java Call Control Extensions (JCAT) . . . 45
3.4.5 JCAT Finite State Machines . . . 45
3.4.6 Conclusion . . . 48
3.5 Parlay State Models . . . 48
3.5.1 Generic Call Control . . . 49
3.5.2 Multi-Party Call Control . . . 52
3.5.3 Conclusions . . . 57
3.6 SIP . . . 57
3.6.1 Conclusions . . . 60
3.7 Call Model Mapping . . . 60
3.7.1 OSA/Parlay-IN CS1 Mapping . . . 60
3.7.2 OSA/Parlay-SIP Mapping . . . 64
3.8 Conclusion . . . 66

4 Design of the Extended Call Control Call Model 67
4.1 Extended Call Control Connection Model . . . 67
4.2 The Extended Call Control Call State Model . . . 69
4.3 Methodology to Determine ECC API Methods . . . 72
4.4 ECC Methods . . . 79
4.4.1 Register . . . 79
4.4.2 createCall . . . 80
4.4.3 endCall . . . 80
4.4.4 addCallParticipants . . . 81
4.4.5 moveCallParticipants . . . 81
4.4.6 removeCallParticipants . . . 83

4.4.7 modifyMedia . . . 83
4.5 Transitions . . . 83
4.6 Notification Mapping . . . 84
4.7 Conclusion . . . 85

5 Demonstration of Extended Call Control Call Model 88
5.1 Virtual Private Branch Application . . . 89
5.2 Use Cases . . . 89
5.3 Graphical User Interface . . . 91
5.4 Implementation Architecture . . . 92
5.4.1 Communication . . . 95
5.5 Call Model Mapping . . . 97
5.5.1 Registration . . . 98
5.5.2 Creation of a Call . . . 99
5.5.3 Transferring a Participant . . . 101
5.5.4 Adding a Participant . . . 101
5.5.5 Incoming Call Notification . . . 104
5.5.6 Disconnecting Participants . . . 104
5.5.7 Ending a Call . . . 105
5.6 Discussion . . . 105
5.6.1 Web Service . . . 106
5.6.2 Network Emulator . . . 107
5.6.3 State Manager . . . 107
5.6.4 Application . . . 108
5.6.5 Notification . . . 108
5.6.6 Architecture . . . 108
5.7 Conclusion . . . 109

6 Conclusion 110
6.1 Summary . . . 110
6.1.1 Requirements for Advanced Web Service Call Control . . . 110
6.1.2 Stateful Web Services . . . 112
6.1.3 Existing Call Models . . . 113
6.1.4 Extended Call Control Web Service and Call Model . . . 114
6.1.5 Implementation of Virtual Private Branch Exchange Web Application 114
6.2 Extended Call Control Web Service and Call Model Contribution . . . 115
6.3 Research Limitations . . . 116

6.4 Future Work 117

References **119**

A Papers **125**

B OSA/Parlay Extended Call Control Telecom Web Services **126**

List of Figures

1.1 Generic Click to Dial application (Adapted from (Vannucci and Hanrahan 2005b)) . . . 3
2.1 Service Architecture (Adapted from (Caprio et al. 2004; Caprio and Moiso 2003)) . . . 10
2.2 Web service stack (Adapted from Hogg et al. (2004)) . . . 11
2.3 Synchronous Web service . . . 13
2.4 Asynchronous Web service . . . 13
2.5 Ajax Browser Operation (Adapted from (McCarthy 2005)) . . . 14
3.1 First Party Call Control (Adapted from (Bayer 2000; Graf 2000)) . . . 23
3.2 Third Party Call Control (Adapted from (Bayer 2000; Graf 2000)) . . . 23
3.3 Full Call Model . . . 25
3.4 Half Call Model . . . 25
3.5 Pre IN service processing model (Adapted from (Jain et al. 2005, pg. 66)) . 27
3.6 IN service processing model (Adapted from (Jain et al. 2005, pg. 67)) . . . 27
3.7 IN Service Invocation, from (Hanrahan 2007, pg. 15) . . . 28
3.8 Example Originating BCSM (Adapted from (ITU-T 1993, After Fig. A.2)) . 29
3.9 Example Terminating BCSM (Adapted from (ITU-T 1993, After Fig. A.3)) 31
3.10 JTAPI Call Model . . . 33
3.11 JTAPI Connection Object FSM . . . 34
3.12 JTAPI Call Control extension Connection Object FSM (Adapted from (JTAPI 2002)) . . . 34
3.13 JTAPI State transitions for a two party call (Adapted from (Jain et al. 2005, pg. 42)) . . . 35
3.14 Call model for local two-party call (Adapted from (Graf 2000; Jain et al. 2005)) . . . 36
3.15 Finite State Machines for Call, Connection and Terminal Connection objects (Adapted from (Graf 2000)) . . . 36
3.16 Call model for two-party call with remote addresses and a full call model . 37
3.17 Call model for two-party call with remote addresses and a half call model . 38

3.18 Relationship of JCC package to Parlay (Adapted from (Sun Microsystems 2002)) . . . 40
3.19 JCC Basic Call State Model (Adapted from (Sun Microsystems 2002)) . . . 41
3.20 JCC Provider Finite State Machine (Adapted from (Sun Microsystems 2002)) 42
3.21 JCC Call Finite State Machine (Adapted from (Sun Microsystems 2002)) . 43
3.22 JCC Connection Finite State Machine (Adapted from (Sun Microsystems 2002)) . . . 44
3.23 JCC/JCAT Basic Call State Model (Adapted from (Sun Microsystems 2002)) 45
3.24 JCAT Connection Finite State Machine (Adapted from (Sun Microsystems 2003)) . . . 46
3.25 JCAT Terminal Connection Finite State Machine (Adapted from (Sun Microsystems 2003)) . . . 47
3.26 OSA/Parlay Call Control Inheritance (Adapted from (Jain et al. 2005, pg. 163)) . . . 49
3.27 Generic Call Control Manager FSM (Adapted from (ETSI and Parlay 2005a, pg. 46)) . . . 50
3.28 Call FSM (Adapted from (ETSI and Parlay 2005a, pg. 46)) . . . 51
3.29 Multi-Party Call Control Call Model . . . 52
3.30 Multi-Party Call Control Manager FSM (Adapted from (ETSI and Parlay 2005b, pg. 52)) . . . 53
3.31 Multi-Party Call FSM (Adapted from (ETSI and Parlay 2005b, pg. 53)) . . 54
3.32 Multi-Party Originating Call Leg FSM (Adapted from (ETSI and Parlay 2005b, pg. 56)) . . . 55
3.33 Multi-Party Terminating Call Leg FSM (Adapted from (ETSI and Parlay 2005b, pg. 64)) . . . 56
3.34 INVITE Dialogue State Machine (Adapted from (Rosenberg, Schulzrinne and Mayh 2005)) . . . 59
3.35 CS1 Originating BCSM and OSA/Parlay MPCC Originating Call Leg Mapping (Adapted from (Vannucci and Hanrahan 2006)) . . . 62
3.36 CS1 Terminating BCSM and OSA/Parlay MPCC Terminating Call Leg Mapping . . . 63
3.37 SIP INVITE Dialog FSM and OSA/Parlay MPCC Mapping . . . 65
4.1 Extended Call Control Connection Model . . . 69
4.2 Extended Call Control Call Model . . . 70
4.3 Creation of multiple participants for a conference call . . . 74
4.4 Parlay-X Extended Call Control Application (Adapted from (Vannucci and Hanrahan 2007b)) . . . 75

4.5 Virtual PBX Example (Adapted from (Vannucci and Hanrahan 2007b)) . . 76
5.1 Virtual Private Branch Exchange Use Cases 90
5.2 Virtual PBX Graphical User Interface 93
5.3 Call Created 94
5.4 Add Extension 94
5.5 Transfer an Extension 94
5.6 Remove an Extension 94
5.7 End a call 94
5.8 Manual Web Service Tester 95
5.9 Virtual Private Branch Exchange Implementation 96
5.10 Third Party Proxy Web Service 97
5.11 Mapping Extended Call Control Call Model to Asterisk 99
5.12 Extended Call Control Registration 100
5.13 Extended Call Control Multi-Party Call Creation 102
5.14 Extended Call Control Participant Transfer 103
5.15 Extended Call Control Add a Participant 103
5.16 Incoming Call Notification 104
5.17 Extended Call Control Remove a Participant 105
5.18 Extended Call Control End Call 106

List of Tables

1.1 Advanced Call Control Functionality 4
3.1 Originating Detection Points and OSA/Parlay MPCC Notification Mapping (Adapted from (Vannucci and Hanrahan 2006)) 61
3.2 Terminating Detection Points and OSA/Parlay MPCC Notification Mapping 62
4.1 registerRequest 79
4.2 registerResponse 79
4.3 createCall 80
4.4 createCallResponse 80
4.5 endCallRequest 81
4.6 endCallResponse 81
4.7 addCallParticipantsRequest 82
4.8 addCallParticipantsResponse 82
4.9 moveCallParticipantsRequest 82
4.10 moveCallParticipantsResponse 82
4.11 removeCallParticipants 83
4.12 removeCallParticipantsResponse 83
4.13 removeCallParticipants 84
4.14 removeCallParticipantsResponse 84
4.15 Transitions 85
4.16 Originating Notification Mapping of IN Q.1214 and Parlay Multi-Party Call Control notifications to Extended Call Control Call Model 86
4.17 Terminating Notification Mapping of IN Q.1214 and Parlay Multi-Party Call Control notifications to Extended Call Control Call Model 87

Acronyms

AJAX Asynchronous Javascript and XML

API Application Programming Interface

ATM Asyncronous Transfer Mode

BCSM Basic Call State Model

CAMEL Customized Applications for Mobile Network Enhanced Logic

CCF Call Control Function

CORBA Common Object Request Broker Architecture

CS1 Capability Set 1

DP Detection Point

ECC Extended Call Control

ETSI European Telecommunications Standards Institute

FSM Finite State Machine

GCCS Generic Call Control Service

HTTP Hypertext Transfer Protocol

IMS IP Multimedia Subsystem

IN Intelligent Network

INAP Intelligent Network Application Protocol

IP Internet Protocol

ISDN Integrated Services Digital Network

ISUP ISDN User Part

JAIN Java APIs for Integrated Networks

JCAT Java Coordination and Transactions

JCC Java Call Control

JCP Java Call Processing

JTAPI Java Telephony API

MPCC Multi-Party Call Control

MPCCS Multi-Party Call Control Service

OSA Open Service Access

PBX Private Branch Exchange

PSTN Public Switched Telephone Network

SCF Service Control Function

SCP Service Control Point

SIB Service-Independent Building Block

SIP Session Initiation Protocol

SMS Short Message Service

SOAP Simple Object Access Protocol

SSF Service Switching Function

UAC User Agent Client

UAS User Agent Server

UDDI Universal Description Discovery and Integration

URL Universal Resource Locator

VPBX Virtual Private Branch Exchange

WS Web Service

WSDL Web Services Description Language

WSRF Web Service Resource Framework

XML Extensible Markup Language

Formatting Conventions

The following formatting conventions are used throughout the document.

`State`	States are represented in this font.
OBJECT	Objects are represented in this font.
Method	Method names are represented in this font.
Point in Call	Call model point in call positions are represented in this font.

Chapter 1

Introduction

Telecommunications operators rely on services to provide added value to the network beyond that of voice traffic. Provision of these services and the opening of the network to third parties enables network operators as well as customers to derive greater value from the network. The move to transform the network from a closed system (that is operated and programmed by a few companies) to an open system that, in principle allows any third party access to the network, is now a standard industry trend. This transformation of telecommunications networks unlocks the possibility of services that telecommunication operators would not otherwise create, especially with regard to interworking with the Internet, which often addresses applications used only by a minority.

This research is focused on *call control*, the process by which an operator can create, manipulate and terminate call sessions within the network. In particular Web based call control is addressed and the requirements to support such control. Specifically how to provide developers with a means to abstract the required knowledge of telecommunications call systems, and therefore enable development of new, useful, innovative and lucrative services for both the operators as well as customers. Initiatives such as the Open Service Architecture/Parlay (OSA/Parlay) and Java APIs for Integrated Networks (JAIN) provide the foundation for this research.

Telecommunications networks are complex systems that have evolved to provide a separation of physical transport and logical control, allowing the provision of value added services such as voice mail, prepaid billing, and location to name a few. The ability to make two-person voice telephone calls is considered a commodity in networks, with competition focusing on tariffs rather than quality or availability (coverage), and as such advanced services provide a means for operators to differentiate themselves from competitors (Jain et al. 2005).

Service Delivery Platforms such as those supporting OSA/Parlay provide developers with an abstraction of the network, allowing developers to create applications that do not require understanding of the underlying network protocols such as ISDN Q.931, ISUP, INAP, or SIP. However in order for developers to create services they are required to have a good understanding of the mechanics of a telecommunications call and network setup and the steps required to complete the call. This requirement of telecommunications familiarity limits the number of developers for telecommunication services, as developers have to be familiar with the technologies and concepts used by network operators to offer access to their Service Delivery Platforms.

Network operators are looking towards Web services as a method to further increase the use of the network, and provide increased return on investment. Web services provide the potential to further abstract an operator's Service Delivery Platforms, in such a manner that IT developers require only superficial knowledge of telephony or telecommunications. Initiatives such as the Parlay X Web Services API aim to provide "powerful yet simple, highly abstracted, imaginative, building blocks of telecommunications capabilities that developers and the IT community can both quickly comprehend and use to generate new, innovative applications." (JWG 2002).

Telecommunication Web services are generally of a simple message exchange type, such as invoking the network to send an SMS or a ringtone to a mobile phone. In this case, third party developers have simple interactions with operators. Such services do not require Web developers to know the state of resources within the operators' network, and the session does not consist of multiple messages. Lack of information regarding the state of the operator network limits Web developers' applications. Dobrowolski, Grech, Qutub, Unmehopa and Vemuri (1999) found that the ability to provide complex services is proportional to the complexity of the knowledge of state, and in order for advanced call control applications, knowledge of call state is required (Vannucci and Hanrahan 2005b).

Web based call control is limited in responsiveness and state information if restricted to synchronous Web services. Figure 1.1 (Vannucci and Hanrahan 2005b) illustrates an example Click-to-Dial Web application making use of a Parlay-X like Third Party Call Control Web service where the application has to poll for updates of the status of the requested call, during the time between polling the state is uncertain. This limits control of the call to actions that are not heavily time dependent. Conversely *advanced call control* aims to provide Web services with a level of control similar to that of a call control application within an operators network, with functionality as described in Table 1.1.

Telecommunication service delivery platforms maintain the state of resources within a

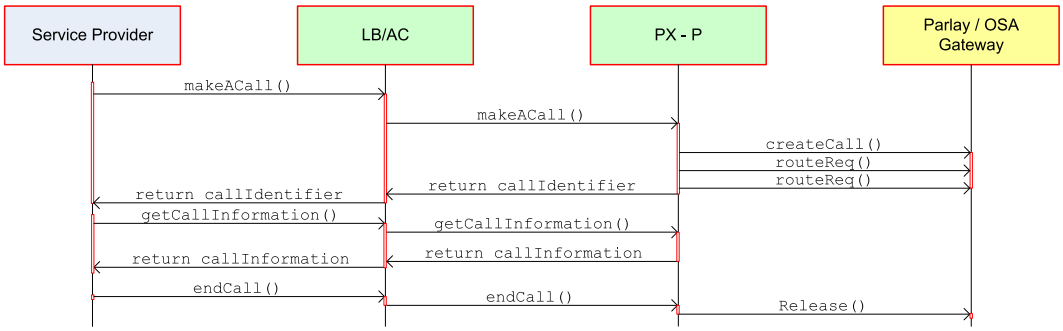


Figure 1.1: Generic Click to Dial application (Adapted from (Vannucci and Hanrahan 2005b))

session or call by means of call state models. The *call model* represents all the essential features of a session from either the application's or network's point of view (Jain et al. 2005, pg.39), and is a high level, technology independent abstraction of the call (Graf 2000).

Many call models exist for telecommunication networks and their Service Delivery Platforms as discussed in Chapter 3. However these call models are not created for Web based call control, thus this research defines an innovative call model suitable for Web service based advanced call control.

1.1 Research objectives

This research aims to develop a call model suitable for Web based advanced call control of a telecommunications network. The call model should support an Extended Call Control Web service which would provide advanced call control functionality.

A proof of concept implementation of the Extended Call Control call model is required to show the applicability of the model in abstracting the complexity of a telecommunications network.

The Extended Call Control call model and API should fulfil the following requirements for advanced call control:

- The developed call model should have various levels of control and abstraction for Web developers.

Service Description	Required Functionality	Interaction
Application Initiated Call	call setup	Pre Call
Network Initiated Call Triggers: Originating	notifications	Pre Call
Network Initiated Call Triggers: Terminating	notifications	Post Call
Network Initiated Notifications: Application	notifications	Mid Call
Call completion	call leg control	Mid Call
Join call legs	call leg control	Mid Call
Call transfer: Blind	call leg control	Mid Call
Call transfer: Supervised	call leg control	Mid Call
Call hunting: Sequential	synchronous call setup	Pre Call
Call hunting: Parallel	asynchronous call setup	Pre Call
Conference: attach	call leg control	Mid Call
Conference: detach	call leg control	Mid Call
Conference: reserve	call setup	Pre Call
Time Limits (call):	call leg control	Mid Call

Table 1.1: Advanced Call Control Functionality

- The call state model should contain sufficient functionality to interwork with existing call models.
- The call model must be capable of supporting control of multiple parties.
- The call model should provide a complete view of the connection and accurate description of the state of other parties.
- The call model must provide the ability to alter a party during any part of the call.
- The call model must be not be overly complex for developers and regardless of the parties present information in a similar manner.
- The call model must be equally well suited to an all Internet Protocol or circuit switched Public Switched Telephone Network (PSTN), and therefore should be able to map to SIP, OSA/Parlay, and CAMEL.

The development of the Extended Call Control call model and Extended Call Control Web service is the focus of this research.

Scope of research

This research is intended to present the call model and Web service required for advanced call control. The following limit the scope of research into this topic:

- The functionality available to the Web developer through the Extended Call Control Web service is dependent on the telecommunications operators' underlying network capability.
- Control of a call requires detailed knowledge of underlying network state and as such the Extended Call Control Web service is assumed to be within a trusted domain of the telecommunications operator.
- The use of the Extended Call Control Web service and associated Extended Call Control call model by the developer requires prior service agreements which are not considered in this research.
- Security of Web applications and requirements for cross-domain invocation are not considered.
- Authentication, authorisation and accounting of the Web applications making use of the Extended Call Control Web service is not considered.
- Implementation and performance of an Extended Call Control Web service requires access to a telecommunications network and gateway and could not be explored in this research.

1.2 Thesis Outline

This thesis is organised as follows:
Chapter 1: The requirements for the research and the objectives are defined.
Chapter 2: Web services and how stateful Web services are currently implemented are reviewed. The requirements for stateful Web services and the key considerations for an asynchronous Extended Call Control Web service are discussed.
Chapter 3: Existing telecommunications call models are analysed with respect to the underlying requirements of a Web based call model, and features of each call model are identified for inclusion into the Extended Call Control call model.
Chapter 4: This chapter describes the main focus of the work, the Extended Call Control call model, as well as key methods required for an Extended Call Control Web service

Application Programming Interface (API).

Chapter 5: Use Cases are used to illustrate the application and suitability of the Extended Call Control call model to advanced Web based call control. The developed Use Cases are implemented in a proof of concept application. Message sequence charts of the Use Cases detail the interworking of the components of the distributed implementation.

Chapter 6: The output of the Extended Call Control call model and Web service research is summarised as well as its potential applicability to future work.

Appendix A: Papers which resulted from this research.

Appendix B: A selected paper.

The source code for the proof of concept implementation is provided on a separate removable storage disc.

Chapter 2

Stateful Web Services

Web services arose as a response to the demand for software-to-software interactions through the Internet. Web services are based on four core technologies: Extensible Markup Language (XML), Universal Description Discovery and Integration (UDDI), SOAP (originally defined as Simple Object Access Protocol), and the Web Services Description Language (WSDL). Together these technologies provide a methodology of interconnecting software processes across multiple domains and systems. Newcomer (2002) states that any system can be mapped to Web services, and Web services can be mapped to any system. Web services can exist in any Web environment for example the Internet, company intranet and extranet (Esposito 2002, pg. 285).

There are many differing opinions as to the definition of a Web service, for example Esposito (2002) considers a Web service to be "a software application that can be accessed over the Web by other software" , whilst Jønvik et al. (2003) considers XML Web services as "a new breed of Web applications, which are defined as self contained, self describing, modular applications that can be published, located and invoked across the Web". Caprio and Moiso (2003) have a broader interpretation of a Web service and define it as "an interface that describes a collection of operations that are accessible on the network through standardised messaging mechanisms", Newcomer (2002) narrows Web services to XML as "Extensible Markup Language (XML) applications mapped to programs, objects, or databases or to comprehensive business functions".

This highlights the fact that Web services are designed solely for system-to-system interaction: Web services receive XML text messages, convert them to a format understood by the underlying system, which processes the information and optionally sends a response. In addition to providing data independance for programming languages, Web services also include semantic information associated with the data, thereby providing a complete

definition of the data and how to process the data (Newcomer 2002, pg.16).

In Esposito (2002) Web services have three characteristics,

1. A Web service is acessed over the Web using an uniform resource locator (URL).
2. Web service communication is performed using Extensible Markup Language (XML), usually packaged using the Simple Object Access Protocol (SOAP).
3. Web services are published in public registries together with a description of its methods and other publisher information using WSDL.

Web services can use many different communication protocols, however Hyper Text Transfer Protocol (HTTP) is most commonly used as it is the protocol of choice for the Internet.

2.1 Web services usefulness

Web services operate with a different paradigm than that of other distributed computing systems. Standard distributed computing systems and architectures like CORBA and Java remote method invocation have a tight coupling between the various components in the architecture whilst Web services strive towards a loose coupling[1]. As such Web services can provide a further layer of abstraction and a defined method of integration between disparate systems, providing unmatched interoperability and integration (Newcomer 2002).

Tight coupling requires a higher overhead than low coupled systems and thus high coupling is not well suited for the Internet, as the presence of firewalls and elements such as network address translators and proxies limit tight coupling between disparate domains. In (Kaye 2003, pg. 133) a comparison of tight and loose coupling is given as shown in table 2.1.

As can be seen in Table 2.1 with respect to granularity, Web services are limited to simple text XML-based data structures instead of objects, and the use of self describing semantic definitions and the HTTP protocol ensure that Web services are platform and language independent (da Silva et al. 2004; IBM Web Services Architecture Team 2000). Web services in addition can themselves use dynamic service discovery to discover other Web services

[1]Hagel (2002) defines loosely coupled as: "Loosely coupled is an attribute of systems, referring to an approach to designing interfaces across modules to reduce the interdependencies across modules or components in particular, reducing the risk that changes within one module will create unanticipated changes within other modules. This approach specifically seeks to increase flexibility in adding modules, replacing modules and changing operations within individual modules"

Service Architecture (Adapted from (Kaye 2003, pg. 133))

	Tightly Coupled	**Loosely Coupled**
Technology Mix	Homogeneous	Heterogeneous
Data Typing	Dependent	Independent
Interface Model	API	Service
Interaction Style	RPC	Document
Synchronisation	Asynchronous	Synchronous
Granularity	Object	Text Message
Syntactic Definition	By Convention	Published Schema
Semantic Adaption	By Re-Coding	via Transformation
Bindings	Fixed and Early	Delayed
Software Objective	Reusability, Efficiency	Broad Applicability
Consequences	Anticipated	Unexpected
Development Time	Longer	Shorter

and bind to them at runtime, thus providing self configurable, adaptive Web services, which can be a conglomeration of multiple dynamically linked smaller Web services.

2.2 Architecture

A Web service architecture has three components: the Service Registry, the Service Provider and the Service Requestor. The Web service is described in a Service Description and contained within the Service Provider. The Web services model architecture is shown in Figure 2.1.

The Web service provider in a business sense is the owner of the service (Caprio and Moiso 2003), whilst in an architectural sense is simply a computer connected to the Web with a content delivery platform. The Web service registry is a directory containing descriptions of many Web services. Service requestors use the service description when binding to the service either statically or dynamically. Since a WSDL service description document is co-located with any SOAP based Web service, it is not necessary for the Web service to be published in a service registry: the service requestor, if aware of the location of the service, can statically bind to the Web service. The Web service requestor in an architectural sense is the application accessing the Web service, whilst in a business sense it is the customer requiring certain functions (Caprio and Moiso 2003).

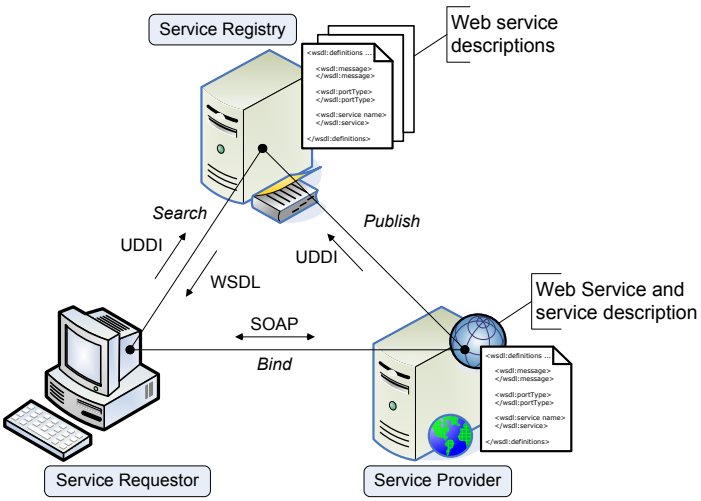


Figure 2.1: Service Architecture (Adapted from (Caprio et al. 2004; Caprio and Moiso 2003))

Figure 2.2 shows the Web service stack. XML is the key technology common to all Web services, be they SOAP-based or otherwise, and provides a common language to describe the data passing between the Web service requestor and the Web service provider and also how to process it (Newcomer 2002, pg. 16). SOAP defines the envelope to package the XML and provides serialisation of the data over the network (Newcomer 2002). Web service Description Language (WSDL) describes the service in terms of interfaces, data and message types, permitted responses, interaction patterns and protocol mappings (Newcomer 2002). Universal Description, Discovery and Integration (UDDI) is used for storing the business information related to the Web service, publishing and discovering the location of the Web service (Jønvik et al. 2003). As shown in Figure 2.2, HTTP is only providing transportation of the SOAP, and any network transport protocol would be acceptable.

2.3 Stateful Web Services

Standard practice for Web services and the Web servers that host these services is to handle each client request message separately, and in isolation of any previous or future messages. This lack of knowledge when dealing with a request is known as *stateless* behaviour. This inability to relate messages to a common session can be overcome by incorporating any information regarding state as part of the message exchanged. That is any information regarding state must be part of the message exchanged. Stateless services have the ability

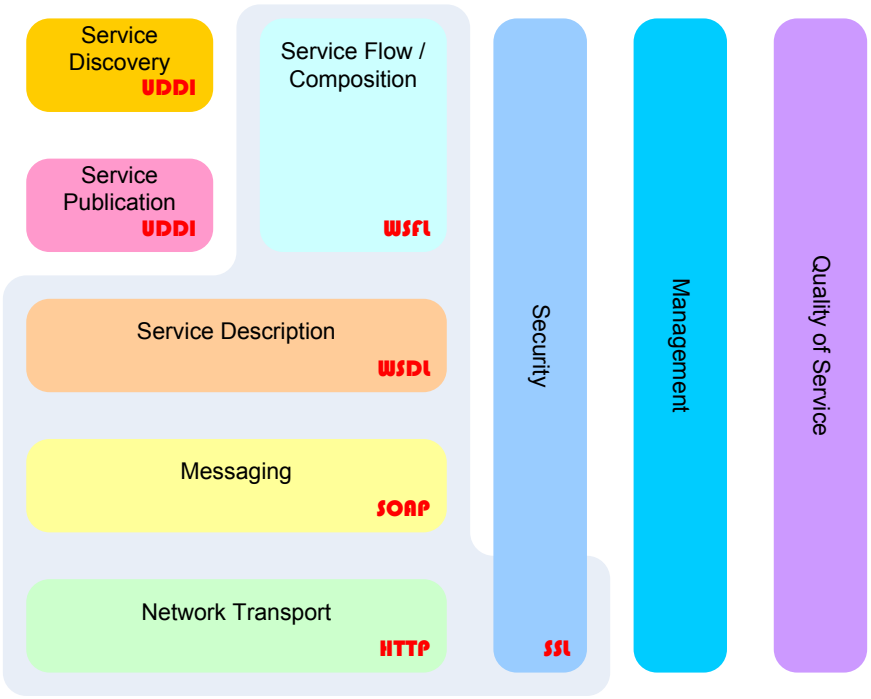


Figure 2.2: Web service stack (Adapted from Hogg et al. (2004))

to recover quickly from a failure due to the fact that the Web server does not require any data to be associated with a particular session. In addition the Web server does not require a large quantity of memory as separate processes are not required for each session (Weerawarana et al. 2005, pg. 55). The preservation of knowledge of the state of a process is important when the process spans a large amount of time, such as a telephone call, or an on-line shopping transaction requiring many related interactions with a customer (Vannucci and Hanrahan 2005b). This demand for service state between the requestor and service has led to a number of methods of maintaining state by means of a token, such as Web cookies, encoding URL parameters, and HTTP POST data (Hirsch et al. 2006; Vannucci and Hanrahan 2005b).

A stateful Web service is one in which the messages passed between the Web requestor and the Web service relate to a resource by means of a common context which is used when processing future messages. In Foster et al. (2004) three possible associations with state are presented:

Stateless. This is a Web service where all user information is passed in the requesting message and no information regarding the requestor is maintained on the server.

Conversational. This is a Web service where messages represent a series of related operations. The service uses each message to determine the behaviour of the service, as each message is logically related. A typical pattern for such a conversational Web service is by

means of an HTTP session or cookie (Foster et al. 2004).
Stateful Resource. This Web service uses sent and received messages to manipulate a set of logical stateful resources. However, in this association the Web service itself does not have to be stateful if the responsibility of maintaining state is handled by a separate component (Foster et al. 2004). This association forms the basis of the WS-Resource Framework (Czajkowski et al. 2004).

Stateful resources are any logical or physical elements with state, and can consist of a collection of other stateful resources. Stateful resources, as described in the stateful resource association above, provide a means to preserve the benefits of a stateless Web service, as dynamic state is not stored in the Web service itself. Rather, the state is stored within the request message or within components with which the Web service can interact (Singh and Huhns 2005, pg. 190).

Two ways of implementing a stateful Web service are commonly accepted; the first being a stateful service in which all state resides on the Web server and state identifiers are embedded in messages (Hirsch et al. 2006, pg. 60), or alternatively to use stateful interactions in which the full state information is passed between the requestor and the service in the exchanged messages.

The choice of stateless or stateful service is reliant on a number of factors (Hirsch et al. 2006):

- The number of users of the service; stateless services can support a large number of users, as memory constraints to not increase per user.
- The extent of state exchanged; when the reliance on previous information is relatively small state can be passed in a single message, however if the state exchanged is large then a separate context handler would be required to maintain the session state and associated data.
- The number of interactions; if the number of interactions per service is low, then state information does not need to be preserved, and a simple stateless service would suffice. However, as the number of interactions increase, the dependency on previous messages increases and the need to preserve state information requires a stateful Web service.

Events generated within a telecommunications network with regards to the progress of a call are numerous, and due to this reason, as well as the long duration of the call, and the

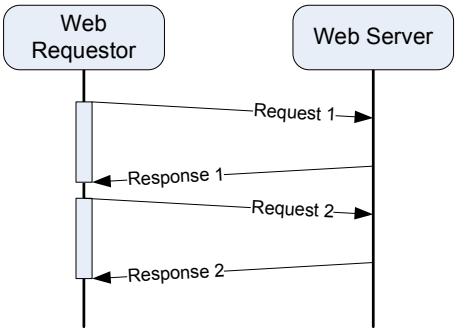


Figure 2.3: Synchronous Web service

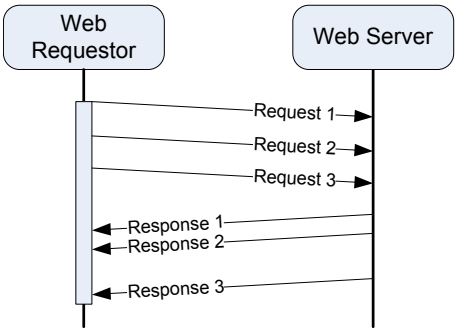


Figure 2.4: Asynchronous Web service

intention to control the call from a Web application, stateful Web services are required for Extended Call Control.

2.4 Synchronous and Asynchronous Stateful Web services

Invocation of any type of service is either synchronous or asynchronous. Synchronous invocation is when a request is expected to be immediately followed by a response, and no further computation takes place until the response is provided, as shown in Figure 2.3. Thus synchronous invocations block the flow of the application in that no other requests can be completed whilst the response is outstanding. Standard Web browsing is synchronous in that a HTTP GET or POST is immediately followed by a response. Asynchronous invocation is one in which the generation of a request does not block the process, and a result can arrive at an unknown time, as shown in Figure 2.4. The requestor can be notified of the response in one of two ways: either by providing a predetermined *callback* reference, usually an URI, so that the responding process can notify the requestor immediately when ready, or by the requestor performing *polling* to check the status of the request (Singh and Huhns 2005, pg. 180). The implementation of a callback for Web services is a topic of much research, as usually all Web connections are created by the service requestor, and a Web service cannot initiate connections to the requestor. Callback Web services can be implemented in a number of ways and are known variously as “server push”, “streaming”, “two way Web” or “comet” (Mahemoff 2006, pg. 19), and are a key component in grid service computing research. The notification pattern requires the Web requestor to provide a URI to which an asynchronous response can be sent, which updates the Web requestor by means of a background process. This effectively requires the requestor to act like a service provider by accepting incoming connections. This notification pattern is used by third party call control Web services such as Twilio.com.

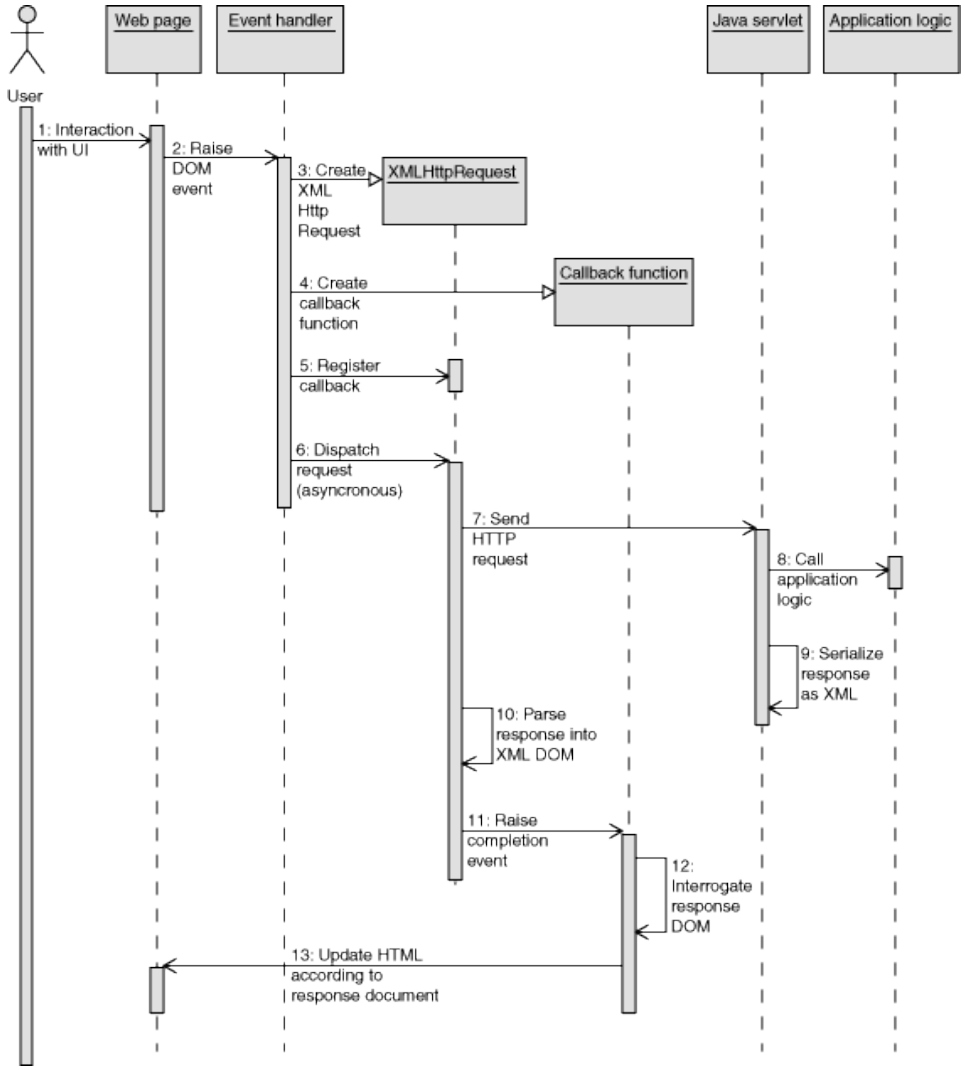


Figure 2.5: Ajax Browser Operation (Adapted from (McCarthy 2005))

An application implementing polling would have a separate process where the result of the request is checked at regular intervals, notifying the application when the result is available. Polling is wasteful of resources as it generates messages periodically which can be costly in terms of bandwidth and processing power. However, such a method is essential if the requestor is without a fixed location and can therefore not provide a callback, as is the case with a majority of Web requestors behind firewalls. The use of a separate process to perform the underlying polling and communication with the Web server forms the basic premise of Ajax or "Asynchronous JavaScript + XML", and is known as the *periodic refresh* or *call tracking* pattern (Mahemoff 2006), as shown in Figure 2.5.

Enablement of Asynchronous Web services is usually entirely client-side driven (Freeman et al. 2002, pg. 334), by means of the described polling method. This means that standard Web services can appear to be asynchronous, without requiring a server side change of

the service. However, this method of providing asynchronous services is not a true asynchronous invocation as the Web requestor is always the party to initiate the communication.

Ajax is an architectural style of providing seemingly asynchronous Web applications, which consume Web services. (A Web application or Rich Internet Application is a program which consumes Web services with a graphical user interface for human machine interaction). Ajax consists of a number of related technologies and ideas namely asynchronous JavaScript, cascading style sheets, the Document Object Model and most importantly uses XMLHttpRequest (Mahemoff 2006, pg. 6). Due to Ajax using JavaScript such applications can be upgraded on the service provider domain without user intervention, and as such can continuously offer better features.

Desktop Web applications or widgets by contrast have a number of features that extend beyond the capabilities of Web browsers (Mahemoff 2006, pg. 66), some of which are:

- Access to URI other than the Web service URL, providing connections to other servers on different ports, and possibly in other protocols.
- Access to hardware, such as local or remote file storage.
- Access to other software processes, such as databases, or client-side Web services.
- Faster processing capabilities, as the application is compiled for the operating system.

In the case of Extended Call Control, asynchronous behaviour is required to timeously notify the Web application of the constant change of state of the underlying network and associated calls, so that the application can perform call control based on correct knowledge of the state of the network.

2.5 Maintaining state in a Web server

There are a number of methods to maintain state in a Web server. Some of these are (Powell 2002):

1. Repost application data with each request. It is assumed that the state information contained within the HTTP GET or POST is not excessive.
2. Use HTTP authentication to map requests to users. The requestor is provided a session key which is included in all transactions.

3. Use cookies[2] to preserve the state of a string of requests. Cookies have a name/value pair for identification of the client application on the Web server. There is a session state class that has the session information and is continuously updated.

4. Unique URL. Redirect requests are sent back to the client application specifying a new URL with a unique session identifier. Occasionally redirection can be refused by a requestor for various reasons such as security (especially in the case of Web requestors that are not Web browsers, with user adjustable security levels, or user interaction). In the HTTP specification there is the possibility of accepting redirects and sending a HTTP GET in response to the redirect. This HTTP GET does not format the message in a SOAP bubble and is not well suited to non browser applications. Despite the process being automated, it is still necessary to confirm the URL redirect, otherwise the security of the application could be suspect.

Occasionally cookies on the client application store the state (Powell 2002). This implies the Web server is less burdened by the maintenance of state. In addition there are no issues about locating the session object across clustered Web servers. However, cookies are plain text and there are a number of security issues since confidential implementation and preference information might be obtainable. If there is a large amount of data serialisation a cookie might not be practical in limited bandwidth.

SOAP state information can be implemented in a number of ways, one method is to use a session identifier in the SOAP header. This implies that there is additional server side software to handle the header identifier, and secondly the session identifier has to be included by the client in each SOAP message sent by the client, otherwise state could be disrupted (Powell 2002).

2.6 Web Service Resource Framework

The WS-Resource Framework (WSRF) (Czajkowski et al. 2004), together with WS-Notification (Graham et al. 2004), defines an infrastructure for stateful services for business applications, grid based computing resources and systems management (Singh and Huhns 2005, pg. 190). The WS-Notification specification provides a scalable messaging model, whilst the WSRF provides the ability to model stateful resources.

[2] A cookie is a text file stored by the client, that is created by a Web server to both store and retrieve information about the client

The Web services architecture defines a service-orientated distributed computing model as a means for interoperating between diverse processes on different platforms. Distributed object computing has a number of architectural challenges, such as a lack of shared memory between the caller and the object, latency and unreliability of the underlying transport. Web services are suitable for distributed systems in which the gained platform and vendor neutrality justify the loss of performance.

Whilst Web services facilitate the provision of platform independent distributed systems, the basic Web service architecture does not deal with more sophisticated interactions, such as transactions and reliable messaging (Foster et al. 2004). Thus initiatives such as WS-Notification and WSRF move to address these issues.

The premise of the WSRF is to introduce a formalisation of interactions with state, i.e. the description, and modification of state.

In the WSRF the model that is adopted is a stateless Web service that acts upon a stateful resource. All responsibility for state is maintained by the stateful resource. This provides the Web service with the advantage of statelessness, such as scalability, and robustness to failure (in which the Web service can be restarted, or multiple different URI for the service can be provided) (Foster et al. 2004).

This leads to the need for correlation between the requests and the stateful resource upon which the requests act. Since Web services interacting with stateful components will modify the state of the resource, such a service interface cannot be provided without knowledge of the state in which the resource is currently, especially when the resource is acted upon by more than one Web service simultaneously. This strengthens the case for asynchronous notification of the underlying state to the involved Web requestors.

In the WSRF stateful resources have three properties, firstly they have a set of state data expressible in XML, secondly they have a defined life cycle (in terms of creation and destruction) and lastly they can be acted upon by one or more Web services (Foster et al. 2004). The implementation of the stateful resource is independent of standards, however the instantiation of the stateful resource may occur in one of two patterns: either statically or dynamically. Static association is when the association of the Web service with the stateful resource occurs when the Web service is deployed. Dynamic association involves a Web service factory, that creates the association when a requestor message starts a new session requiring state. Common to both methods is the concept of a stateful resource identity that is passed to the requestor by the Web service. This identity serves to correlate the requestor's messages with the correct stateful resource. The WS-Addressing standard

serves to standardise the manner in which the address of the Web service can be described.

The WSRF document also mentions another pattern to handle stateful resources, which is the maintenance of the resource's state by the Web service as a static state, thus removing the need for the Web service to provide a resource identity. However, this implies that the Web service itself has a one-to-one mapping with the resource identity, and the requestor would thus access the Web service through a unique Web service endpoint, such as a unique URL (Foster et al. 2004).

The stateful resource identity serves to enable the Web service to correlate the messages with the correct stateful resource, the identity does not contain any state information, and is unique in the life cycle of the Web service. The identity may be unique beyond the scope of the Web service, however this is not guaranteed. All messages regarding state contain the endpoint reference, so that the Web service can interact with the correct stateful resource.

A stateful resource can be associated with multiple Web services, allowing multiple network protocols or network endpoints to process messages for the WS-Resource.

The encapsulation of the stateful resource provides the requestor with the ability to modify the resource's properties without knowledge of the underlying systems, and varying degrees of encapsulation are possible. Access to the state of a given stateful resource can be accomplished with message exchanges between a single WSRF type, or alternatively a single stateful resource may be part of multiple WSRF types.

The state of the service requestor is always managed by the Web service, and the requestor can only provide messages to modify the state (Foster et al. 2004), thus eliminating the need for the requestor to have specific knowledge of the identity and location of the encapsulated stateful resource. This provides additional security and a well understood interface to the implementation.

Ultimately, the WSRF provides a standardised pattern to describe access and manage stateful resources, without compromising the statelessness of the Web service. Thus there is still the standard request-response for all Web service messages, and the stateful resource normally cannot directly initiate an update or notification to the service requestor, as is the case with the Web server.

2.7 Conclusion

Web services provide lightly coupled interoperation between distributed systems across multiple heterogeneous domains. The use of XML and text-based communication allows Web services to interact across firewalls that would otherwise prevent communication. The Web service definition allows Web applications to access SOAP based Web services in a well understood manner. The need for stateful operation in the case of an Extended Call Control Web service is justified by the large duration over which the Web application maintains control of the call. The need for asynchronous notification of the state of the underlying network is highlighted by the number of interactions, as well as the need for the Web application to have precise knowledge of the underlying network state to effect logical changes within the network. Sessions occuring over a large duration of time can impose significant performance penalties when synchronous methods such as polling are used, asynchronous Web services reduce time spent on HTTP invocations as well as providing notification as soon as the information is available, as shown in (Wang et al. 2010), where a 69% reduction in invocation duration was observed.

The WSRF provides a mechanism for Web services to provide stateful services by means of a separate process to handle the session state and associated data. The use of an identifier for the identification of the correct stateful resource allows Web services to operate in a stateful manner without sacrificing the advantages of stateless services. By means of a common resource identifier, multiple Web services can act on the same stateful resource simultaneously. A one-to-one mapping between the Web service and stateful resource is to be avoided as this can lead to a number of problems should the Web server become unavailable, such as a power outage or restart.

Using a separate stateful resource allows a Web based Service Delivery Platform to support a large number of stateful interactions. The use of Ajax enables asynchronous access to Web services and coupled with stateful resources, Rich Internet Applications can be supported. The state of the service requestor is always managed by the Web service, and the requestor can only provide messages to modify the state (Foster et al. 2004), thus eliminating the need for the requestor to have specific knowledge of the identity and location of the encapsulated stateful resource. This provides additional security and abstraction of underlying implementation for the telecommunications operator and a well understood interface to the implementation for third parties.

In Chapter 1.1 a number of objectives were identified for an Extended Call Control call model and API. A stateful resource pattern and identifiers, to relate stateful interactions,

provide the ability to associate control for the entire duration of the call. In addition the stateful resource and asynchronous notification of changes in state provide a current view of the connection.

Chapter 3 analyses existing call models for elements suitable for a Web based call model.

Chapter 3

Call Control Call Models

Telecommunication Web services are usually of a simple message exchange type, such as invoking the network to send an SMS or a ringtone to a mobile phone. In the case of these services the session does not consist of multiple messages nor state information. Lack of state information limits advanced telecommunications Web services, and in (Dobrowolski, Grech, Qutub, Unmehopa and Vemuri 1999) the ability to provide complex services was found to be proportional to the complexity of the supporting state model, as the number of states and transitions in the state model allows finer grained processing and information.

Unlike Web services, provisioning standard or advanced telecommunication services of a level similar to an Intelligent Network service like call waiting, conference calling, or pay-per-call require many messages to be exchanged during the life of the particular service being provided. In addition, control of the session by the service logic is in place for the entire duration of the call. These services require the controlling application to implement a call model to interpret network state messages based on the last known state of the session (Dobrowolski, Montgomery, Vemuri, Voelker and Brusilovsky 1999b).

Control of resources within a system requires knowledge of the state of the resources, as well as the state of the system in terms of the ability to modify the resources. Telecommunication service architectures maintain the state of resources within a session or call by means of call state models. The *call model* represents all the essential features of a session from either the application's or network's point of view (Jain et al. 2005, pg. 39), and is a high level, technology independent abstraction of the call (Graf 2000).

A call model can be defined in many different ways, in Dobrowolski, Grech, Qutub, Unmehopa and Vemuri (1999) it is defined as "an abstract representation of user and/or terminal and/or network expectation built during the process of establishing, progressing and terminating a call. A call model is most conveniently represented using the notation of

a graphical finite state machine (Moore and Mealy state machines are commonly used)." In Jain et al. (2005) a call model is thought of as "the basic component of a specialised virtual machine for the development of applications that involve communication sessions.", whilst Dobrowolski, Montgomery, Vemuri, Voelker and Brusilovsky (1999a) defines the call model as "a finite state machine that governs/controls the execution of a call, how a call progresses, what features/services may be accessed, and at what call states."

From the above definitions it is clear that in order to sucessfully control any call in a manner which allows access to the call during the session, requires knowledge of the state of the call and understanding of the call model. As outlined in Chapter 1.1 the goal of this research is to define a call model suitable for Web based advanced call control.

Within the context of a telecommunications service architecture such state is represented in terms of call state models to maintain track of the progress of each session and the resources consumed during the session. In Dobrowolski, Grech, Qutub, Unmehopa and Vemuri (1999) it was found that for any communication session, a call model can always be built to describe the logical entities involved and their inter-communication.

Call models are used to represent all the essential features of a call session from either the application's or network's point of view (Jain et al. 2005, pg. 39), and is a high level, technology independent abstraction of the call (Graf 2000).

Through the implementation of a call state model, a far richer set of service functionality becomes available, than that offered by a simple request response type (Dobrowolski, Montgomery, Vemuri, Voelker and Brusilovsky 1999b). Thus the call model can be thought of as the least common denominator supported by the call processing entities of the different domains (Dobrowolski, Grech, Qutub, Unmehopa and Vemuri 1999), in this case the Internet third party domain and the telecommunications operator domain. A call state model is extremely important for the service application developer to successfully understand and implement services within the telecommunications network.

In this chapter the bacground of call models is presented. A large number of existing call models are analysed for their applicability with regards to Web services and the research objectives of Chapter 1.1. Useful features of the call models reviewed in this chapter are synthesised in Chapter 4.

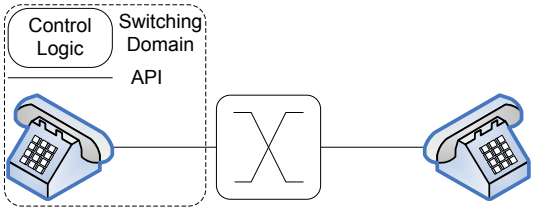


Figure 3.1: First Party Call Control (Adapted from (Bayer 2000; Graf 2000))

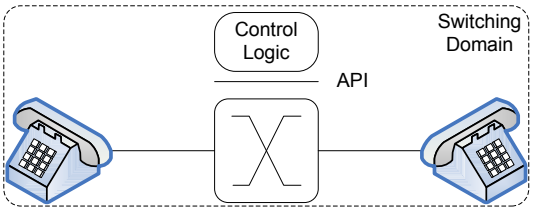


Figure 3.2: Third Party Call Control (Adapted from (Bayer 2000; Graf 2000))

3.1 Call Model Background

In the following section the concepts of call models are introduced with respect to the controlling entity (first or third party), information contained within the call model (full or half call model), and the contained states of the call model (symmetric or asymmetric). The application and suitability of these concepts will be discussed in further detail in the remainder of the chapter.

3.1.1 First and Third-Party Call Control

There are two distinct variations of call control; *first-party*, and *third-party*. In (Bayer 2000, pg. 303) *first-party call control* is defined as "a call control model in which only a single device or device configuration can be observed and controlled". Thus in first-party call control the caller has no control over the destination leg. In this scenario the application control logic is located at the terminal, as is shown in Figure 3.1, and is unaware of the controlling logic of the destination leg (Jain et al. 2005, pg. 44). Calls are seen as either external incoming calls or external outgoing calls to the switching domain (Bayer 2000). In first-party call control the application has the same control as that of a user (Graf 2000).

In third-party call control, application control logic is independent of all call parties and the application creates and controls multiple call legs. The application is usually contained within the operator domain, providing greater call control than a first-party call scenario. In Bayer (2000) *Third-party call control* is defined as "a call control model in which multiple devices, or device configurations, can be observed and controlled simultaneously". These calls are controlled by a single application, responsible for the maintenance of state of all the involved parties, as depicted in Figure 3.2 where the controlling logic addresses both parties of the call. The switching domain in third-party call control includes multiple call legs, and is the general case of call control, whilst first-party call control is the special case (Bayer 2000).

3.1.2 Full and Half-Call Models

In a call there might be many parties, each with a different connection and associated state. The entity recording the state, and the level of detail of the call model forms a basis for the system architecture, as the location of this information is an important factor of the operation of the call control.

There are, in principle, two ways in which to store the call model state, the first being each party maintains information only about itself, and the second being each party has full knowledge of the other parties. It is also possible to store all of the state information in just one side of the connection, however this is impractical as both sides of the connection require a certain amount of state information such as the other party's address and status of the connection to interoperate successfully. In the case of third party call control the controlling application has knowledge about each party.

It is possible to maintain the complete call information in both sides of the connection, and when the call model is such that all parties have a complete view of the connection and state of other parties, as shown in Figure 3.3, this is know as a *full call model*. If each party of the call is logically separate, with each party only knowing the state of their connection, as shown in Figure 3.4, this is know as a *half call model*. In Figures 3.3 and 3.4 each party's controller within the context of a call is represented by a rectangle. The state of each party within the controller is represented as a circle. As shown in Figure 3.3 state information regarding other parties is proxy information, represented by dotted circles, and as such might not be absolutely current.

The full call model is advantageous in that it represents all parties in the call, thus allowing a controlling application full knowledge of all parties, and therefore being able to alter any

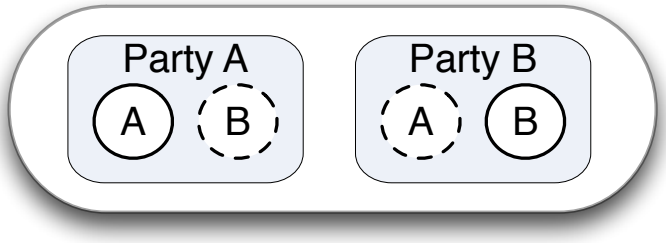


Figure 3.3: Full Call Model

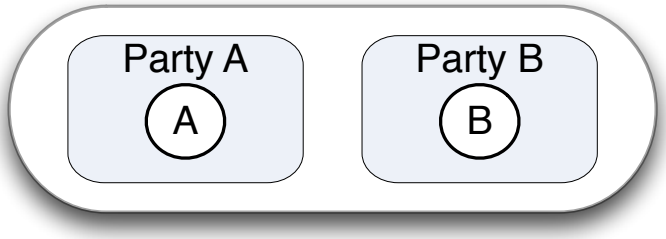


Figure 3.4: Half Call Model

party. Conversely objects representing the other parties have to be created for all parties, resulting in additional memory requirements (the total number of objects to represent each party is the square of the number of parties), and as the state of each object has to be kept current, this results in a large amount of messages regardless of whether or not they are required.

In a half call model the call state is simply updated as a message is received from the other call controller. This allows for a far simpler view of the connection, requiring an object for each party in the call (Jain et al. 2005). However, this reduction in state knowledge means that an application only has concise knowledge of the party within which the application is hosted and can not authoritatively alter other parties.

Regardless of the type of call model used, each party would require a basic common set of information such as the address of the other parties, as well as the supposed state of the parties, thus requiring the exchange of state information. In the case of third party call control, the use of a half or full call model is dependent on the technology and implementation, as shown in the following analysis of existing call models.

3.1.3 Symmetric and Asymmetric call models

The term symmetric and asymmetric refers to the states in the finite state machine describing the various parties involved in the call.

In the case of a call model having identical finite state machines for all parties involved in the call, regardless of their role as either an originating or terminating party, it is known as a *symmetric call model*. If the finite state machines are different depending on the role of the party then it is known as an *asymmetric call model*. In a symmetric call model, the call model for each participant will go through different states in the finite state machine depending on whether it is the terminating or originating party, with certain states not permitted, as described by the pre- and post-conditions of the API methods (Jain et al. 2005, pg. 53). Often, the terminating and originating finite state machines share similar states, for example an idle or connected state, and thus a symmetric call model has the potential to simplify the specification of the call model by making it more compact, and easier to manage (Jain et al. 2005, pg. 53).

3.2 The Intelligent Network

The Intelligent Network (IN) was one of the first service architectures to separate application or service logic from the network equipment, and as a result was one of the first architectures to introduce programmability. The IN call model was the first call model to facilitate call processing, and IN concepts form the basis for almost all other call models.

The standard public switched telephone network processing model before the introduction of the IN is shown in Figure 3.5. Supplementary services were independent, each hosted on the switch equipment (node), requiring a particular service to be loaded on each node supporting that service. In the case of an update all nodes within the network required updating, an extremely laborious and time-consuming process.

As a result of the difficulties of service provisioning the fundamental IN concepts focused around independence (Faynberg et al. 1996; Jain et al. 2005).
Service Independence: The IN architecture supports centralised separate service application processing platforms.
Logical separation of basic switching from services: Basic exchange functionality such as standard routing and call connection does not require the intervention of service application logic.

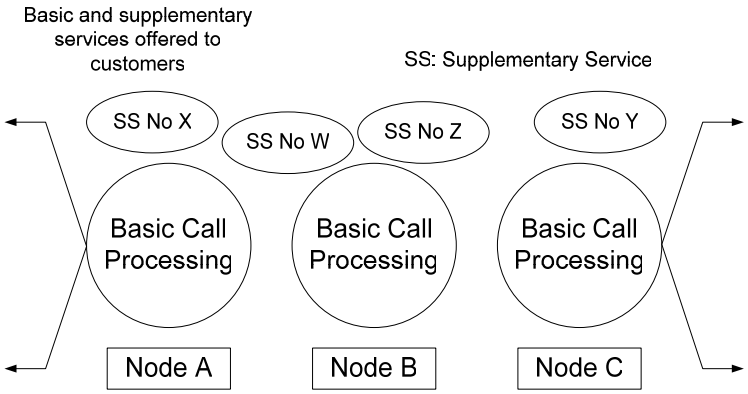


Figure 3.5: Pre IN service processing model (Adapted from (Jain et al. 2005, pg. 66))

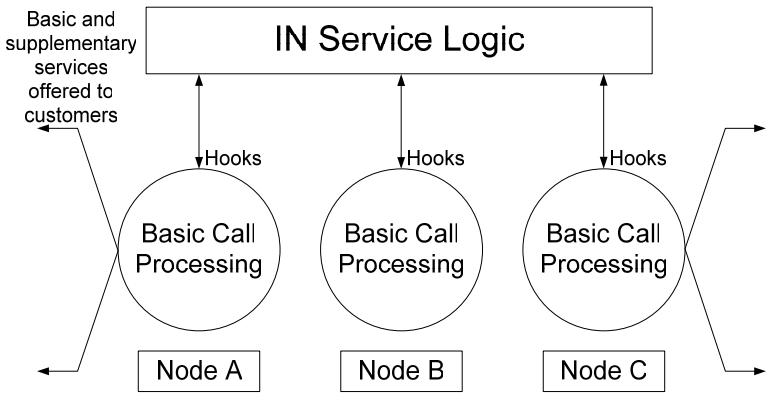


Figure 3.6: IN service processing model (Adapted from (Jain et al. 2005, pg. 67))

Independence of service interactions from lower level communication details: The service application logic communicates with the switching through a protocol independent of the underlying network, separating services from lower level protocols.

This network processing model was altered to provide a common service logic, and a way for the service logic to interact with the nodes though hooks, as in Figure 3.6. These hooks are instantiated as trigger detection points in the Intelligent Network basic call model, and allow suspension of call processing whilst the service logic determines the correct course of action to take and instructs the node how to continue. The processing of a call was also split into separate service-independent sub processes represented as states within the basic call model (Jain et al. 2005, pg. 67). The concept of a *Service Control Point* was introduced to host the service applications within the network, that would interact with the network switches and their associated call models, and limited service logic. The call model within the Intelligent Network is known as the *Basic Call State Model (BCSM)*, and is defined in the ITU-T's Q.1204 standard (ITU-T 1993).

There are several key attributes to an Intelligent Network system:

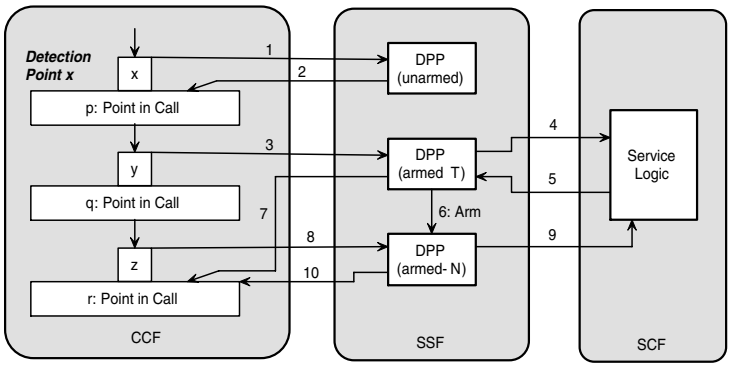


Figure 3.7: IN Service Invocation, from (Hanrahan 2007, pg. 15)

- The basic call process is available to all systems within the network.
- The basic call process is modular and independent of available services.
- Triggers allow any basic call process to interact with the service logic, and allow the service logic to suspend and control the basic call process.

The Intelligent Network call model is a half call model and thus the originating and terminating basic call models are separated, as shown in Figure 3.8 and Figure 3.9. Figure 3.8 (a copy of Figure A.2 (ITU-T 1993)) shows a recommended version of the finite state machine for the originating BCSM.

The BCSM consists of states, or points in call, which are numbered, and detection points, usually before each point-in-call. The detection points are places where the call execution logic may be suspended for further call processing handed over to the service application logic.

In Figure 3.7 the IN switch Call Control Function (CCF) contained within each exchange switch is responsible with connecting the parties from end to end, and passes though detection points whilst handling the processing of the call. At each detection point a lookup-table in the Service Switching Function (SSF) is checked to see if any statically armed *trigger criteria* are met. If no trigger criteria are met, call processing is resumed, however if there is the requirement for additional service logic, then call processing is suspended whilst the SCF by means of the INAP protocol notifies the service logic within the Service Control Point (SCP). The service logic returns instructions on how the SSF should proceed. It is possible for the service logic to request further detection points in the call to be set, as the particular context requires, which would lead to further SCP involvement.

In the BCSM shown in Figure 3.8 points in call one to six are the states during the creation

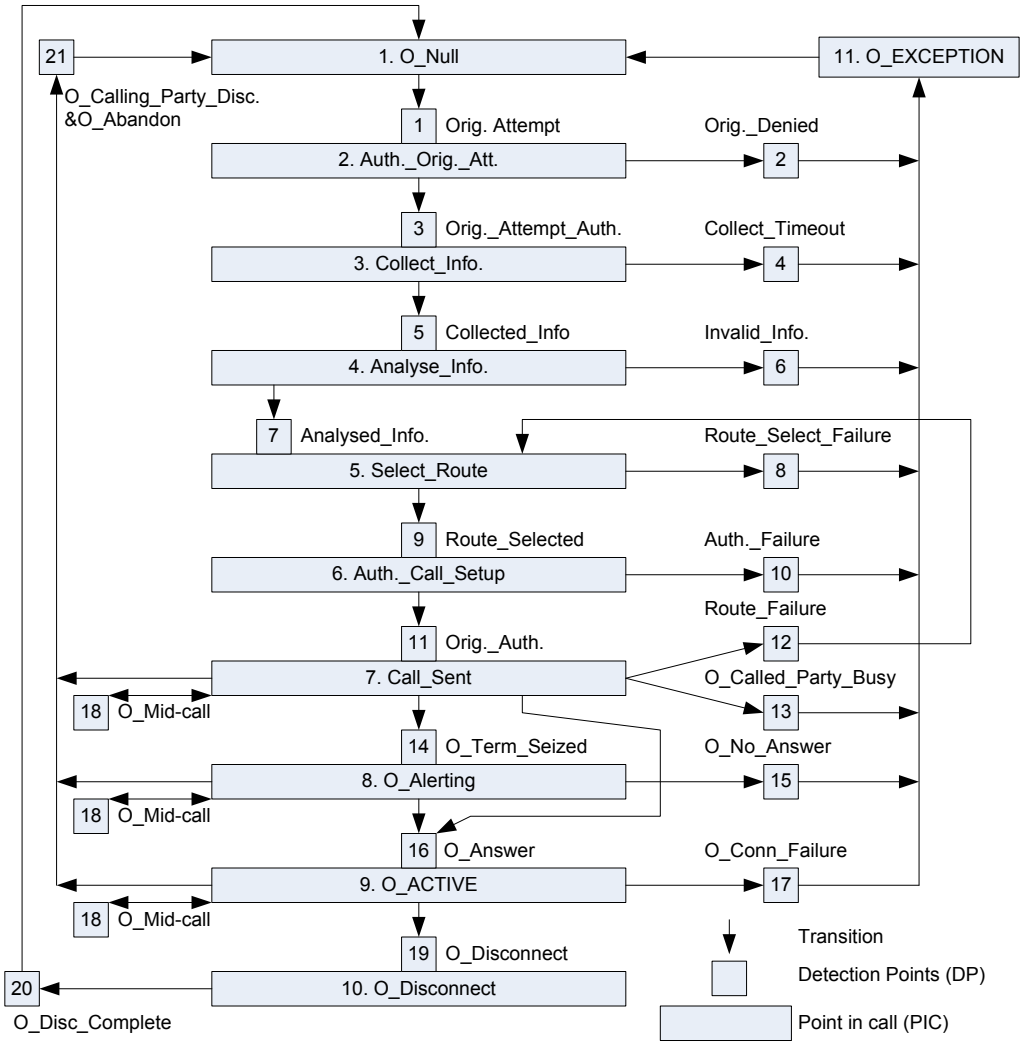


Figure 3.8: Example Originating BCSM (Adapted from (ITU-T 1993, After Fig. A.2))

of the call, states seven to nine during the call, and state ten during the release of the call. A transition from one completed point-in-call to another will always pass through a detection point, so that the opportunity to have control of the call processing is never lost.

With respect to the BCSM, as shown in Figure 3.8, calls begin in the `Null` state, in which case the call is being supervised by the switch, yet no action has been taken. This state occurs when the telephone goes off-hook, indicating to the switch that further events are to occur. The switch then transitions to the `Authorise Origination Attempt`, in this state the authority of the extension to place a call is confirmed against a service profile related to the telephone, such as a time varying charging plan.

If the origination attempt is authorised, the call transitions to the `Collect Information`

state, wherein the dialling strings are detected and collected by the switch. The dialling plan is consulted, to determine when sufficient digits have been collected to leave this point-in-call to begin the process of routing the call attempt.

In the *Analyse Info* point-in-call the collected address is analysed and translated according to stored dialling plans to determine the routing address. If a dialling string is collected that requires translation by a service control point, then the call processing can be suspended when leaving the point-in-call and passing through the detection point, for example if a dialling string has a premium number prefix marked for IN processing.

Once the address has been translated, the switch in the *Select Route* point-in-call selects the route, in terms of the physical switch address and port number. The authorisation to use the given route is then checked in the *Authorise Call Setup*, and if granted, the terminating switch is notified of the connection in the *Send Call* point-in-call. The terminating switch begins ringing the phone of the destination party, and the *Alerting* point-in-call is entered until such a time as there is an answer, or a timeout.

The receiving party answering the call results in a transition to the `Active` state.

Likewise the finite state machine for the terminating basic call state model is shown in Figure 3.9 (a copy of Figure A.3 (ITU-T 1993)).

3.2.1 IN Service Creation

The programation of IN services is a laborious process requiring detailed knowledge of the underlying network and operation. As such service development is constrained to specific service development platforms and a handful of developers. The logic of a service application is defined in terms of *Service-Independent Building Blocks* (SIBs), which are predefined reusable elements containing set uninterruptable scripts. SIBS are chained together in a process flow with branching conditions to create service logic programs (Faynberg et al. 1996). Whilst the different IN Capability Sets defined standardised SIBs, operators would typically develop their own proprietary SIBs that ensured their equipment and service development environment would be used throughout the network.

The Intelligent Network model is extremely important to all service architectures, as almost any service architecture has to provide a mapping from the defined interfaces to the Intelligent Network call state model. Application Programming Interfaces such as JTAPI, JCC/JCAT and OSA/Parlay only define the interfaces so as to facilitate service application

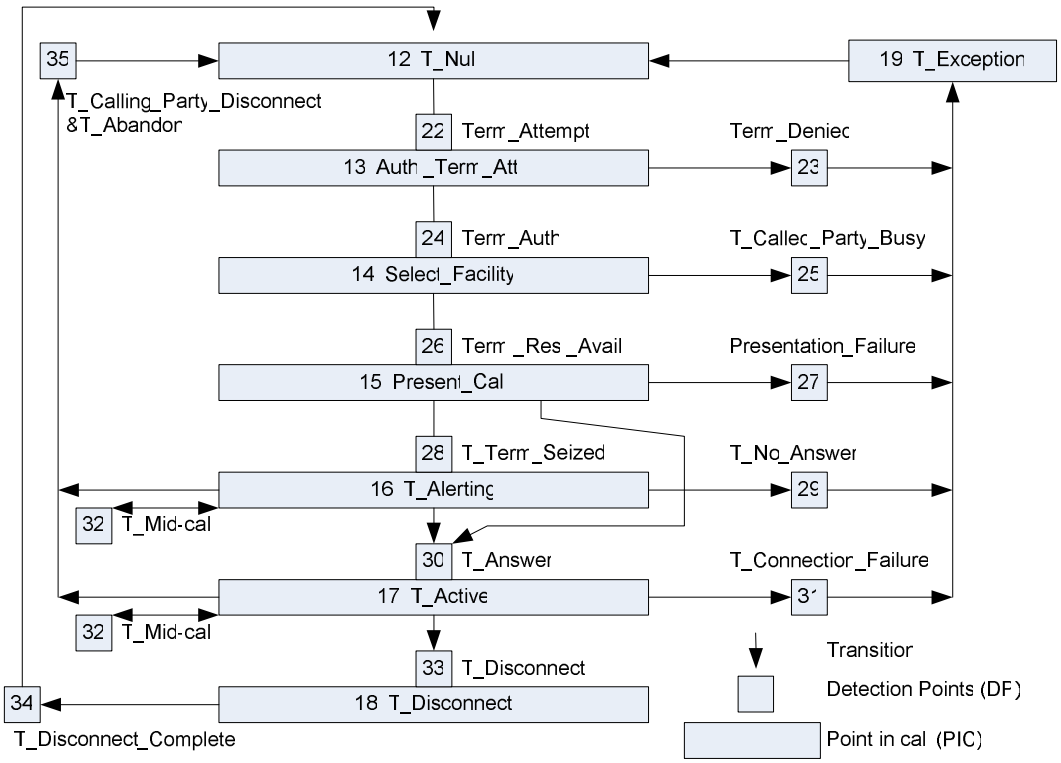


Figure 3.9: Example Terminating BCSM (Adapted from (ITU-T 1993, After Fig. A.3))

programming, but usually do not specify the underlying mechanism to implement the calls to the corresponding network equipment. The underlying mechanism is implemented by middleware implementations communicating using various protocols such as the Session Initiation Protocol, or the General Inter ORB Protocol and Internet Inter ORB Protocol in the case of CORBA. These middleware objects then communicate directly with the network equipment using the network equipment protocols such as Signalling System no 7 or ISDN user part (ISUP).

3.2.2 Conclusions

The Intelligent Network introduced the fundamental concepts that are now core to all telecommunications service architectures. The concept of service logic independence from the underlying architecture allowed simpler deployment of services. The basic call state model defined the call model that underpins all circuit switched operation in one guise or another, be it a mobile or a fixed line network. The use of a half call model to represent the

originating and terminating parties separately provide fine grained control over all aspects of the call, albeit at the expense of simplicity. The separation of call processing into sub functions allowed operators to create service independent building blocks. These service independent building blocks were packaged together to create service applications, however they lacked the now common concepts of an application programming interface. The concept of scripts to perform blocks of functionality lends itself to Web services, although in a different level of abstraction. The IN BCSM can be used as a reference to any Web based call model to ensure that regardless of the level of abstraction, a mapping is possible, for interworking with existing telecommunications networks. The lack of ease for third parties to develop standardised services lead to a number of further abstractions and technologies, as shall be discussed next.

3.3 JTAPI

The Java Telephony API (JTAPI) is an object orientated interface for Java based telephony applications, usually deployed within a private branch exchange (PBX) or Call Centre. JTAPI is not intended for use in large distributed systems with many interworking domains, but rather smaller centralised systems such as within a single corporate company. Due to object-orientation, development is intended to be faster than that of standard Intelligent Network methodologies, but the level of control of an advanced network is sacrificed, in that JTAPI does not specifically cater for suspension of call processing and invocation of application logic (Jain et al. 2005, pg. 88). This is an important consideration if the Extended Call Control call model and Web service is abstracting such a platform.

The JTAPI call model is defined in the JTAPI API definitions, as a collection of classes and interfaces, with a core call model javax.telephony and additional extensions that provide further granularity and methods to the core call model.

The JTAPI core call model, shown in Figure 3.10, uses six objects to represent the system, where each object corresponds to a physical or logical entity.

The PROVIDER object represents the software application that interfaces with the telephony system, and is considered to be an abstraction of the application, encapsulating the specifics of the underlying system, allowing platform independence. The PROVIDER also maintains information of all associated CALL objects in the underlying system, for a particular domain, regardless of the presence of an application.

The CALL object at its simplest models a telephone call and has state which describes the

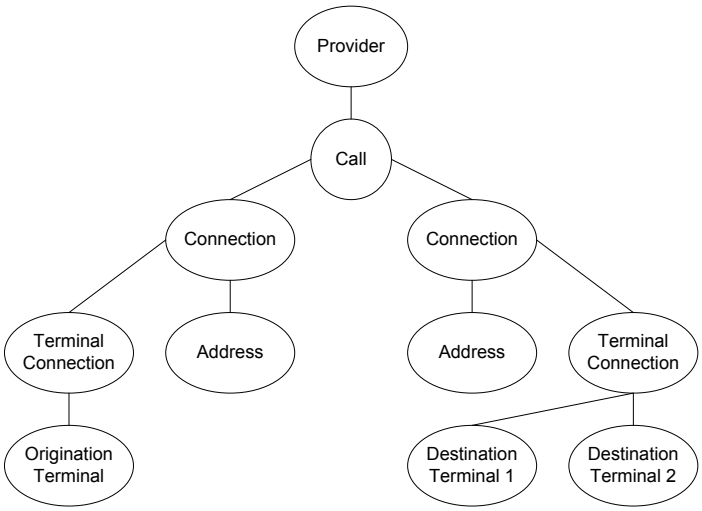


Figure 3.10: JTAPI Call Model

life cycle of the call and its associated CONNECTIONS, of which it can have zero or more (JTAPI 2002). Each party of the call is represented with a separate CONNECTION, and the CALL maintains a list of valid CONNECTIONS.

CONNECTION objects serve to associate CALL objects with ADDRESS objects and describe the relationship between the two. Each CONNECTION has an associated state that describes the particular stage of the call progress.

The ADDRESS object represents the *logical* endpoints of the call, being either an IP address or a telephone number (Jain et al. 2005; JTAPI 2002). ADDRESS objects are never created by the application, rather local ADDRESS objects are created by the PROVIDER when it is first instantiated. Objects to represent remote addresses outside of the domain of the Provider are created dynamically, based on received information.

The TERMINALCONNECTION represents the relationship between the TERMINAL and CONNECTION objects.

The call model is an important construct, enabling programmers to understand the order and processing of the call in greater detail by the various responsible objects. The CALL, CONNECTION and TERMINALCONNECTION object in the call model all have associated finite state machines (FSM). The FSM for the core call model CONNECTION object is shown in Figure 3.11 where shaded states are present on the Terminating side only. The finite state machine for the extended CONNECTION is shown in Figure 3.12, with additional detail for the `InProgress` and `Connected` states. A state transition arrow with a $\star$ shows that the state can be reached from any other state, whilst a `*/StateName` shows that the state can

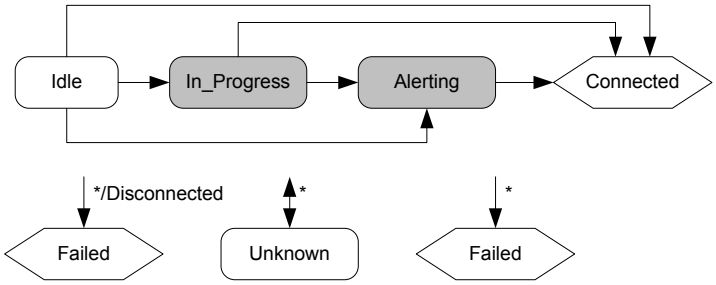


Figure 3.11: JTAPI Connection Object FSM

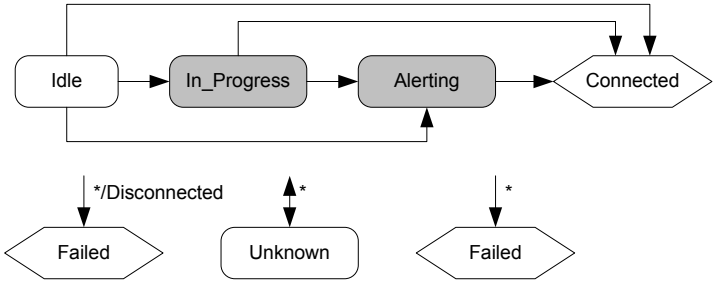


Figure 3.12: JTAPI Call Control extension Connection Object FSM (Adapted from (JTAPI 2002))

be reached from any state except `StateName`.

The Call Control extension adds further detail to the Core Control extension by splitting the `InProgress` state into `Offering` and `Queued`, where the `Offering` state is used to indicate that the call is being offered to the ADDRESS associated with the CONNECTION, with the controlling application typically having to accept or reject the call based on set policies before the called party is alerted (Jain et al. 2005; JTAPI 2002). `Queuing` allows the call to wait until the called party is available. The additional states are coupled with additional pre-conditions and post-conditions, as described in the API.

Where in the Core Control once a call had reached the network no further information could be gathered, the Call Control Extension added significantly more information. Depending on the ability of the telephone network to provide such information, the Call Control Extension added: reporting if the network is alerting the destination, if the network has begun to handle the processing of the call request, and whether the network has begun the process of routing the call (JTAPI 2002).

The sequence of changes for the objects that have FSMs are shown by means of a time sequence diagram, where the state of objects are indicated at different times, as shown in

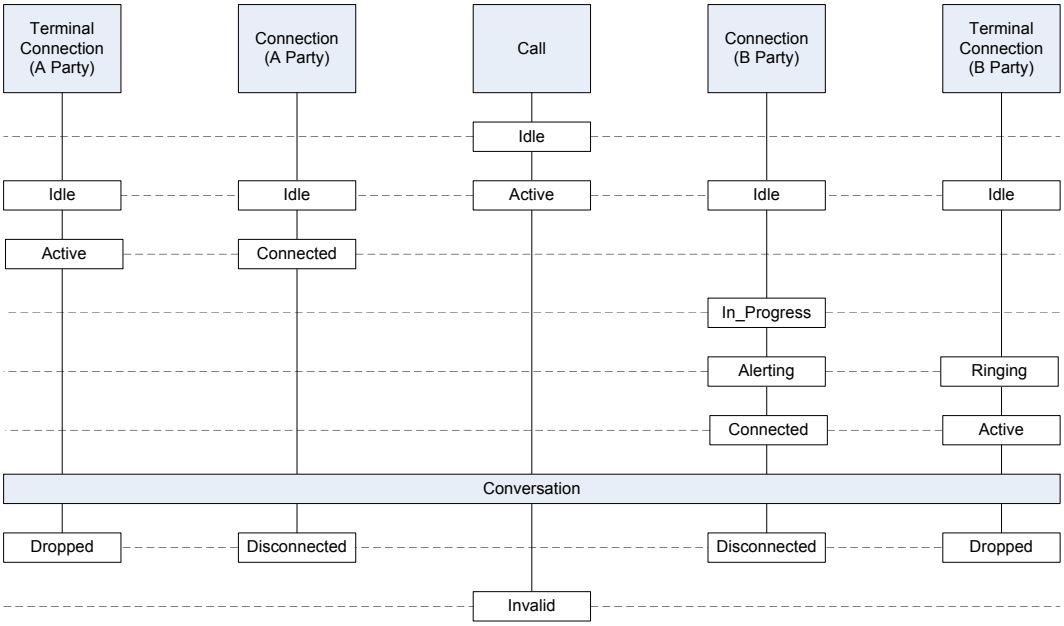


Figure 3.13: JTAPI State transitions for a two party call (Adapted from (Jain et al. 2005, pg. 42))

Figure 3.13.

A simple call configuration for a two party call is shown in Figure 3.14. The call model is a symmetric third-party view and therefore does not distinguish between local and remote users. In this example both parties are connected to a single JTAPI implementation and under the control of a local provider, thus the addresses are *local addresses* (Jain et al. 2005, pg. 88).

In this case the PROVIDER object represents the network provider responsible for these users. The CALL object has two legs, represented by the CONNECTION objects, and each CONNECTION object has an ADDRESS. TERMINAL objects represent different terminals the user is connecting with.

Application objects implement the interfaces required to control and query information from the call control objects.

The finite state machines for the CALL, CONNECTION and TERMINALCONNECTION objects are shown in Figure 3.15.

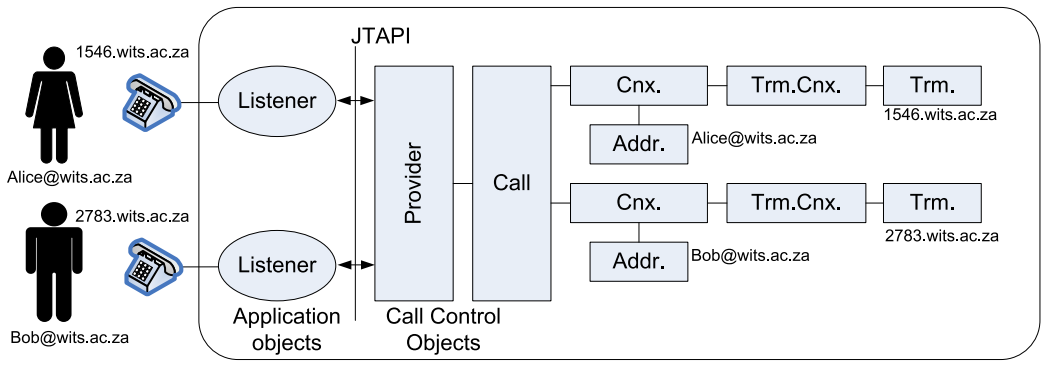


Figure 3.14: Call model for local two-party call (Adapted from (Graf 2000; Jain et al. 2005))

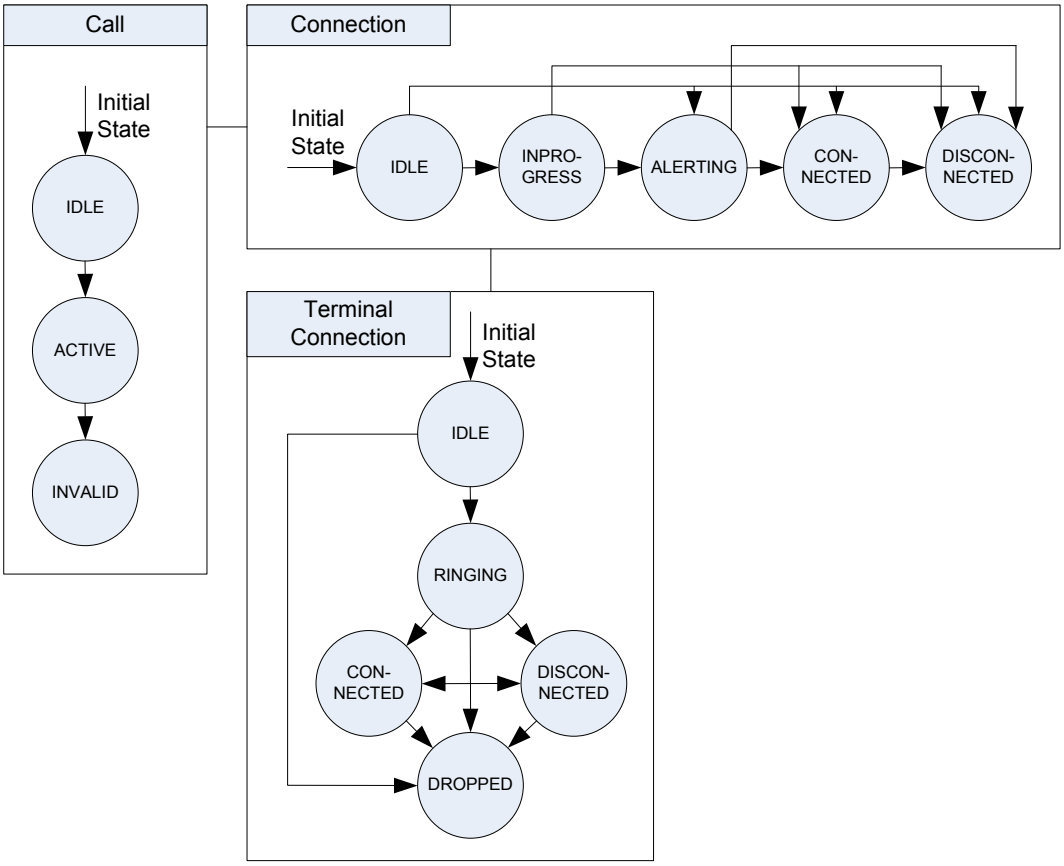


Figure 3.15: Finite State Machines for Call, Connection and Terminal Connection objects (Adapted from (Graf 2000))

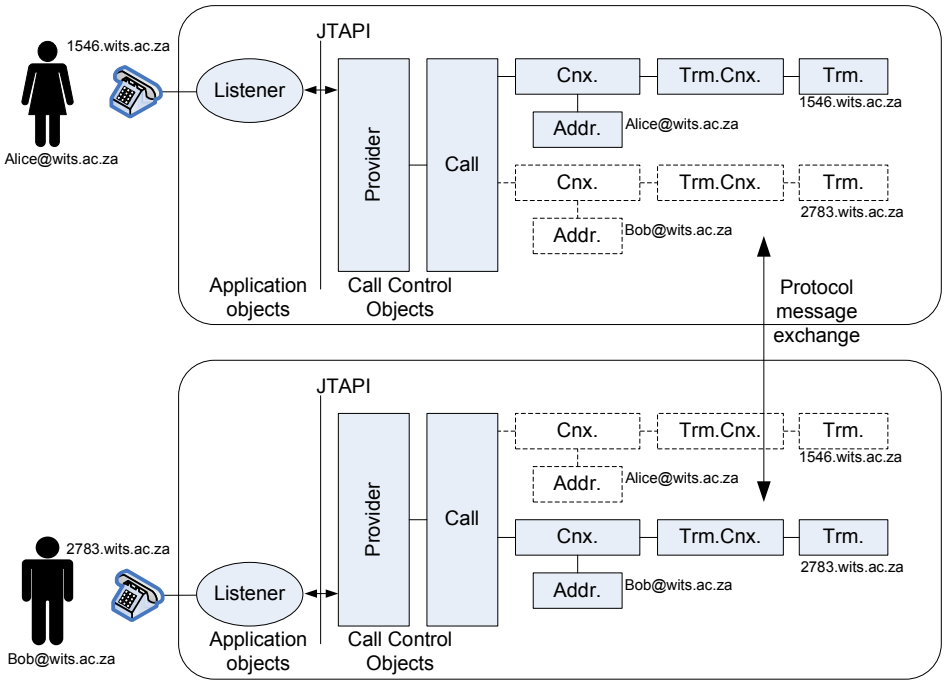


Figure 3.16: Call model for two-party call with remote addresses and a full call model

3.3.1 Distributed JTAPI

In the case of a distributed JTAPI implementation the providers are considered to be different and the parties in the call are represented by remote addresses.

As in section 3.1.2 such a situation can have two different models, a full-call model or a half-call model, as described in 3.1.1. In the case of a full call model, the remote parties are effectively proxy objects (state represented with dotted lines), as in Figure 3.16. Even though proxy objects are created the local applications have no control over the remote objects, and in order for a change to be effected in a remote object, protocol messages have to be exchanged with the remote provider, and enacted upon by the remote application.

In the case of a half call model proxy objects for the remote address are not created, and as in the case of a full call model, protocol messages have to be again exchanged with the remote provider and enacted upon by the remote application. However in the case of a half call model, the volume of messaging can be reduced slightly, as synchronisation with the remote party state machine is not required. The half call model for a remote call is shown in Figure 3.17.

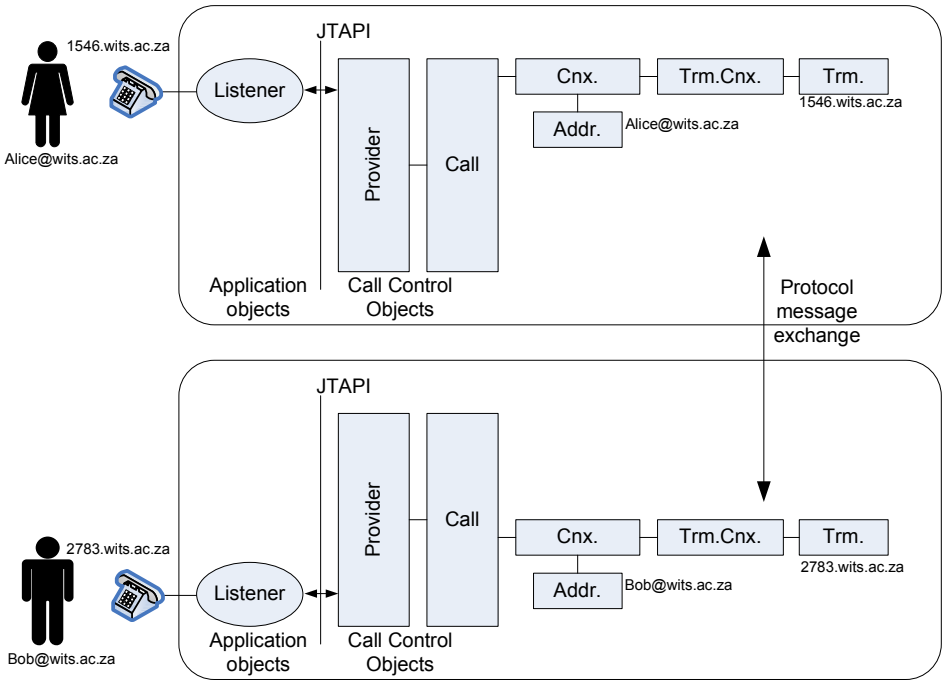


Figure 3.17: Call model for two-party call with remote addresses and a half call model

3.3.2 Conclusions

The underlying functionality of the service delivery platform will always dictate the possible functionality of an abstracting Web service. The ability to reflect state in a service delivery platform that does not specifically cater for suspension of call processing is an important requirement. Being able to operate on a call that is in progress that does not support suspension requires that relevant state information is passed to the controlling Web application timeously, thus reinforcing the requirement for asynchronous notification of change in state. The use of multiple objects to represent the state of the call, connection and terminal connection provides granularity on a high level view in the case of the CALL FSM, and specific information regarding individual connections in the case of the CONNECTION FSM. The availability of state progress information with regard to network operators in different domains has to be considered in the case of an Extended Call Control Web service. The JTAPI CONNECTION FSM is a symmetric call model, which presents a simpler abstraction to call state than the IN BCSM. The need to add the call control extension to split progress state into offering and queued is of special note to fulfilling the requirements of Extended Call Control, as laid out in Chapter 1.1, where the network provides the facility of call suspension. Figure 3.13 is useful in providing a summary of how individual CONNECTION FSMs interact with the CALL FSM, and the levels of abstraction provided by JTAPI. The CALL FSM, not differentiating between originating and terminating parties, is notable in

that it provides abstraction of call participants. The ability to support mid call operations is lacking in JTAPI, one of the requirements of Extended Call Control. JTAPI uses Java exceptions and the Java events model to report changes in state to an application (Jain et al. 2000), this correlates closely to asynchronous state notifications in a Web service. One of the most important assumptions used in JTAPI is that the provider is in control of all connections of the call.

3.4 JCC/JCAT

The JAIN subgroup known as the Java Call Control (JCC) and Java Coordination and Transactions (Java Call Control Extensions) (JCAT) Edit Group had a goal to develop an API that could be applied to not only the PSTN but also packet based networks such as IP or ATM (Jain et al. 2000). The resulting JCC/JCAT call model not only applies to the PSTN but also multimedia multi-party multi-protocol communication sessions (Jain et al. 2000).

The IN call model was developed for the PSTN, to be deployed on the PSTN architecture, namely telephone switches performing the basic call functionality, and JTAPI focuses on a centralised environment like a private branch exchange or call centre. JTAPI introduced object orientation, and provides clear abstractions for manipulating calls and the associated logical entities.

JCC/JCAT uses the concepts in both IN and JTAPI to create a new API, with benefits from each, allowing an abstraction of the many underlying protocols for managing calls and managing interaction between the call and the applications. JCC/JCAT is not bound to a single switch or domain, rather taking the view of a converged network with multiple domains and parties, and supports IN features such as third-party service invocation (Jain et al. 2000). The JCC/JCAT call model is intended to be a generic call model that incorporates the essential features of the IN as well as JTAPI models. The API is also extensible so that it may be revised and additional functions added. In Jain et al. (2000) call control is considered to be the ability to observe, initiate, answer, process and manipulate calls. The JCC/JCAT API provides independence of the underlying protocol and the type of transport network (Jain et al. 2000).

3.4.1 Java Call Control

The Java Call Control API developed by the JCC Edit Group had three functional areas, each with differing levels of abstraction and control. The three areas were *Elementary Call Control: JCP*, *Core Call Control: JCC*, and *Extended Call Control: JCAT*. Java Call Processing (JCP) package, initially the fundamental package for call processing included all basic facilities for the monitoring of calls. This was relatively simple on its own, and not very usefull for a complex service, and thus subsequently discontinued in favour of JCC. The Java Core Call Control included additional facilities for observing, initiating, processing and manipulating calls, as well as suspension of call processing and invocation of service application logic to JCP (Jepsen 2001, pg. 63). The Java coordination and transactions (JCAT) package further extended JCC to provide fine grained call control, providing support for detailed services. JCC and JCAT are now the supported APIs by the JCC Edit group.

3.4.2 Relation of JCC/JCAT API to Parlay API

There is an effort to align Parlay and JCC/JCAT as JCC is based on the language neutral Parlay 3.0 Multi-Party Call Control Service (MPCCS) (Sun Microsystems 2002). Figure 3.18 shows the relationship of the JCC API and the Parlay Call Control APIs, the JCC and JCAT packages are defined by the JAIN consortium. Parlay extends from JCC, but JCC is the official Java implementation of the Parlay call control (Sun Microsystems 2002). The Parlay call control objects finite state machines (FSM) are very similar to those of JTAPI (Jain et al. 2000). A thorough analysis of the Parlay call model is provided in Chapter 3.5.

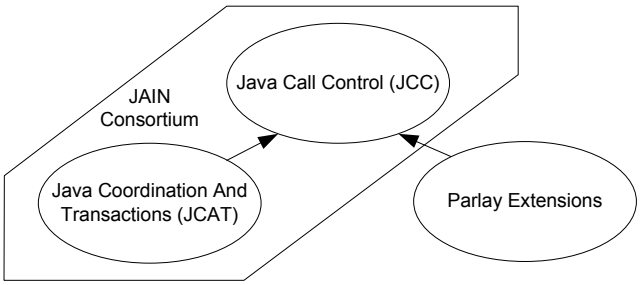


Figure 3.18: Relationship of JCC package to Parlay (Adapted from (Sun Microsystems 2002))

3.4.3 JCC Finite state Machines

Java Call Control borrows heavily from JTAPI for its core call models, with the view of creating consistency with the APIs, thus the basic call state model of JCC consists of four key objects a PROVIDER, CALL, CONNECTOR and ADDRESS object, as shown in Figure 3.19 (Sun Microsystems 2002):

Provider: As in JTAPI, the PROVIDER object allows the application to monitor call processing.

Call: This represents the call and is a dynamic collection of physical and logical entities representing the endpoints.

Connection: The CONNECTION object as in JTAPI represents the relationship between the CALL and an ADDRESS. A CONNECTION object exists as soon as an ADDRESS becomes part of the call, and is unique for that particular call.

Address: The ADDRESS represents a static logical endpoint such as telephone number or uniform resource identifier.

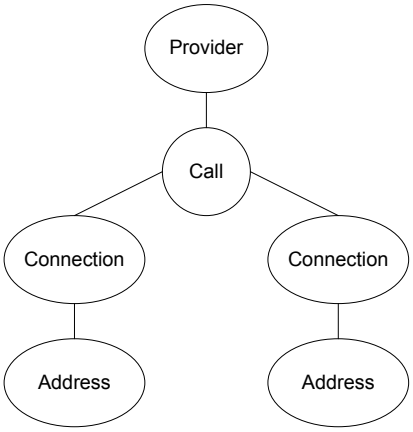


Figure 3.19: JCC Basic Call State Model (Adapted from (Sun Microsystems 2002))

Multiple parties are represented as additional CONNECTION and ADDRESS objects associated with the same CALL, with no limitations on the number of parties in the model. The JCC model as in the JTAPI model is symmetric as there is no distinction between the originating and terminating finite state machines, only the validity of transitions are affected. Unlike the JTAPI model which facilitates a half call model, the JCC call model is a full call model as the application has a complete view of all the parties of the call (as complete as is possible).

Provider Finite State Machine

As in JTAPI the PROVIDER represents the software application that is interfacing with the telephony subsystem, and the PROVIDER object (JccProvider) has the finite state machine shown in figure 3.20 (Sun Microsystems 2002). The JCC PROVIDER finite state machine is the same as for JTAPI. The states as defined in Sun Microsystems (2002) are as follows:

InService: The JCC PROVIDER is currently instantiated and available for use.
Out of Service: The JCC PROVIDER is temporarily not available for use, and some methods are invalid. The PROVIDER may come back to `InService` at any time, however, the application can take no direct action to cause this change.
Shutdown: The JCC PROVIDER is permanently unavailable for use, and almost all methods are invalid. Applications may use the shutdown() method to cause a PROVIDER object to move into the `Shutdown` state.

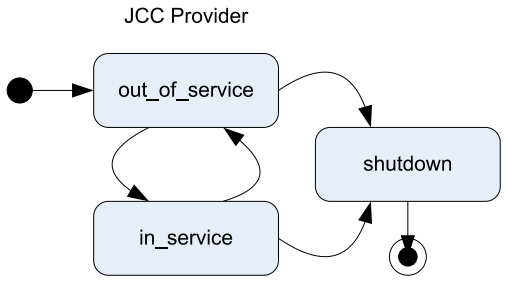


Figure 3.20: JCC Provider Finite State Machine (Adapted from (Sun Microsystems 2002))

Call Finite State Machine

The CALL object is an association of zero or more ADDRESSES. The CALL object (JccCall) has the finite state machine shown in Figure 3.21 (Sun Microsystems 2002). Due to the symmetrical full call model, from JCC version 1.1, a call may be controlled by any of the parties involved in the call (Jain et al. 2005, pg. 120). This finite state machine is slightly different from JTAPI in that a transition from `Idle` to `Invalid` is possible. The states as defined in Sun Microsystems (2002) are as follows:
Idle: In this state the CALL object has no connections.
Active: In this state the CALL object has one or more CONNECTIONS and there is ongoing activity.
Invalid: This state is entered when a CALL has lost all of its CONNECTION objects, and no further action is required.

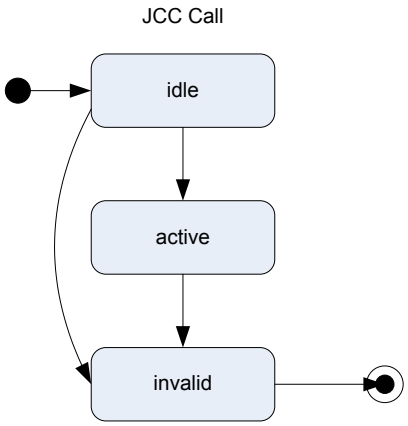


Figure 3.21: JCC Call Finite State Machine (Adapted from (Sun Microsystems 2002))

Connection Finite State Machine

The JCC CONNECTION object describes the relationship between a JCC CALL object and a JCC ADDRESS object. When a CONNECTION object moves to a `Disconnected` state the application would lose its reference to the object and the object would be subsequently destroyed. The finite state machine of the CONNECTION object is a refinement of the JTAPI Core Call model, and more closely resembles the IN BCSM. In particular the `Inprogress` and `Connected` states are expanded, as is the case with the JTAPI Call Control Extensions. The states are as follows:

Idle: This is the initial state for all new CONNECTIONS, not yet actively part of a telephone call. References to CALL and ADDRESS objects are valid in this state. As the `Idle` state is only the initial state for a connection, and a transition to another state occurs quickly.

Disconnected: A `Disconnected` CONNECTION is no longer part of a telephone call even though the references to the object remain valid. A `Disconnected` CONNECTION had to previously belong to a CALL.

Authorise Call Attempt: Authentication is required in order for the call processing to continue. This state is applicable to both originating and terminating connections.

Address Collect: In this state information regarding the initial information package is examined to determine if a complete destination address has been provided for the call.

Address Analyze: In this state the information collected is used to determine the routing of the call and type of call to continue.

Call Delivery: The function performed in this state is dependent of whether the connection is an originating or terminating connection. If the connection is originating then this state selects the route through the network as well as the connection delivery notification, however, for terminating connections this state is responsible for checking the state of the terminal and informing it of an incoming call. Thus in this state the signalling network used

for the call control is busy processing the call through the network.

Alerting: This state occurs when the terminating endpoint is being notified of the incoming call.

Connected: The CONNECTION and its associated ADDRESS are actively part of a telephone call. Each ADDRESS of the CALL would have its CONNECTION object in a `Connected` state.

Failed: The connection to the associated endpoint of the call has failed. Possible reasons include the endpoint being busy. CONNECTIONS that are in the `Failed` state are still connected to the CALL.

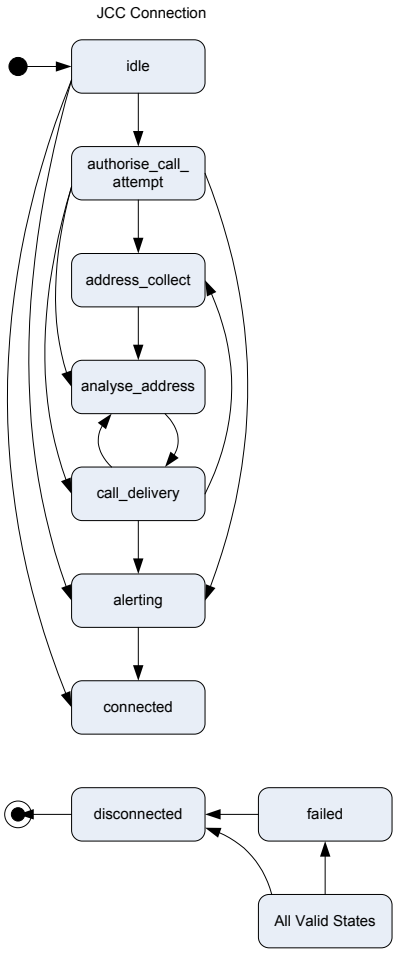


Figure 3.22: JCC Connection Finite State Machine (Adapted from (Sun Microsystems 2002))

3.4.4 Java Call Control Extensions (JCAT)

Java Call Control Extensions is an extension to JCC, and provides advanced features such as transfer and conferencing (Sun Microsystems 2003), mainly in the way of additional methods, rather than an extended finite state machine. As JCAT is an extension to JCC, the finite state machines are identical for the JCAT PROVIDER and JCAT CALL objects, and similar for the JCAT CONNECTION object. However JCAT does add additional objects to the call model not in JCC, such as JCAT TERMINAL and JCAT TERMINALCONNECTION. The JCC/JCAT call model is shown in Figure 3.23.

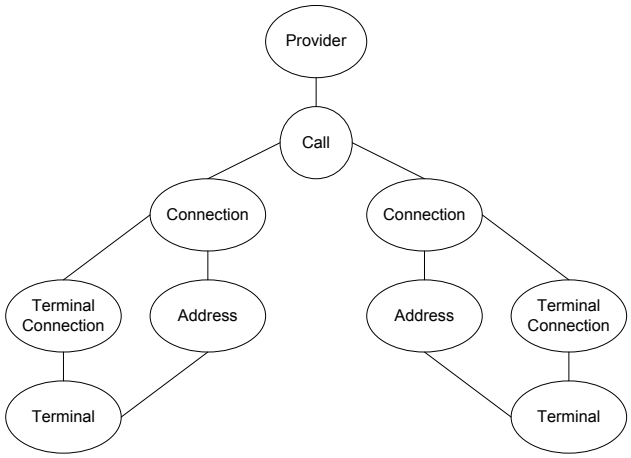


Figure 3.23: JCC/JCAT Basic Call State Model (Adapted from (Sun Microsystems 2002))

3.4.5 JCAT Finite State Machines

JcatProvider

The JCAT PROVIDER object extends the JCC PROVIDER and has the same finite state machine as the JCC PROVIDER.

JcatCall

The JCAT CALL object extends the JCC CALL, and provides additional call features such as the ability to transfer a call or set up a conference.

JcatConnection

The JCAT CONNECTION extends the JCC CONNECTION, and enhances it with additional objects, such as the JCAT TERMINAL and JCAT TERMINALCONNECTION objects. This additional association allows services which require multiple terminals per address or multiple addresses per terminal.

The JCAT CONNECTION object is associated with a CALL, ADDRESS, and TERMINAL-CONNECTION object, and maintains the state between the CALL and ADDRESS objects and the CALL and TERMINAL objects. Detected events are reported to the application, and each CONNECTION object is associated with one CALL and may not be reused. Due to the call model being symmetric, the states transversed depend on whether the endpoint is terminating or originating. The finite state machine describes the allowable transitions, and the API enforces these transitions. The CONNECTION finite state machine is slightly extended, with the JCC CONNECTION `Connected` state being separated into a `Connected` and `Supended` state, which allows for additional call processing. There is a direct mapping between states in order to ensure compatibility and permits applications to view either the JCC CONNECTION state or the JCAT CONNECTION state and remain consistent.

The CONNECTION finite state machine is shown in Figure 3.24 (Sun Microsystems 2003). The additional states added by JCAT are shown as shaded.

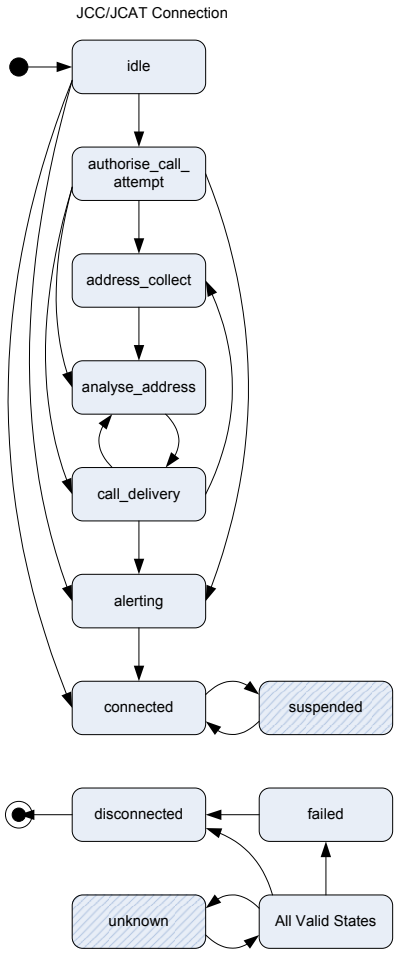


Figure 3.24: JCAT Connection Finite State Machine (Adapted from (Sun Microsystems 2003))

JcatTerminalConnection

The JCAT TERMINALCONNECTION object maintains the state between the TERMINAL and the CONNECTION. Whereas the JCC CONNECTION object reflects the relationship between the endpoint JCC ADDRESS object and JCC CALL, the JCAT TERMINALCONNECTION describes the relationship between the TERMINAL object and its CONNECTION. Different terminals on the same CONNECTION may have different states, and each is dependent on the state of the CONNECTION.

The JCAT TERMINALCONNECTION finite state machine is shown in Figure 3.25 Sun Microsystems (2003).

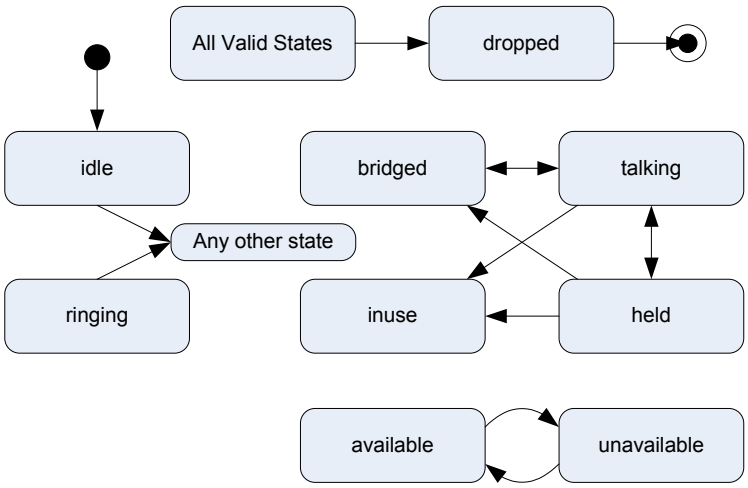


Figure 3.25: JCAT Terminal Connection Finite State Machine (Adapted from (Sun Microsystems 2003))

Idle: This is the initial state. A transition out of this state occurs quickly.
Ringing: This state indicates an incoming call and that the terminal is ringing.
Dropped: This state indicates that the terminal has permanently left the call, and is no longer referenced by the CONNECTION or TERMINAL objects. The TERMINALCONNECTION however still maintains its references to the CONNECTION and TERMINAL object for reference.
Bridged: This state indicates that the terminal is currently bridged into a call, but not yet active.
Talking: The terminal is actively part of a call, and the parties are communicating.
Inuse: This represents the terminal resources currently in use, and not available to be associated to the CALL object.

Held: This state represents the terminal being part of a call, but on hold. Only terminals actively part of a call may go on hold, and had to have been in the `Talking` state before.

3.4.6 Conclusion

JCC/JCAT introduces call control for not only PSTN but also multi-party multi-protocol sessions. Seen as incorporating concepts of both IN and JTAPI, JCC/JCAT adopts a converged view to the network, with multiple domains and parties. Independence of the underlying network and suitability to both circuit switched and packet switched networks are requirements for Extended Call Control as outlined in Chapter 1.1. This makes JCC/JCAT an important technology to consider when determining the Extended Call Control model. JCC provides facilities for observing, initiating and manipulating calls, including mid call operations. JCAT extends JCC to provide call control more closely aligned to that of IN, including a terminal finite state machine. Extended Call Control requires the control of multiple parties, and JCC/JCAT has no limitation on the number of parties in the model. JCC/JCAT uses a full call model to provide the controlling application with a complete view of all the parties of the call. The JCAT extension to JCC provides additional functionality mainly in the way of additional methods, rather than an extended finite state machine. This leads to the concept that should sufficient information be available in the call model, the API can be extended or reduced based on the required level of complexity, with the finite state machine describing allowable transitions and the API enforcing such transitions. JCC/JCAT aligned itself with OSA/Parlay Multi-Party Call Control and as such the OSA/Parlay call model is of interest for the development of an Extended Call Control call model.

3.5 Parlay State Models

To facilitate the third party development of services the Open Service Architecture (OSA) and Parlay service architecture were standardised with respect to APIs to abstract the network equipment. This standardised abstraction of the equipment made it possible to create services that communicated with the equipment without the knowledge of the detailed, often proprietary, protocols and signalling complexity of the equipment (Vannucci and Hanrahan 2005a). The Parlay and Open Service Architecture (OSA) standards converged into a common standard as a result of the similarities between the two service architectures, and the Parlay Group was absorbed into the Open Mobile Alliance.

Parlay has defined four call control APIs: Generic Call Control, Multi-Party Call Control,

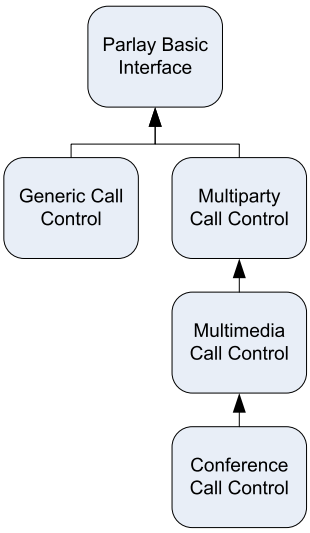


Figure 3.26: OSA/Parlay Call Control Inheritance (Adapted from (Jain et al. 2005, pg. 163))

Multi-Media Call Control, and Conference Call Control. The inheritance of the Call Control APIs are shown in Figure 3.26. Conference Call Control is not available in the OSA suite.

The Parlay APIs are application-centric, i.e. only the parts of the call that are of specific interest to the application are defined within the call model (Vannucci and Hanrahan 2005a). Application-centric call models are often simpler than network-centric ones and the objects may be deleted if no further control of the call is required (Vannucci and Hanrahan 2005a).

3.5.1 Generic Call Control

The Generic Call Control API contains all the basic methods necessary to manage a two-party call. The Generic Call Control was designed with Capability Set 1 PSTN interworking in mind, and as such does not support third-party initiated sessions, but provides functionality to allow call routing and management for the Intelligent Network. The API facilitates only basic services and provision for more than two parties is not supported, as shown in Figure 3.28. This lack of multiple parties is an inhibiting factor to the extent to which Generic Call Control can contribute to the Extended Call Control call model. The Generic Call Control Service (GCCS) is based around a third party model, which allows calls to be started from the network (ETSI and Parlay 2005a).

The Parlay Call Control Working group together with JAIN, ETSI and other involved parties has focussed on the Multi-Party Call Control and Multi-Media Call Control APIs, and these are greatly improved from the Generic Call Control. Thus, the joint call control group decided that the Multi-Party Call Control API was to be considered as the future base call

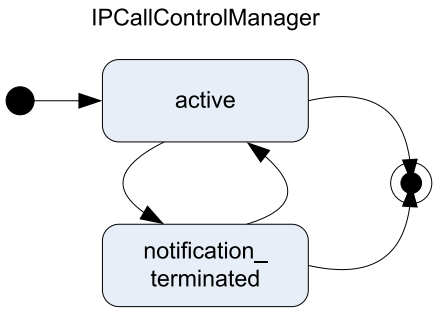


Figure 3.27: Generic Call Control Manager FSM (Adapted from (ETSI and Parlay 2005a, pg. 46))

control and technical work on the Generic Call Control was suspended.

The GCCS is represented by the IPCALLCONTROLMANAGER and the IPCALL interfaces, the application implements call back interfaces for responses and reporting.

IpCallControlManager

Figure 3.27 shows the application view of the GCCS IPCALLCONTROLMANAGER (ETSI and Parlay 2005a, pg. 46), and has the following states:

- **Active:** This state indicates a relationship has been established between the application (service logic) and the GCCS. `Active` is defined in (ETSI and Parlay 2005a, pg. 30) as "being routed or connected."
- **Notification Terminated:** In this state the GCCS will no longer forward event notifications to the application, and will not accept any requests for new notifications.

IpCall

Figure 3.28 shows the application view of the IPCALL object (ETSI and Parlay 2005a, pg. 47), which has the following states:

- **No Parties:** In this state the IPCALL object is created. The application can set the charging for the call and request charging information updates.
- **Active:** In this state a call between two parties is being set up, or exists. If a network event related to the call occurs, processing is suspended and the application has to resume the call. There are a number of sub states:

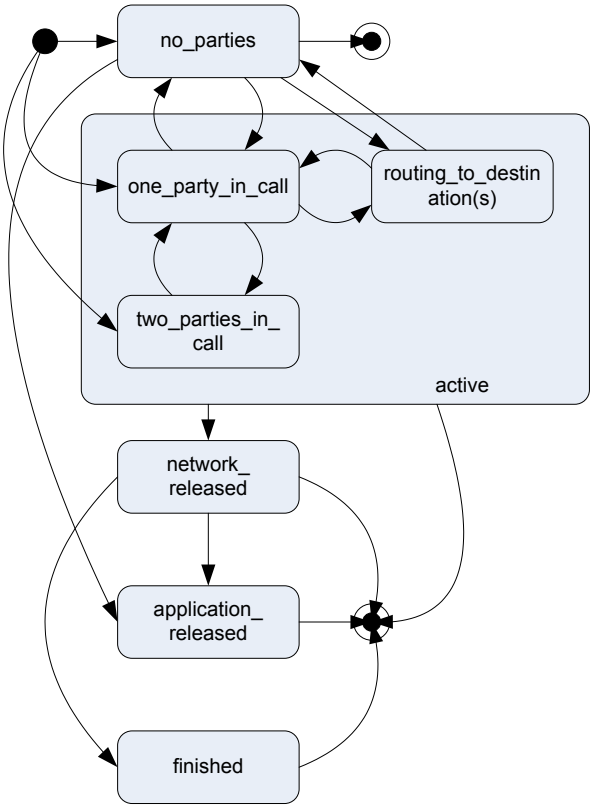


Figure 3.28: Call FSM (Adapted from (ETSI and Parlay 2005a, pg. 46))

– One Party In Call: Only one party is associated with the call. The charging plan for the second party is setup before the application can request a connection to the second party. Additional digits can be collected in this state as user interaction is possible. When the second party answers a transition to the state `Two_Parties_In_Call` is made.

– Two Parties in Call: There is a connection between both parties. In this state user interaction is still possible. Once the second party disconnects, there are three possible transitions:

  1. If the application requested to be notified of a disconnect and for the call processing to be interrupted then a transition back to the `One_Party_In_Call` state is made and the application has control of the call.
  2. In the case of the application requesting just a notification but no interrupt then a transition to `Network_Released` occurs.
  3. If the application is not monitoring a disconnect then the call is ended and a transition to the `Network_Released` state occurs and the application

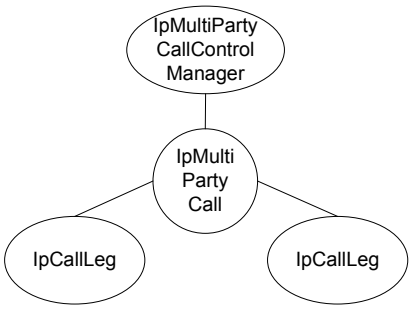


Figure 3.29: Multi-Party Call Control Call Model

is notified of the call ending without additional information being given.

– Routing to Destination: In this state there is an outstanding routing request for the second party connection. Generic Call Control assumes that all parties are contacted synchronously, with the first party being the "originating" party.

- **Network Released:** In this state the call has ended and the necessary requested information is returned to the application. In the case of no further information being required a transition to `Finished` is made.
- **Finished:** In this state the call has ended and no further information is returned to the application. The gateway waits for the application to request the destruction of the IPCALL object, as it is expected the application follows good object orientated practice and does its own garbage collection.
- **Application Released:** In this state the application has requested the release of the IPCALL object and the requested information is returned to the application and the IPCALL object is destroyed.

3.5.2 Multi-Party Call Control

The Multi-Party Call Control API enhances the functionality found in the Generic Call Control Service, allowing control of the individual legs in a call, and providing multi-party call functionality. The Multi-Party Call Control Service is represented by the IPMULTIPARTYCALLCONTROLMANAGER, IPMULTIPARTYCALL, and IPCALLLEG objects as shown in the finite state machine in Figure 3.29. Multi-Party Call Control supports third party initiated sessions, and does not inherit from the Generic Call Control API as the decision was made to avoid having changes to the Multi-Party Call Control affecting the Generic Call Control (Vannucci and Hanrahan 2005a).

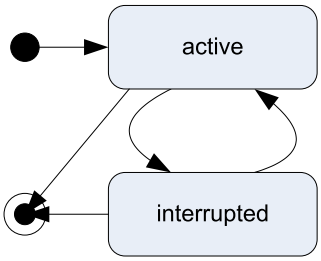


Figure 3.30: Multi-Party Call Control Manager FSM (Adapted from (ETSI and Parlay 2005b, pg. 52))

IpMultiPartyCallControlManager

The Multi-Party Call Control Manager provides the management of the service. The application uses this interface to create IPMULTIPARTYCALL objects, enable and disable notifications and provide overload functionality (ETSI and Parlay 2005b, pg. 26). Figure 3.30 shows the application view of the IPMULTIPARTYCALLCONTROLMANAGER object (ETSI and Parlay 2005b, pg. 52), with the following states:

- **Active:** A relation exists between the application and the service, and the application can request call related events. If the application requests information, the IPMULTIPARTYCALLCONTROLMANAGER will create the necessary IPMULTIPARTYCALL and IPCALLLEG objects.
- **Interrupted:** The IPMULTIPARTYCALLCONTROLMANAGER is unavailable for use by the application in this state. No notifications are forwarded by the IPMULTIPARTYCALLCONTROLMANAGER to the application and the application cannot invoke any of the API methods.

IpMultiPartyCall

The IPMULTIPARTYCALL allows the application to control and supervise the routing of the call legs, request information regarding the state of the call, set the call charging parameters, and release the call (ETSI and Parlay 2005b, pg. 35).

Figure 3.31 shows the application view of the IPMULTIPARTYCALL object (ETSI and Parlay 2005b, pg. 53), the associated states are:

- **Idle:** This state exists when the IPMULTIPARTYCALL object has no IPCALLLEG objects associated with it. As soon as the first IPCALLLEG object is created, a state

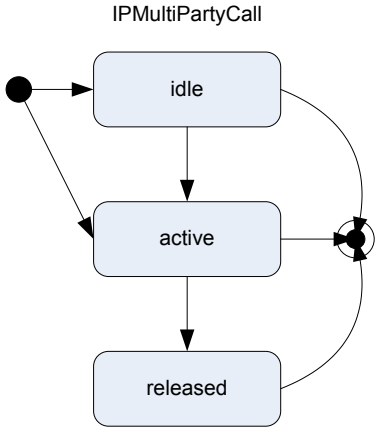


Figure 3.31: Multi-Party Call FSM (Adapted from (ETSI and Parlay 2005b, pg. 53))

transition is made to the `Active` state.

- **Active:** In this state the IPMULTIPARTYCALL object has one or more IPCALLLEG objects associated to it. The application may create additional IPCALLLEG objects up to the limit specified by the API implementation. The application can request call supervision and other call relation information such as charging reports and set call parameters such as charging plan.
- **Released:** The IPCALLLEG object has been released from the IPMULTIPARTYCALL or the IPMULTIPARTYCALL is released. Requested call information is returned to the application together with a notification that the call has ended and the IPMULTIPARTYCALL object transitions to the end (null) state ETSI and Parlay (2005b).

IpCallLeg

The Multi-Party Call Control API separates the call legs into two objects, one for the originating party and the second for the terminating party. This asymmetric model implementation has a number of implications, such as upstream events if one of the call legs is not immediately evident in the other call leg, and vice versa. In addition if an event causes a suspension in call processing, other events, which may happen at almost the same time, are handled in a sequential order by the application. There are cases where a state transition in one of the call leg objects results in a state transition in another call leg object, for example if the originating IPCALLLEG object went in to a `Terminating` state, this would also imply the terminating IPCALLLEG object would also transition to a `Terminating` state. The IPCALLLEG object associates the IPMULTIPARTYCALL object with an address, and

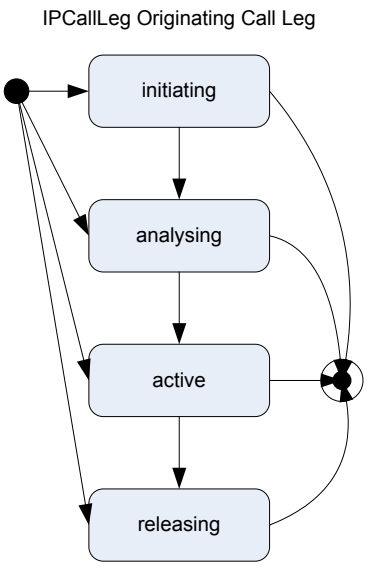


Figure 3.32: Multi-Party Originating Call Leg FSM (Adapted from (ETSI and Parlay 2005b, pg. 56))

represents the signalling relationship between the IPMULTIPARTYCALL and an address (ETSI and Parlay 2005b, pg. 41). For the originating IPCALLLEG finite state machine, the states `Active` and `Analysing` may be considered to be one state with sub states to perform digit collection (ETSI and Parlay 2005b, pg. 55).

The finite state machine for the Originating IPCALLLEG object is shown in Figure 3.32, the associated states are:

- **Initiating:** In this state the originating party authorisation to create a connection to the terminating party is confirmed. If the authorisation is confirmed then the connection is completed upon successful collection of the initial dialling information package.
- **Analysing:** In this state the IPCALLLEG object is responsible for the collection and analysis of the destination address provided by the calling party, further digits can be collected if necessary. The address analysis is performed according to the dialling plan to determine the charging and routing of the terminating call leg.
- **Active:** In this state there is a connection between the two parties and they are communicating, mid-call events can be received and the call suspended. Using the collected address information from the previous states, the authority of the calling party is confirmed and the application has to create and route the terminating IPCALLLEG. A transition out of this state occurs when a party disconnects or the application

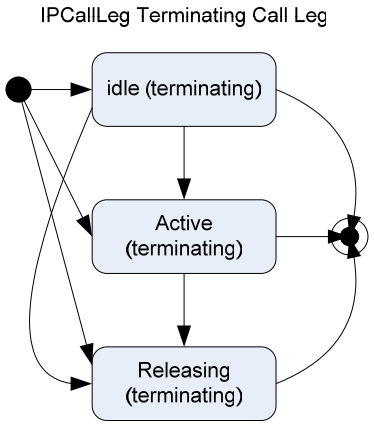


Figure 3.33: Multi-Party Terminating Call Leg FSM (Adapted from (ETSI and Parlay 2005b, pg. 64))

deassigns or releases the call.

- **Releasing:** In this state the connection to the called party is released either by the network or the application, and the relevant call information is returned to the application. If no further information is required by the application from the IPCALLLEG object, it is destroyed.

The finite state machine for the Terminating IPCALLLEG object is shown in Figure 3.33, with the states being:

- **Idle (terminating):** This state is entered when a notice to create the terminating party call leg is received from the application. The CALL LEG object is created and the interface connection is idle. Once the CALL LEG object receives a route request, a transition out of this state occurs.
- **Active (terminating):** Once the call leg is routed, the CALL LEG object becomes active. Before routing, authorisation of the calling party to create the leg is confirmed.
- **Releasing (terminating):** This state occurs when a network or application release occurs. If the terminating party is busy or does not answer then the network initiates the transition to the `Releasing` state. The relevant requested reports are sent to the application and the release event handling is performed and the IPCALLLEG object destroyed.

3.5.3 Conclusions

Parlay APIs are application-centric, with only the parts of the call that are of interest to the application defined within the API. Application-centric call models have the benefit of being deleted if no further call control is required by the application. This however assumes a tightly coupled implementation such as CORBA, where the application is extremely stable and there is a one-to-one matching between the application and associated state objects. Such is often not the case for a Web application, and this leads to the conclusion that lightly coupled systems should have a call model that can persist within the operators domain, should the Web application become unavailable. Generic Call Control does not fulfil the requirements laid out in Chapter 1.1, however does provide a glimpse as to how third party call control is handled for two parties only, in that call legs are often set up synchronously within the network, with the first party being the "originating" party. Multi-Party Call Control provides control of individual legs in a call, by representing the state of each leg with a separate finite state machine. The states within the Call Leg finite state machine closely align to those of the IN BCSM,the however, does provide a greater level of abstraction, ensuring the suitability of the Parlay API to packet switched networks. The asymmetrical call model retains the complexity of the IN BCSM, and requires developers to be familiar with the process of call creation within telecommunications networks in general, despite not specifying the implementation.

3.6 SIP

The Session Initiation Protocol (SIP) is a protocol whose main benefits are seen as simplicity and its stateless nature (Johnston 2003). SIP was not intended to contain large amounts of state information, and rather is seen as mostly stateless, as in the case of many other Internet based technologies. Due to the increasing complexity of SIP messaging and interworking with other telecommunication systems, a requirement for the transferral and representation of state arose. As defined in RFC 3261, "A dialog represents a peer-to-peer SIP relationship between two user agents that persists for some time." (Rosenberg et al. 2002). Indeed the proxy systems within a SIP enabled network, play the part of routing and delivery platforms for services (Marshall et al. 2001), and these proxies are required to have state information to enable many of these services. The correlation of messages received and sent to user agents within the network allows this state to be reconstructed and maintained within proxy elements.

As proxies deal with a large number of calls, maintaining state information can overwhelm

some systems, and make load balancing problematic. The SIP Extensions for Distributed Call State proposed maintaining state in the SIP header, which would then allow proxies to maintain state in the SIP message headers as opposed to within their own systems, thus effectively rendering the proxy stateless. This solution does have drawbacks in that the size of the header is far larger than necessary, and subsequently all messages within the network are larger.

SIP is a transactional protocol, in that interactions between elements takes place in the form of a series of independent message exchanges (Rosenberg et al. 2002). These message exchanges consist of a single request and a number of responses, which can include zero or more provisional responses, and one or more final responses. RFC 3261 defines transactions depending on the message passed, an INVITE client transaction and a NON-INVITE client transaction (Rosenberg et al. 2002). The invite transaction, or dialogue, consists of a three-way handshake, with the client sending an INVITE, the server then responding, and finally the client acknowledging the response. The INVITE client transaction state machine in RFC 3261 is superseded by the dialogue state in RFC 4235. RFC 4235 identifies the need for multiple dialogues at a user agent due to the possibility of forking in which multiple terminals can be registered to a single address (Rosenberg, Schulzrinne and Mayh 2005). In RFC 4235 support of state dialogues was enabled by providing a greater number of states between the transmission of the INVITE dialogue and the creation of actual dialogues based on receipt of 1xx and 2xx response messages (Rosenberg, Schulzrinne and Mayh 2005). In addition to support of dialogue state before dialogues are fully instantiated, the state machine for the originating and terminating parties was combined, essentially changing the state machine from being an asymmetric call model to a symmetric call model.

The finite state machine for dialogue state is shown in Figure 3.34. This dialog finite state machine models the state from the initial INVITE message through to the disconnecting BYE message. The finite state machine is created upon receipt of an INVITE request message from a user agent client (UAC). All provisional responses have a tag for dialog identification, and if a response is received without such a tag then the finite state machine would transition to the `Proceeding` state as a full dialogue has not yet been created, due to only two of the three components being available, namely the call identifier and local tag. Once a 1xx response message with a tag is received (such as a 100 Trying) then the full dialog identifier is defined and a transition to the `Early` state can occur. Upon receipt of a 2xx message such as 200 OK the call can move to the `Confirmed` state.

If a 1xx or 2xx message with a different tag is received, then the finite state machine is recreated and initialised to the appropriate state, ensuring that the finite state machine is current and representative of the entire state of the call. If the UAC cancels the call, with

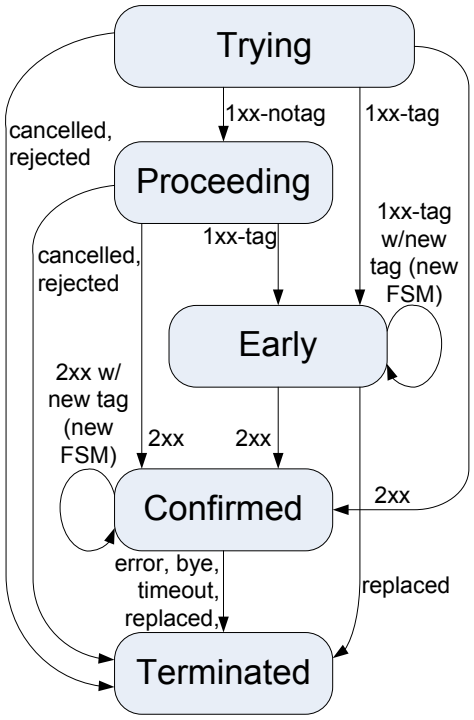


Figure 3.34: INVITE Dialogue State Machine (Adapted from (Rosenberg, Schulzrinne and Mayh 2005))

a CANCEL message, and a corresponding 487 Request Terminated is received, then the call transitions to the `Terminated` state, and indicates the call is cancelled by means of a 'cancelled' event. In the event of a new registration (replacing the original invite), the finite state machine transitions to the `Terminated` state, with the notification 'replaced'. If there is no positive response from the UAS for the INTIVE message then the call transitions to the `Terminated` state with the notification 'rejected'. In the case of an entity that is subscribed to receive notifications of state, actual dialog states across all dialogs are not reported, RFC 4235 defines a single 'virtual' dialog finite state machine, which is a cumulative summary of each dialog (Rosenberg, Schulzrinne and Mayh 2005).

From the `Confirmed` state the call can either end (`Terminated` state) by means of a BYE message, or end by means of a mid-dialogue error message such as a 481 Dialogue Does Not Exist or a 408 Request Timeout. Note that the SIP standards prohibit sending of a CANCEL message until a provisional response is sent (Rosenberg, Schulzrinne and Mayh 2005). RFC 4235 recommends that notifications of state for a single subscriber are not generated faster than once per second (Rosenberg, Schulzrinne and Mayh 2005).

3.6.1 Conclusions

The SIP dialog provides a mechanism for the maintenance of state between SIP user agents and servers. The concept of a dialog was necessary for long lived connections, that persist for a period of time. This parallels the requirement of a call model for Extended Call Control as outlined in Chapter 1, where control of a call is for the entire duration of the call. RFC 4235 identifies the need for multiple dialog states per terminal registered to an address. The Extended Call Control Web service abstracts the state of the terminal which is involved in the call, as opposed to unused terminals. The change from an asymmetric call model in RFC 3261 to a symmetric call model in RFC 4235 is important to note for the Extended Call Control call model, in which simplicity is a key requirement. The transitions within the SIP dialog are frequent, with a new dialog being created each time a new registration is received. This reinforces the requirement of Extended Call Control asynchronous notification of state, as outlined in Chapter 1. The recommendation of RFC 4235 with regard to notifications for the state of a single subscriber not exceeding one per second indicates the volume of state messaging that an Extended Call Control Web service can have, with control of multiple parties. The SIP dialog is a half call model with each user having a state dialog based on sent and received messages with tags for dialog identification. The single 'virtual' dialog finite state machine as a means of summarising multiple dialog state is useful in defining priority of call states with the Extended Call Control call model. The lack of mapping between the SIP dialog and existing telecommunications call models indicates that whilst the SIP dialog is useful to packet switched networks, it cannot form the sole basis for a Web based call model. The subsequent section on mapping aligns the SIP dialog with the Parlay and IN call models, using Parlay as a common denominator.

3.7 Call Model Mapping

In this section it is shown how the Parlay call model maps to an IN circuit switched model, as well as an IP SIP based packet switched model.

3.7.1 OSA/Parlay-IN CS1 Mapping

In order to provide a mapping from the underlying network call state model to a corresponding OSA/Parlay MPCC IPCALLLEG object, this research considers IN Capability Set 1 (CS1), as this demonstrates the principle of operation, which can be extended for more

Table 3.1: Originating Detection Points and OSA/Parlay MPCC Notification Mapping (Adapted from (Vannucci and Hanrahan 2006))

IpAppMultiPartyCallControl-Manager.reportNotification-(Originating_Call_Attempt_Authorised)	Orig_Attempt_Authorised
IpAppMultiPartyCallControlManager.-reportNotification(Address_Collected)	Collected_Info
IpAppMultiPartyCallControlManager.-reportNotification(Address_Analysed)	Analysed_Info
IpAppMultiPartyCallControlManager.-reportNotification(Answer)	O_Answer
IpAppMultiPartyCallControlManager.-reportNotification(Redirected)	O_Mid_Call
IpAppMultiPartyCallControlManager.-reportNotification(Originating_Release)	Route_Select_Failure O_Called_Party_Busy O_No_Answer O_Disconnect O_Abandon

complex call models. Mapping from the CS1 BCSM to the OSA/Parlay MPCC IPCALLLEG object state transition is performed considering the state entry and exit events. Not all notifications received from the CS network correspond to a transition. OSA/Parlay has defined a limited set of notifications from the MPCC Manager to the corresponding Call Leg objects.

Both the CS1 call state model and the OSA/Parlay Multi-Party Call Control call state model have half call models, separating the originating and terminating cases. Many of the CS1 detection points correspond closely with OSA/Parlay notifications. The call state mapping for the originating basic call state model is shown in Figure 3.35. Note that the OSA/Parlay call state model has been simplified for the sake of clarity and does not show all the possible transitions. The CS1 call state model detection points can be mapped as shown in Table 3.1. The call state mapping for the terminating basic call state model mapping is shown in Figure 3.36, again a simplified state model is shown. CS1 BCSM detection points can be mapped as shown in Table 3.2.

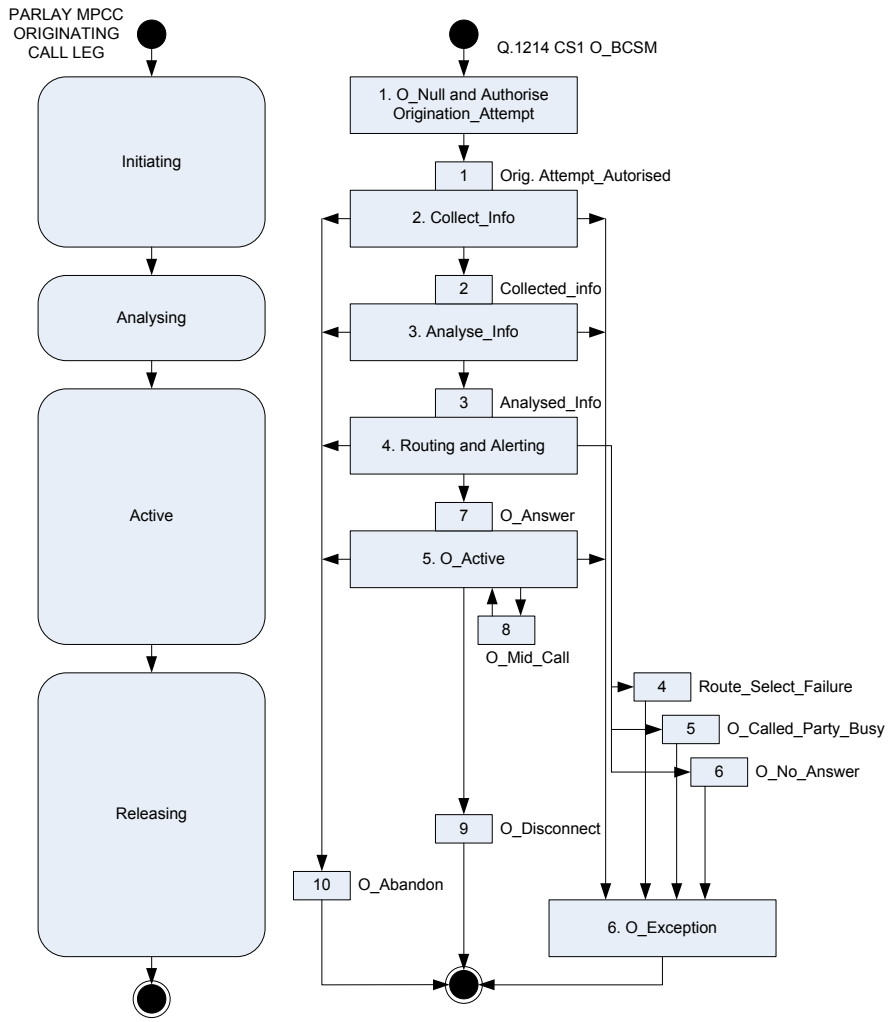


Figure 3.35: CS1 Originating BCSM and OSA/Parlay MPCC Originating Call Leg Mapping (Adapted from (Vannucci and Hanrahan 2006))

Table 3.2: Terminating Detection Points and OSA/Parlay MPCC Notification Mapping

IpAppMultiPartyCallControl-Manager.reportNotification-(Terminating_Call_Attempt_Authorised)	Term_Attempt_Authorised
IpAppMultiPartyCallControlManager.-reportNotification(Answer)	T_Answer
IpAppMultiPartyCallControlManager.-reportNotification(Terminating_Release)	T_Called_Party_Busy T_No_Answer T_Disconnect T_Abandon

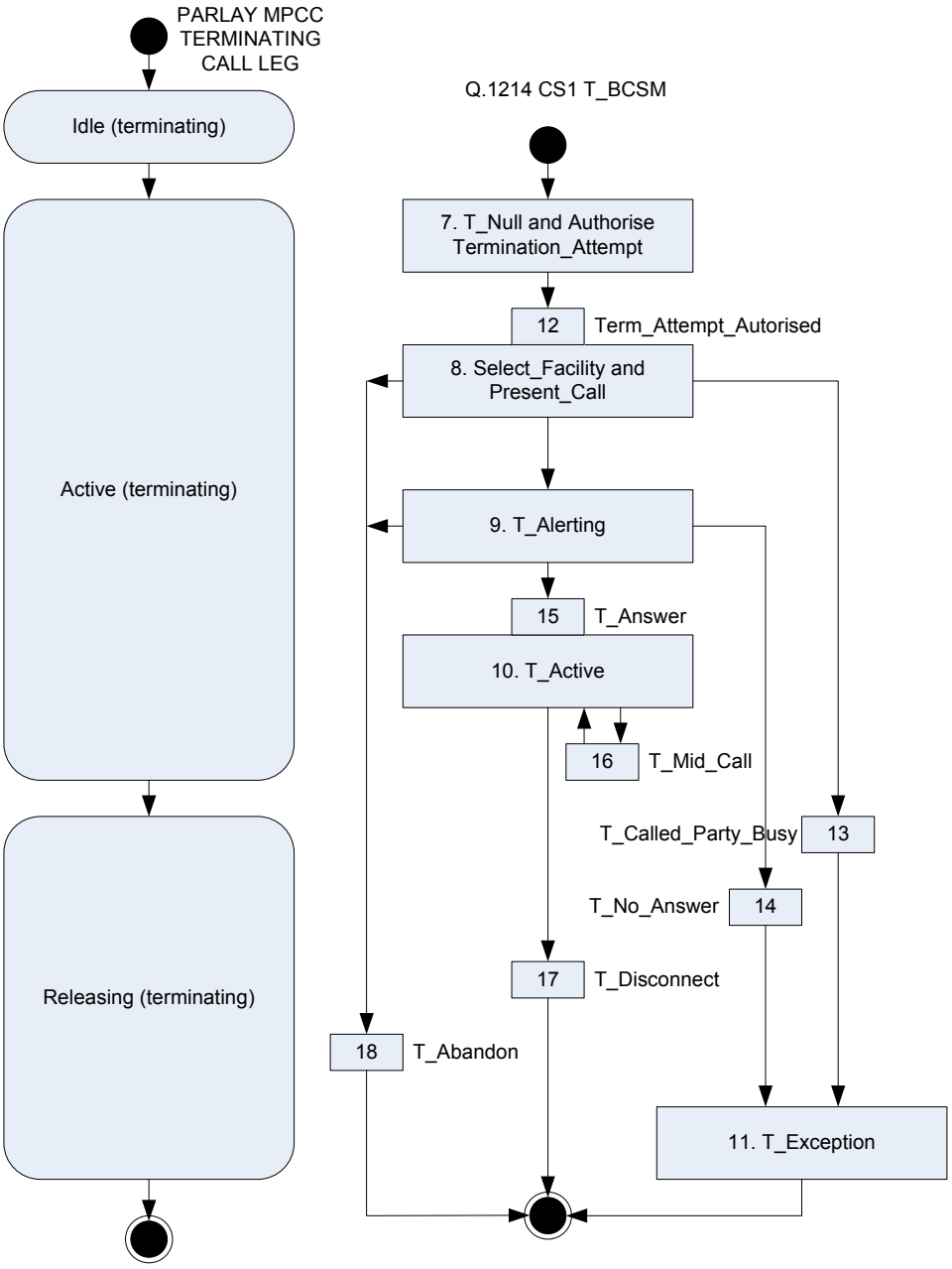


Figure 3.36: CS1 Terminating BCSM and OSA/Parlay MPCC Terminating Call Leg Mapping

3.7.2 OSA/Parlay-SIP Mapping

SIP Dialogs are used to model the state of SIP calls, in particular the INVITE initiated dialog. In an Internet Draft by Rosenberg, Schulzrinne and May (2005) a dialog finite state machine is presented that combines the state for both the case of the originating UAC and terminating UAS. This dialog models state from the initial INVITE message through to the disconnecting BYE message. It is assumed that all provisional responses have a tag for dialog identification. The `Proceeding` state can then be ignored, and any provisional 1xx response causes a transition to the `Early` state, as shown in Figure 3.34.

A mapping is required between the SIP state machine and the OSA/Parlay state machine if OSA/Parlay is to provide services as well as service interworking between the IMS network and the CS network. Thus a mapping is defined as shown in figure 3.37. The MPCC originating states `Initiating` and `Analysing` correspond to the SIP dialog state `Trying`, since an INVITE message causes a transition to `Trying`, and similarly CreateAndRouteCallLeg() originates a call leg. The `Early` and `Confirmed` states correspond to the MPCC Call Leg `Active` state, as a provisional SIP response corresponds to a OSA/Parlay Address Analysed or Answer notification. The mapped states do not overlap and clear boundaries are preserved in the mapping as recommended by Dobrowolski, Montgomery, Vemuri, Voelker and Brusilovsky (1999a).

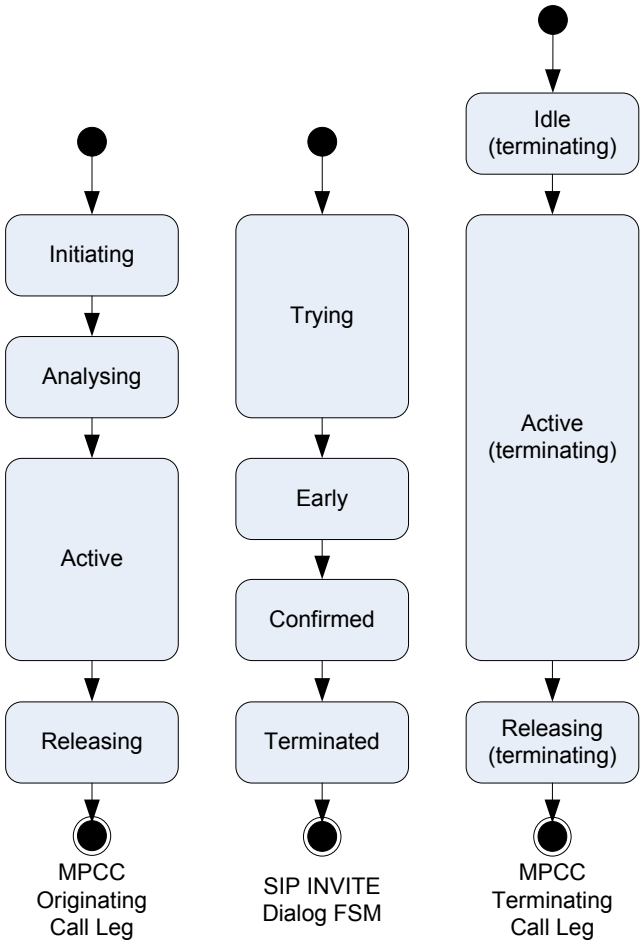


Figure 3.37: SIP INVITE Dialog FSM and OSA/Parlay MPCC Mapping

3.8 Conclusion

Advanced Web based call control has a number of requirements for the call model, as outlined in Chapter 1.1. The Extended Call Control call model aims to provide a suitable level of abstraction for use in a Web environment, whilst containing sufficient information to interwork with existing call models. In addition a complete view of the connection is also required. The Extended Call Control (ECC) call model aims to provide a suitable level of abstraction of a telecommunications network for use in a Web environment. These requirements necessitate a review of existing telecommunication networks call models in order to determine best practices that should appear in the ECC Call Model.

The theory of call models was reviewed with respect to the party controlling the call, the detail of the call model for involved parties, and the states contained within the call model for each party. The following call models were reviewed:

- The IN basic call state model was analysed as it forms the basis for all telecommunications call models.
- The JTAPI call model is useful as it provides a simplified call model intended for private branch exchange equipment.
- The JCC/JCAT call model applies to not only circuit switched but also packet switched multimedia multi-party sessions.
- The OSA/Parlay APIs provide complex services, whilst abstracting the underlying network equipment (be it for packet switched or circuit switched).
- SIP dialogues are analysed with respect to call models for purely packet based communication.

Mappings between circuit switched and packet switched call models by means of the Parlay call model show how interoperation of all call models is possible.

The useful features of each call model reviewed in this chapter are synthesised in the following chapter to create the Extended Call Control call model meeting the requirements in Chapter 1.1.

Chapter 4

Design of the Extended Call Control Call Model

As discussed in Chapter 3, advanced call control requires knowledge of the state of the resources within the network. Telecommunications service architectures use call state models to keep track of the progress of each session and the services that are used during the sessions. The call model represents all the essential features of a session (Jain et al. 2005), and is a high level, technology independent abstraction of the call (Graf 2000; Vannucci and Hanrahan 2007a).

Telecommunication services, such as conference calling and call centre queueing, require many messages to be exchanged and control of the call is usually in place for the duration of the call. These services usually require the application to implement a call model to interpret these messages based on the last known state of the session (Dobrowolski, Montgomery, Vemuri, Voelker and Brusilovsky 1999b). Thus, implementation of a call state model and an asynchronous Extended Call Control Web service allows for a far richer set of service functionality than that offered by simple request response type messaging (Vannucci and Hanrahan 2007a).

4.1 Extended Call Control Connection Model

Chapter 3 identified a number of features of a call model; calls within a network can be thought of consisting of three entities: a device with which the user communicates through, a connection through which all information flows, and the state of the connection within the network. Both JTAPI and JCAT use the concept of a connection object to describe call state, a terminal connection object to describe the state of the terminal, and a call object

to describe the state of call. The Parlay Multi-Party Call replicates similar characteristion of state, with an IPMULTIPARTYCALL object and an IPCALLLEG object and media to describe the state of the terminal with respect to the media session. The Parlay media can be seen as an abstraction of the state of the terminal as viewed by the network. These three entities can be related to each other by means of a connection model.

As outlined in Chapter 1 there are a large number of devices, each with different properties, and abilities. However, the direct control of the device is not considered in the Extended Call control Web service. The control of the call is the focus of this research and thus the control of the connections within the network is the focus of the Web service. To abstract control of each individual connection, the call is represented as a unified entity (as in RFC 4235 (Rosenberg, Schulzrinne and May 2005)), with notifications advising the Web application of changes in specific connections if required. The underlying network is assumed to be responsible for the negotiation of connectivity, and the selection of the appropriate media services.

The SIP proxy Dialog (Rosenberg, Schulzrinne and Mayh 2005) presents the overall state of the call as being the sum of its connections, with transitions occurring on specific events which can affect the entire conversation, such as attempting to contact another party. This concept of an abstracted call state representing multiple terminals at once is implemented in this research for the Extended Call Control (ECC) call model.

The ECC model abstracts the complexity of the state of the individual terminal connection (as is done with the Parlay Multi-Party Call Control) using media streams to indicate the state of the media and associated direction of flow. Thus abstracting the device the user communicates through as a media stream.

A connection model similar to that used in the Services for Computer Supported Telecommunication Applications phase 3 (ECMA 2004) is adopted to illustrate the relationship between the call object and associated connections of the call, as shown in Figure 4.1.

During the life of a call, connections within the network will transit through various stages. State transitions are observed by the Web service, and reported to the Web application as a event notification, the call state will update on a change in a connection state. Individual connection states are available as an added layer of detail.

Using the call model structuring concepts as presented in JTAPI, JCC/JCAT and Parlay, the ECC Call model consists of a CALL object, and CONNECTION objects. CONNECTIONS have the following attributes:

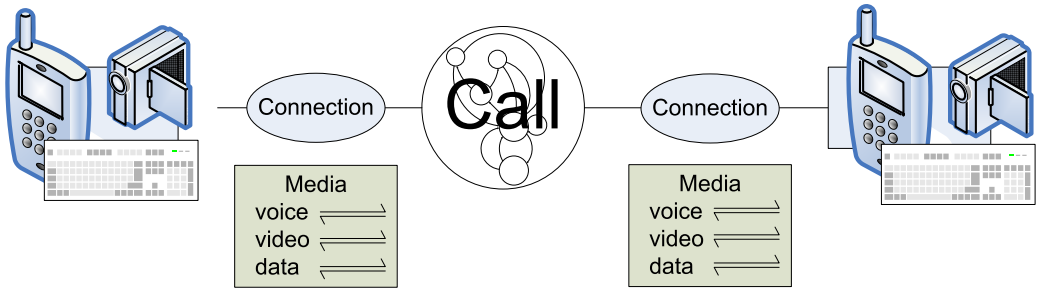


Figure 4.1: Extended Call Control Connection Model

- Connection Identifier - Each CONNECTION has a unique identifier for a given call. This identifier can be based on the address of the participant. There are as many CONNECTIONS as participants.
- Call Identifier - Each CONNECTION has a reference to the identifier of each call with which it is involved. One CONNECTION could be involved in more than one call, for example a conference call.
- Media Stream - This attribute is as defined by Parlay Multi-Media Call Control SCF. A Media stream has a data type, being either voice, video, or data, with an associated flow direction, either bidirectional or unidirectional.
- State - Depending on the level of abstraction required by the Web application this state is updated as notifications are received from the network. If a high level of abstraction is required, this is the same as the state of the CALL object, and can be disregarded.

4.2 The Extended Call Control Call State Model

The proposed call state model, developed in this research, for the CALL object in the CONNECTION model is shown in Figure 4.2. State transitions are observed by the service logic through event reports, caused by either service logic instructions or network notifications.

The following are the connection state definitions:

- `Inactive` - In this state the Web application is registered with the Extended Call Control Web service, however no connections exist. Service logic is waiting for either a Web application instruction or a network notification. Relevant network triggers have been created within the underlying network. A transition to `Inactive` occurs when there are no longer any connections involved in the call.

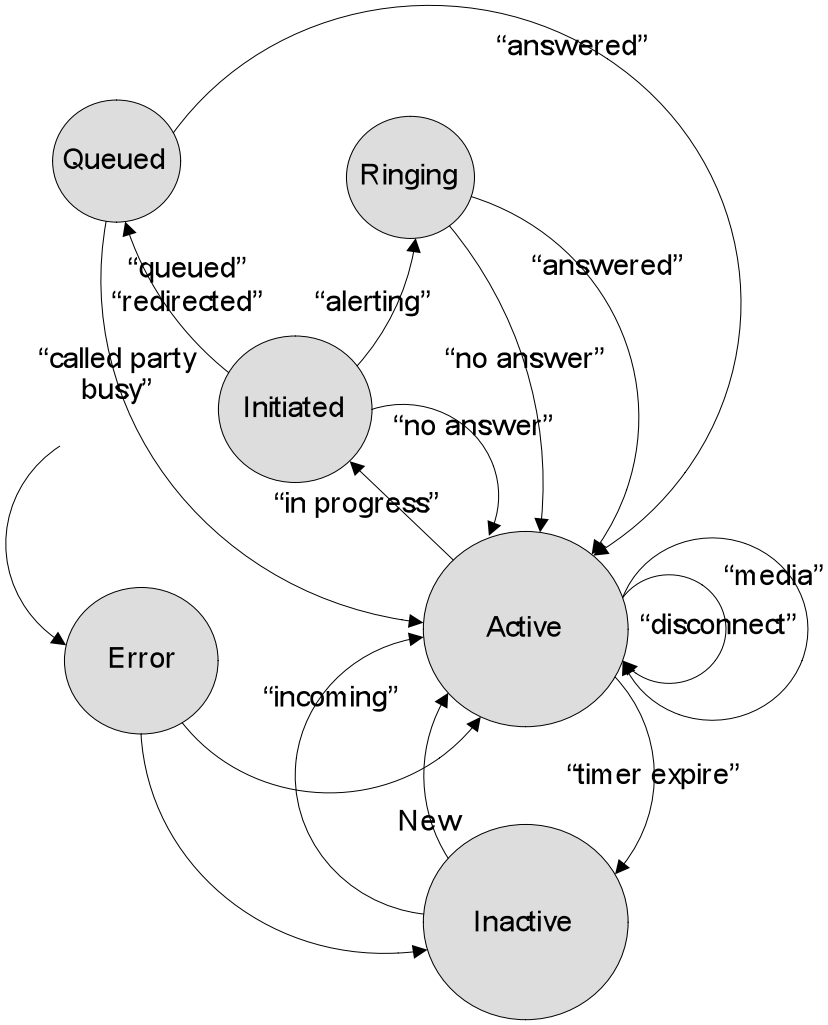


Figure 4.2: Extended Call Control Call Model

The JCC CALL object, as shown in Figure 3.21, represents the relationship between the controlling software application and connections associated to a call (also common to the Parlay Multi-Party call finite state machine (FSM), as shown in Figure 3.31). The idle state of the JCC CALL object FSM occurs when no connections are associated with the call. Likewise the Parlay Multi-Party Call FSM is `Idle` when no connections exist. An `Inactive` state is proposed for the Extended Call Control model and is representative of there being no connections in a call. A transition out of the `Inactive` state occurs with an incoming call notification is received and causes a transition to the `Active` state. If the call is released or once all individual connections are released from the call, a transition is made back to the `Inactive` state after a timeout.

- `Active` - Considering the state of the JCC/JCAT CALL FSM and JCC/JCAT CONNECTION FSM, the CALL FSM `Active` state occurs when the CALL has one or more CONNECTIONS associated with it and there is ongoing activity. The CONNECTION FSM `Connected` state occurs when the connections are actively part of a telephone call. This is again present in the case of the Parlay MPCC CALL FSM `Active` state where the call has call legs associated with it, and the MPCC IPCALLLEG FSM `Active` state where there is a connection between communicating parties. These states can be abstracted such that there may be one or more connections which have a logical relationship between the call and service logic. The state of the media stream between parties is represented by the media of the connection. Thus, the ECC Call Model includes the `Active` state to signify that there is a relationship between the call and the service logic. A change in a connection would cause an update in state information. This event may be responsible for a transition of the overall call state.

  The `Active` state is the default state of a call with connections that are communicating. A transition out of the `Active` state occurs when an additional connection is in progress, or all connections are disconnected. If a connection disconnects, a notification is received; however, the overall call state would not change unless the last connection disconnected. Changes in media will result in a notification if requested, however not a change of state. As a CONNECTION is added to the CALL, the state transitions to `Initiated`.

- `Initiated` - A state in which the network is in the process of establishing a connection. In this state the call is in a pre-delivery state and service logic can alter the routing of the connection. Only once the connection is fully established would the call state transition to RINGING or QUEUED. In the case of multiple connections being created simultaneously the call will transition to QUEUED once a connection is queued, likewise once a connection is ringing the call state would transition to RINGING.

  The JTAPI `In_Progress` state abstracts the processes that occur before the network alerts the user of an incoming call. JCC/JCAT provides a large number of states for the call authorisation, the collection of the address, the analysis of the address and the routing through the network, before finally transitioning to the alerting state. The Parlay originating IPCALLLEG model includes the `Initiating` and `Analysing` states before transitioning to an `Active` state after the address is analysed, likewise the terminating FSM transitions from `Idle` to `Active` immediately after receiving notification from the network that the terminal is ringing. The RFC 4235 SIP Dialog has a single state `Early` to include all messages prior to the call being answered by a 2xx message, such as 200 OK (Rosenberg, Schulzrinne and Mayh 2005). For the

purpose of Web abstraction, the progress of the network prior to presentation to the terminal is abstracted into the single `Initiated` state.

In the case of a particular network not providing an alerting notification, the call will transition from the `Initiated` state through the `Ringing` state to the `Active` state, by receiving an alerted notification and answered notification in succession. In the case of receiving a queued or redirected notification, the state will transition to `Queued`. For example the SIP message 180 Ringing is an optional message which may not be provided by the underlying network, however the 200 OK is mandatory.

- `Queued` - A state in which call progression is suspended or made inactive by the network whilst a connection is being established. For example when a call is queued due to the line being busy, or when a call is queued waiting for processing in a call centre application. When in the `Queued` state if all CONNECTIONS become `Active`, then the CALL transitions to the `Active` state. However if a `Queued` connection starts to ring, then the CALL transitions to the Ringing state.

  The JTAPI `Queued` state indicates that a CONNECTION is queued at the particular ADDRESS associated with the CONNECTION. This is due to some telephony platforms, such as call centres, permitting the "queueing" of incoming telephone calls to an address when the address is busy. In the Parlay MPCC when a call leg is queued, the event is reported to the application and call leg processing is suspended. The inclusion of the QUEUED state in the Extended Call Control call model allows the call processing to wait until the called party is available.

- `Ringing` - In this state the connection is waiting for confirmation from the participant. The device is assumed to be alerting the party that a connection request is incoming. If all ringing connections transition to `Active` then the call transitions to `Active`. A transition to the `Ringing` state occurs on receipt of an "alerting" message.

- `Error` - It is possible for all states to transition to the `Error` state except for `Inactive`. From an `Error` state the call can resume its `Active` state such as for example when another party is added to the call but fails due to some error.

4.3 Methodology to Determine ECC API Methods

The application programming interface (API) methods of the Extended Call Control Web service should be quickly understandable by developers of Web based call control applications. Identifying methods suitable for the Extended Call Control Web service requires a

methodology to ensure methods are applicable to the requirements of advanced call control. As such the methods presented by the ECC Web service are derived using two approaches:

- Identification and formalisation of user perceived call operations by considering a virtual private branch exchange scenario.
- Extension of existing call control Web services such as Parlay-X.

By considering complex APIs that provide call control such as base Parlay APIs, one can determine characteristic message sequences in any given application, such as Multi-Party Call Control; these sequences can be abstracted by simplified methods, including associated notifications, errors and failures. The methods can then be provided by an Extended Call Control Web service.

Parlay Conference Call Example

If one considers the sequences required to connect participants within a conference call using the Parlay Conference Call Control SCF API as shown in Figure 4.3, then the sequence of creating the call leg, requesting notifications regarding the leg, and routing the leg are repeated for each participant in the conference. This can be abstracted into a single initial message containing multiple participants. Parlay-X Third Party Call Control limits control to two parties (ETSI and Parlay 2007), and hence would not be able to control such a conference call. Thus, the makeCallSessionRequest method in the Parlay-X Third Party Call Control can be extended to multiple participants and abstract the initial creation of the call.

Virtual Switchboard Example

By extending the previous scenario an advanced call control scenario can be developed. For the purposes of this research the example of a virtual receptionist's switchboard is considered (Vannucci and Hanrahan 2007b). This Web based application would allow a company to operate in a distributed manner, with the receptionist using a rich Internet application to control the company "extensions", be they mobile or fixed numbers.

The application would have knowledge of all the extensions and be able to provide advanced call control for any of them as required. The user specific logic that would be associated

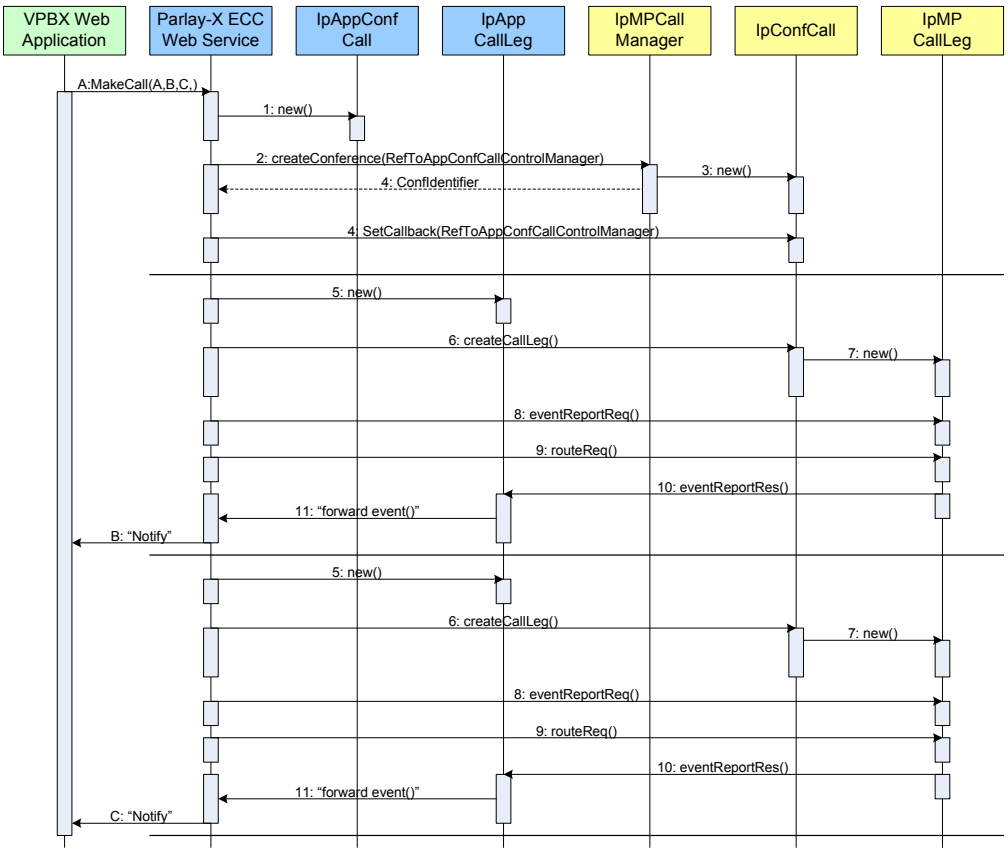


Figure 4.3: Creation of multiple participants for a conference call

with the application would reside within the Web application and as such would use the Extended Call Control Web service to provide call control functionality.

The Web application implementation is independent of the telecommunications operator, and as such could be be hosted on any Web connected device such as a mobile phone, thus allowing the device to provide direct application layer service signalling as proposed in Fricke and Hanrahan (2006), as illustrated in Figure 4.4 (Vannucci and Hanrahan 2007b).

The extensions that would be registered for monitoring and control by the Extended Call Control Web service would have to be separately identified by the telecommunications operator and customer. These extensions would be specifically marked as falling under third party call control, thus allowing the state of each extension to be provided to the third party. The company switchboard number would be marked as falling under the control of the service logic of the Web application. If no Web application is present to assume control of the call, then the number would have an alternative default processing as specified separately. The Web application is required to register with the ECC Web service, and indicate that it is ready to assume control of the processing of the call. The Web application

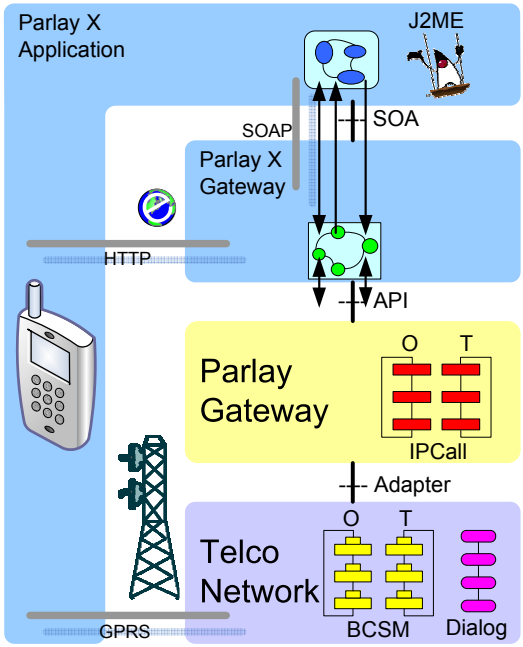


Figure 4.4: Parlay-X Extended Call Control Application (Adapted from (Vannucci and Hanrahan 2007b))

would then receive updates of the state of extensions falling within its control.

When a caller calls the company number, the processing of the call would be suspended, whilst the network reports the incoming call and is subsequently instructed to connect the call to the switchboard operator's number. The switchboard operator is able to perform standard and advanced private branch exchange functionality via the Web application, including:

- Transferring the caller to a salesman's mobile number, either conferring with the salesman or doing a blind transfer. This functionality requires: Call Leg Control and Call Creation.
- Creating a conference call between the caller, support and account salesman, and passing control of the conference to the salesman. This functionality requires Conference Call Creation and Control.
- Viewing which extension is currently involved in a call, and to whom. This functionality requires Call State.

In the following example, as shown in Figure 4.5 (Vannucci and Hanrahan 2007b), it is illustrated how the above concepts can be abstracted by an Extended Call Control Web

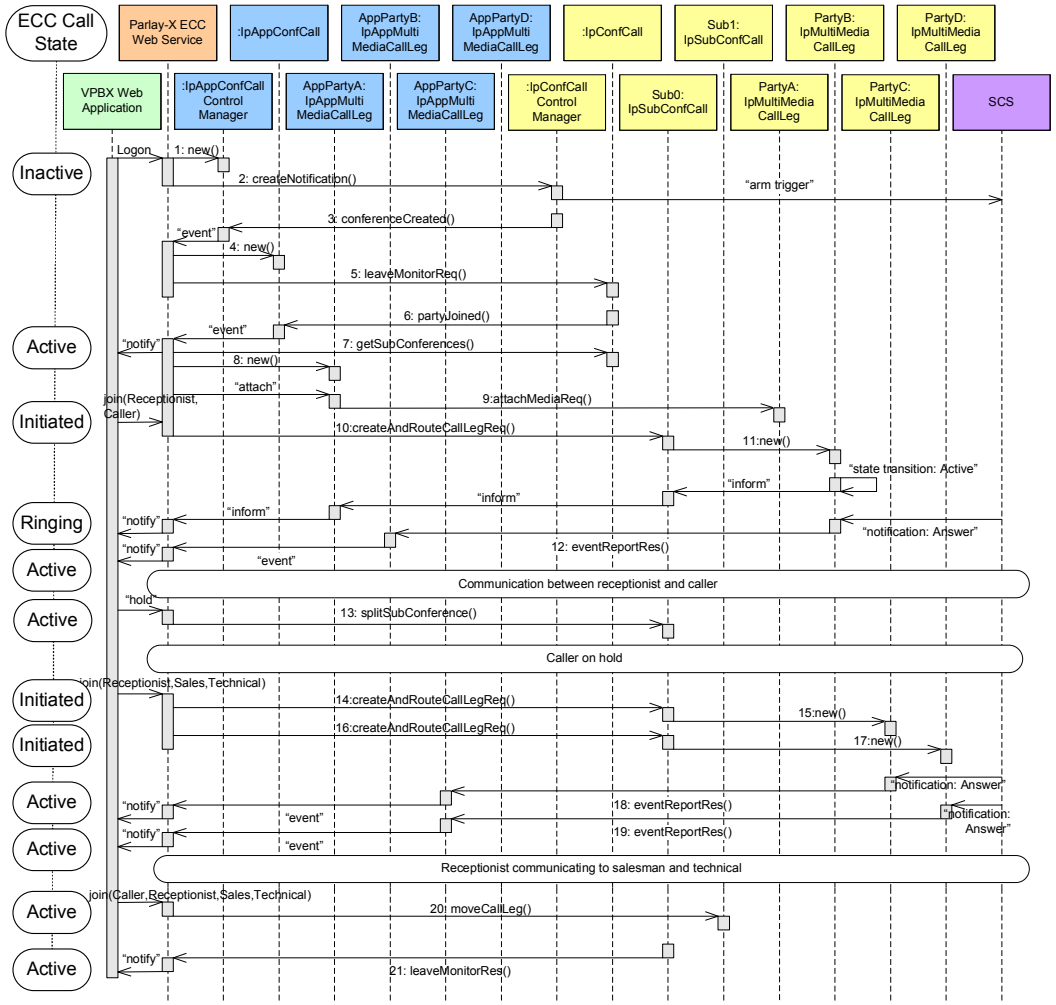


Figure 4.5: Virtual PBX Example (Adapted from (Vannucci and Hanrahan 2007b))

service and mapped to base Parlay APIs. The implementation of the ECC Web service logic to interwork with the network would be vendor specific, however in this example it is assumed that the Parlay Conference Call Control SCF API is used. In this example a receptionist fields various calls to the company number. The receptionist has a Web application to perform call logic by means of the Extended Call Control Web service.

1: The ECC Web service creates an object implementing the IpAppConfCallControlManager interface. The VPBX Web application is registered with the ECC Web service and associated stateful resource. At this point in time there are no active calls, and as such the VPBX Web application is waiting for instructions, from either the user or the ECC Web service associated stateful resource.

2: This message enables the ECC Web service to register with the network to receive notifications of new call events to the company number, by creating a trigger within the

network, and afterwards passing those on to the Web application.

3,4,5: The conference is created, waiting for an incoming connection and an associated object is created. The Web service requests to be notified of parties leaving the conference.

6: A caller calls the company switchboard number, which in turn triggers a notification that is handled by the service logic. The ECC Web service would instruct the network to handle the call leg in the appropriate manner as in 7,8,9, whilst notifying the Web application of the creation of a call within the network. This notification causes a transition within both the ECC Web service stateful resource and the Web application to the `Active` state.

7,8,9: The caller is automatically assigned to a sub-conference by the Parlay gateway. This is obtained by the Extended Call Control Web service, and a call leg object is created to represent the caller. The calling party is then joined to the conference by attaching the media. Note that when there is a change in media, the state model would reflect the change in the relevant connection.

10,11: The switchboard operator's mobile number is then added as a participant of the conference and a new IpMultiMediaCallLeg object is created. As the Web application informs the ECC Web service of the appropriate action to handle the further processing of the call, namely being to create and route the receptionist's number to the waiting caller, the call state transitions to `Initiated`.

The receptionist's number is registered with the ECC Web service, which has subscribed for notifications for each registered number. Thus, as the receptionist's extension enters the IN `T_ALERTING` detection point a notification is passed to the ECC Web service which subsequently notifies the Web application, and causes a transition to the `Ringing` state.

12: Once the receptionist answers the phone, a notification is issued by the network, and the Extended Call Control Web service notifies the Web application. The ECC Web service call state model transitions back to the `Active` state with both connections established, and communicating.

Once the caller is communicating with the receptionist, the receptionist has a number of available options such as transferring the caller, disconnecting the caller or starting a new conversation with other parties within the company.

At this stage there is communication between the caller and the receptionist. The caller requests to be put through to both sales and technical support. The receptionist then puts the caller on hold.

13: The Extended Call Control Web service splits the caller and receptionist so that there is no longer any communication between the two parties. There are now two calls with an active state, within the Web application as well as ECC Web service state manager, both calls have media that is not connected.

The caller is on hold, and the receptionist has to confer with sales and technical, so that the caller can be properly directed. In the following sequence the receptionist initiates a call with sales and technical, and later connects the caller to them.

Once the caller is on hold the receptionist calls the salesman and technical assistant so that they can be informed of the intention to transfer the caller to them.

14,15,16,17: Connections are created to both sales and technical. In this particular example the receptionist adds the sales and technical extensions to her call, this results in the receptionist's call transitioning to the `Initiated` state again, as new connections are being formed. As notifications return from within the network to indicate the progress of the call and that the sales and technical extensions are ringing, the call transitions to the `Ringing` state. As the call model is intended to abstract the individual extensions, the call state transitions to `Ringing` on the first notification of an extension ringing.

18,19: As the parties answer the call, the Web application is notified, and on the last ringing party answering the call state transitions to `Active`.

The receptionist then transfers the caller from his call to the call with sales and technical. The call, in which the caller was previously, no longer has any extensions, and transitions to the `Inactive` state. The receptionist's call remains in the `Active` state, receiving an update regarding the additional connection.

20: The caller is moved by the ECC State Manager to the same conference as the sales and technical and the receptionist is free to disconnect.

21: The receptionist disconnects and the VPBX application is notified, the call remains in the `Active` state, however the connection information is altered.

From this scenario the requirement to be able to add and remove participants from calls in progress is identified, and subsequently to receive notifications regarding those extensions regardless of the call to which the participant is connected. In order to do this again, a resource Identifier is required for each message.

Table 4.1: registerRequest

Part Name	Part Type	Optional	Description
username	xsd:string	No	Contains the identity of the party with whom an operator agreement already exists
password	xsd:string	No	Authenticates the identity of the party with whom an operator agreement already exists

Table 4.2: registerResponse

Part Name	Part Type	Optional	Description
resourceIdentifier	xsd:string	No	Identifies the endpoint Reference of the stateful resource. Uses endpoint Reference identifier if existing.

4.4 ECC Methods

From the previous section 4.3 in which basic Extended Call Control methods were determined, the methods proposed for the Extended Call Control Web service are now described. The requirements determined in Chapter 2 for a resource identifier as well as Chapter 3 for a call identifier are incorporated into the Extended Call Control methods.

4.4.1 Register

The invocation of REGISTER enables the Web service to inform a stateful resource of a controlling Web application's presence. The presence of a Web application causes the stateful resource to monitor and transfer control of extensions to the Web application. Network notifications are published by the stateful resource. The state of all controlled extensions by the application are observed as well as associated connections. By means of the `username` and `password` the Web application is authenticated and the pre-registered Extended Call Control extensions are determined. A `resourceIdentifier` is returned on successful invocation of the REGISTER operation. If a subsequent registration is received for the same user name and password the current resource identity is returned.

Table 4.3: createCall

Part Name	Part Type	Optional	Description
resourceIdentifier	xsd:string	No	Identifies the endpoint Reference of the stateful resource
callParticipants	xsd:anyURI [1..unbounded]	No	Contains the addresses of the participants to add to the call.
mediaInfo	common: MediaInfo [0..unbounded]	Yes	Identifies the media direction for each participant to be added to the call session for each media type. If no media direction - in,out, or bidirectional - is chosen then a default of bidirectional shall be applied. If this part is omitted, the media type(s) shall be negotiated by the underlying network.

Table 4.4: createCallResponse

Part Name	Part Type	Optional	Description
callIdentifier	xsd:string	No	Identifies the call session created.

4.4.2 createCall

The invocation of CREATECALL requests to set up a call between multiple addresses. Optionally the media for the connections in the call can be specified for each participant, if no media information is specified then the connection is left to the default of the underlying network. The `resourceIdentifier` ensures that the Web service passes the request to the correct stateful resource. A `callIdentifier` is returned immediately on invocation of the CREATECALL operation, and the call session is established asynchronously. This `callIdentifier` is used to identify the call in subsequent operations.

4.4.3 endCall

The invocation of ENDCALL terminates the call identified by the `callIdentifier`. All connections to the call are removed, and the call identifier is no longer valid after a set time, at which point the stateful resource no longer maintains the state of the call and associated connections. The provided `callIdentifier` is returned immediately on invocation of

Table 4.5: endCallRequest

Part Name	Part Type	Optional	Description
resourceIdentifier	xsd:string	No	Identifies the endpoint Reference of the stateful resource
callIdentifier	xsd:string	No	Provides the call identifier that should be used to identify the call

Table 4.6: endCallResponse

Part Name	Part Type	Optional	Description
callIdentifier	xsd:string	No	Identifies the call terminated. Uses given call identifier as confirmation of receipt of message.

the ENDCALL operation, if an invalid `callIdentifer` is provided then the response will be null.

4.4.4 addCallParticipants

The invocation of ADDCALLPARTICIPANTS adds additional participants to an existing call, as specified by the `callIdentifier`. The participants to the call are added asynchronously by the underlying network, and the `callIdentifier` is returned immediately on invocation of the ADDCALLPARTICIPANTS operation.

4.4.5 moveCallParticipants

The invocation of the MOVECALLPARTICIPANTS operation enables participants identified by `callParticipants` to be moved to the call identified by `callIdentifier`. The participants can be connected to different calls and thus only the destination call is identified. The stateful resource as specified by the `resourceIdentifier` is responsible for maintaining information regarding the participants and the calls from which they are originating.

Table 4.7: addCallParticipantsRequest

Part Name	Part Type	Optional	Description
resourceIdentifier	xsd:string	No	Identifies the endpoint Reference of the stateful resource
callIdentifier	xsd:string	No	Provides the call identifier that should be used to identify the call
callParticipants	xsd:anyURI [1..unbounded]	No	Contains the addresses of the participants to add to the call
mediaInfo	common: MediaInfo [0..unbounded]	Yes	Identifies the media for each participant added to the call. If this part is omitted, the media type(s) shall be negotiated by the underlying network.

Table 4.8: addCallParticipantsResponse

Part Name	Part Type	Optional	Description
callIdentifier	xsd:string	No	Identifies the call used. Uses given call identifier as confirmation of receipt of message.

Table 4.9: moveCallParticipantsRequest

Part Name	Part Type	Optional	Description
resourceIdentifier	xsd:string	No	Identifies the endpoint Reference of the stateful resource
callIdentifier	xsd:string	No	Provides the call identifier that should be used to identify the call to which the Participants are being moved.
callParticipants	xsd:anyURI [1..unbounded]	No	Contains the addresses of the participants to move to the call

Table 4.10: moveCallParticipantsResponse

Part Name	Part Type	Optional	Description
result	xsd:string	No	Identifies the call session used. Uses given call session identifier as confirmation of receipt of message.

Table 4.11: removeCallParticipants

Part Name	Part Type	Optional	Description
resourceIdentifier	xsd:string	No	Identifies the endpoint Reference of the stateful resource.
callIdentifier	xsd:string	No	Provides the call identifier that should be used to identify the call.
callParticipants	xsd:anyURI [1..unbounded]	No	Contains the addresses of the participants to remove from the call.

Table 4.12: removeCallParticipantsResponse

Part Name	Part Type	Optional	Description
callIdentifier	xsd:string	No	Identifies the call used. Uses given call identifier as confirmation of receipt of message.

4.4.6 removeCallParticipants

The invocation of REMOVECALLPARTICIPANTS removes the participants identified by `callParticipants` from the call identified by `callIdentifier`. The participants connections are terminated within the underlying network. The provided `callIdentifier` is returned immediately on invocation of the REMOVECALLPARTICIPANTS operation.

4.4.7 modifyMedia

The invocation of MODIFYMEDIA modifies the media of the participants identified by `callParticipants`. The participants media are altered within the underlying network. The provided `callParticipants` is returned immediately on invocation of the MODIFYMEDIA operation. As the participants might not be in the same call, the `callIdentifier` is not provided.

4.5 Transitions

Each Extended Call Control Web service method would result in a change of the network and subsequently a change of the state model. The change of state would be reported to

Table 4.13: removeCallParticipants

Part Name	Part Type	Optional	Description
resourceIdentifier	xsd:string	No	Identifies the endpoint Reference of the stateful resource.
callParticipants	xsd:anyURI [1..unbounded]	No	Contains the addresses of the participants to remove from the call.
mediaInfo	common: MediaInfo [0..unbounded]	No	Identifies the media for each participant

Table 4.14: removeCallParticipantsResponse

Part Name	Part Type	Optional	Description
callParticipants	xsd:anyURI [1..unbounded]	No	Identifies the participants. Uses given participants as confirmation of receipt of message.

the Web application so that futher application logic can take place. In Table 4.15 the effect of ECC methods from section 4.4 on the state of the network are shown, together with the outcome of such an action in terms of the change of the call state model as shown in Figure 4.2 and subsequent notification generated.

4.6 Notification Mapping

Mapping the notifications from the underlying network to those of the ECC Call State model ensures that adequate information is preserved for the Web application to perform advanced call control as required in Chapter 1.1. Chapter 5.5 showed the mapping of the call state models between IN CS1 and OSA/Parlay, as well as OSA/Parlay and SIP, and the associated notifications were mapped to each other. Table 4.16 presents a mapping between the IN CS1 originating notifications, OSA/Parlay Multi-Party Call Control originating call leg notifications and the Extended Call Control call model. Table 4.17 presents a mapping between the IN CS1 terminating notifications, OSA/Parlay Multi-Party Call Control terminating call leg notifications and the Extended Call Control call model.

Table 4.15: Transitions

Method Name	State Before	State After	Notification Generated
register	null	update of state	none
createCall	null	initiated	in progress
endCall	any	inactive	disconnect
addCallParticipants	active	initiated	in progress
moveCallParticipants	active	active	disconnect,answered
removeCallParticipants	active	active	disconnect
modifyMedia	active	active	media

4.7 Conclusion

Requirements for Extended Call Control were presented in Chapter 1.1. By analysing existing call models as presented in Chapter 3, and the requirements for Web services in Chapter 2, an Extended Call Control call model and Web service was synthesised to meet those requirements. Abstraction and identification of suitable states for a Web based call model were determined and by using method discovery as shown in section 4.3 the methods for the Extended Call Control call model Web service were created. A mapping between the underlying telecommunications notifications and the Extended Call Control call model was performed to ensure the call model can incorporate network state and abstract it suitably. The Extended Call control call model abstracts the complexity of the state of individual terminal connections, and has the ability to provide such information if required by the Application. The state of the terminal in terms of the media is abstracted, as well as the relationship between the terminal and network. Common identifiers allow unique identification of the call as well as connection by the Web application logic. The Extended Call Control call model state abstracts underlying call state in a manner that is suitable for Web services aiming to provide advanced call control. The use of the Extended Call Control call model and Extended Call Control Web service are demonstrated in the following chapter.

Table 4.16: Originating Notification Mapping of IN Q.1214 and Parlay Multi-Party Call Control notifications to Extended Call Control Call Model

Q.1214 Originating Notifications	ECC Notifications	OSA/Parlay MPCC Originating Notifications
		P_Call_Event_Originating_Call_Attempt
1. Origination_Attempt_Authorized	incoming	P_Call_Event_Originating_Call_Attempt_Authorised
	redirected	P_Call_Event_Redirected
	queued	P_Call_Event_Queued
2. Collected_Info	in progress	P_Call_Event_Address_Collected
3. Analysed_Info	in progress	P_Call_Event_Address_Analysed
4. Route_Select_Failure	error	P_Call_Event_Originating_Release (P_Routing_Failure)
5. O_Called_Party_Busy	called party busy	P_Call_Event_Originating_Release (P_Busy)
6. O_No_Answer	no answer	P_Call_Event_Originating_Release (P_No_Answer)
	alerting	P_Call_Event_Alerting
7. O_Answer	answered	P_Call_Event_Answer
8. O_Mid_Call		P_Call_Event_Originating_Service_Code ()
9. O_Disconnect	disconnect	P_Call_Event_Originating_Release (P_Disconnected)
	disconnect	P_Call_Event_Terminating_Release (P_Disconnected)
10. O_Abandon	disconnect	P_Call_Event_Originating_Release (P_Premature_Disconnect)
	disconnect	P_Call_Event_Originating_Release (P_Disconnected)

Table 4.17: Terminating Notification Mapping of IN Q.1214 and Parlay Multi-Party Call Control notifications to Extended Call Control Call Model

Q.1214 Terminating Notifications	ECC Notifications	OSA/Parlay MPCC Terminating Notifications
		P_Call_Event_Terminating_Call_Attempt
12. Term_Attempt_Authorized	incoming	P_Call_Event_Terminating_Call_Attempt_Authorised
13. T_Called_Party_Busy	called party busy	P_Call_Event_Terminating_Release (P_Busy)
14. T_No_Answer	no answer	P_Call_Event_Terminating_Release (P_No_Answer)
	alerting	P_Call_Event_Alerting
	redirected	P_Call_Event_Redirected
	queued	P_Call_Event_Queued
15. T_Answer	answered	P_Call_Event_Answer
16. T_Mid_Call		P_Call_Event_Terminating_Service_Code ()
17. T_Disconnect	disconnect	P_Call_Event_Terminating_Release (P_Disconnected)
	disconnect	P_Call_Event_Originating_Release (P_Disconnected)
18. T_Abandon	disconnect	P_Call_Event_Terminating_Release (P_Premature_Disconnect)
	disconnect	P_Call_Event_Terminating_Release (P_Disconnected)

Chapter 5

Demonstration of Extended Call Control Call Model

To demonstrate the use of the Extended Call Control call model and Web service, a rich Internet application is defined that consumes the ECC Web service to provide advanced call control functionality to an end user. The rich Internet application uses the ECC Web service together with received state notifications to monitor and control the telecommunication network functioning from outside of the telecommunications operators' domain. Such an application would be hosted and created by a third party provider who is within the IT domain, and would make use of the ECC Web service which is within the domain of the telecommunications operator. The Web service uses the infrastructure of the telecommunications operator to perform necessary underlying call control and bearer connectivity.

The application chosen to demonstrate the concepts of the Extended Call Control Web service is a virtual private branch exchange application. The virtual private branch exchange application can co-ordinate the functionality offered by the underlying network resources, namely voice, video, and data, and provide additional added value from the IT domain. This service is named *Virtual Private Branch Exchange* (VPBX). The service demonstrates real time call and extension control functionality, incorporating asynchronous behaviour for real time notification that is required for rich Internet application logic.

In section 5.1 the VPBX application requirements are described, considering fundamental functionality required to demonstrate the use of the Extended Call Control call model. Section 5.2 follows on to describe the use cases for the application. The graphical user interface for the application is shown in section 5.3. The architecture of the application is described in section 5.4 as well as the proof of concept interface. The mapping of the call model is discussed in section 5.5 as well as message sequence charts detailing the messages exchanged to update the application as state changes.

5.1 Virtual Private Branch Application

The VPBX implementation is intended to demonstrate the value of knowledge of state when providing Web applications with advanced call control functionality. Namely the benefit of a call state model when performing advanced call control functionality such as altering the state of extensions whilst those extensions are participating in a call. The functionality of the VPBX application is limited to displaying the state received by the network and performing call control operations. Additional functionality, such as integration with further Web services, is beyond the scope of demonstrating the value of state information, and not explored in this implementation. The functionality of the VPBX application from the user's point of view is defined in use cases to illustrate fundamental call control behaviour (Kruchten 2003). The application incorporates the basic call control functionality such as starting and stopping a call as well as more advanced functionality such as controlling the call whilst it is in progress. Figure 5.1 shows the use cases determining basic application functionality (Bittner 2000).

The VPBX Application graphical user interface is shown with relation to fulfilling the previous use cases as described in Chapter 4.

5.2 Use Cases

The VPBX application is firstly described from the perspective of the end user, hence the use cases describing interactions between the end user and the VPBX application are defined. The VPBX use cases are shown in Figure 5.1, however the shaded use cases are not explored further, such as login which requires authentication, and configuration of the VPBX application and associated extensions. Each use case is described below:

Login: The end-users authenticate themselves with the VPBX application by providing user names and passwords. Once authenticated they are allowed to access the ECC Web service. Note that the number of users can vary, and identification of the extensions to control and the notifications to receive are determined by the application. Web-based authentication, authorisation and accounting practices can be employed to fulfil this use case.

Logout: The end-users close the VPBX application, resulting in the VPBX application no longer receiving state updates, and no longer providing call control using the ECC Web service. In the case of there no longer being any registered application in control of the VPBX calls, the network operator would default to a standard pre-set behaviour, such as

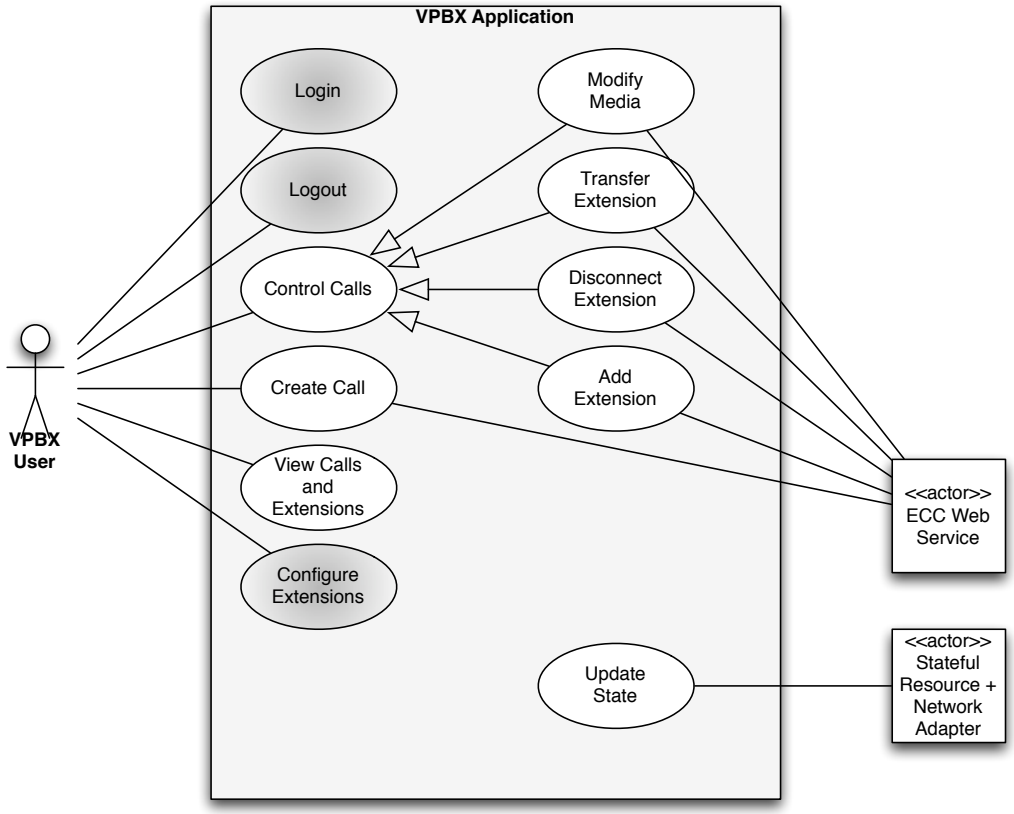


Figure 5.1: Virtual Private Branch Exchange Use Cases

forwarding the call to a particular extension.

Configure Extensions: A user is able to alter the extensions falling within the user's domain, such as adding and removing extensions to the VPBX. Such behaviour would require the interaction of the telecommunications operator, as there could be service level agreements and different pricing structures for extensions falling within the virtual private branch exchange of the user. As this is outside of the scope of call control, it is left for future consideration.

Control Call: A user is able to perform advanced call control operations with the external and registered extensions that are participating in a call. The control of a call can be decomposed into further sub use cases, as shown in Figure 5.1.

Create Call: A user is able to create a call between multiple ECC registered extensions as well as external numbers. Each participant within the call will fall under control of the user.

View Calls and Extensions: The user views the state of calls within the VPBX as well as registered extensions and other external participants that are involved in a call. New calls created by participants calling the VPBX number can be viewed and subsequently controlled.

Update State: The Stateful Resource updates the state of the calls and information on extensions of interest.

The Control Calls use case can be decomposed into the following sub-use cases:

Transfer Extension: A user is able to transfer a participant from an existing call to an alternate call.

Disconnect Extension: A user is able to remove a participant from a call currently in progress. If there are no longer any participants in the call, this has the same effect as ending the call.

Add Extension: Adds a participant not already in a call to a call currently in progress. If the participant added is not registered with the VPBX application then the contact details of the participant are provided by the user.

Modify Media: The user is able to alter the media of a participant, selecting whether the media flow is bidirectional or unidirectional should the network support such functionality.

5.3 Graphical User Interface

The user can view all connections and calls by means of the Web browser, the user is able to select calls that are in progress as well as extensions that are registered on the system, as shown in Figure 5.2. Calls are represented by their call identifier, which is implemented as a AsteriskNow meet me conference address. The extensions are represented by their AsteriskNow extension identifier, such as SIP/6000. The option to create a call requires a particular extension to be first selected, and then the Create button selected, resulting in a message to the user of the resulting action, as shown in Figures 5.3. To add an extension to an already existing call the extension and the call are selected and the Add To button selected, resulting in a message as shown in Figure 5.4. To transfer an extension to an already existing call the extension and the call are selected and the Transfer To button selected, resulting in a message as shown in Figure 5.5. Selecting an extension and clicking the Disconnect Extension will result in a message as shown in Figure 5.6. The message

in Figure 5.7 is shown when ending a call by selecting it and clicking the End Call button.

The Web service is implemented using Java Glassfish, and manual invocation of the Web service is possible using the Glassfish application server, as shown in Figure 5.8.

5.4 Implementation Architecture

The implementation of the Extended Call Control Web service as well as asynchronous notification would vary between service providers; however, certain concepts are required to be present in each case. Firstly the need for the rich Internet application to receive real time notifications of alterations in state of the telecommunications network resources, so that the application logic can provide instructions to the network; and secondly the mapping of state between the underlying network and the Extended Call Control call model.

In this implementation the architecture shown in Figure 5.9 was used to demonstrate the principles of Extended Call Control. The Web Requestor is the end user that controls the VPBX application, for the purpose of this proof of concept the application is JavaScript that is downloaded from the Third Party Service Provider. The application could be implemented in a number of ways such as a precompiled desktop application or as a downloadable widget. A rich Internet application has the advantage of the third party service provider being able to upgrade the application and subscribers would immediately be able to benefit from such an upgrade. The VPBX Application Web Server contains the VPBX application JavaScript as well as the JavaScript to interwork with the Streaming HTTP Server. The Streaming HTTP Server or comet server is responsible for pushing the asynchronous notifications to the VPBX application on the end user's browser. Both the VPBX Application Web Server as well as the Streaming HTTP Server are within the third party service provider domain. For this implementation, authentication as well security concerns with HTTP requests to multiple domains was not considered. Web browser security typically does not permit multiple domain requests, without signed and trusted JavaScript. For this implementation authentication is assumed to be handled separately using standard Web Authentication Authorisation and Accounting (AAA) methods. The Extended Call Control Web service is contained within the telecommunications operator domain and provides the required functionality to control the abstracted underlying network. The Web server hosting the ECC Web service could be implemented by a telecommunications operator as a Parlay-X Gateway with other Parlay-X Web services, or a standalone Web server. To implement the Extended Call Control Web service the standardised Parlay-X set of technologies were extended. The Parlay-

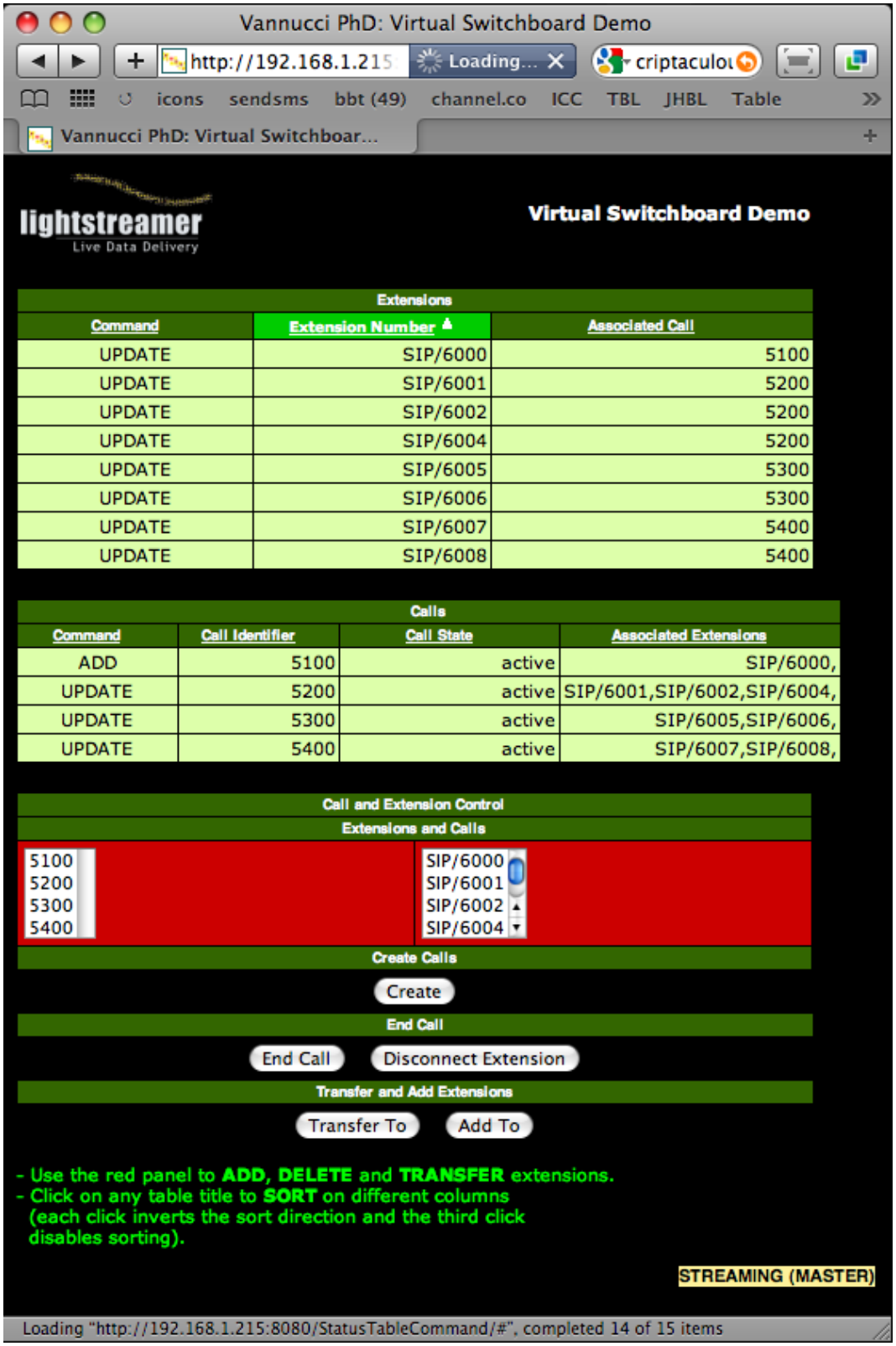


Figure 5.2: Virtual PBX Graphical User Interface

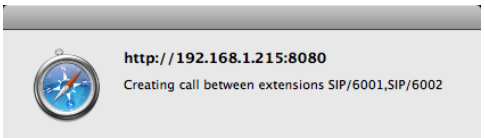


Figure 5.3: Call Created



Figure 5.4: Add Extension



Figure 5.5: Transfer an Extension



Figure 5.6: Remove an Extension

X Third Party Call Control forms a basis for the Extended Call control Web service. The Stateful Resource + Network Adapter is responsible for the maintenance of the Extended Call Control state models, as well as the state models of the underlying network, and the mapping between the models. In addition the Extended Call Control Web service methods have to be adapted to network dependent instructions. The stateful resource and network adapter can be likened to the functionality provided by an application server within the telecommunications network, with the application server providing the application logic to provide mapping and translation of Web requests, and the underlying application server network adapter (such as a Parlay SCS or CAMEL IM-SSF) communicating with the network equipment. The telecommunications network is implemented by the Network Emulator which demonstrates the functionality of the network equipment. The Network Emulator is implemented as an IP based soft switch. The soft switch provides an emulation of the underlying telecommunications equipment such as an IMS Serving Call Session Control Function (S-CSCF).



Figure 5.7: End a call

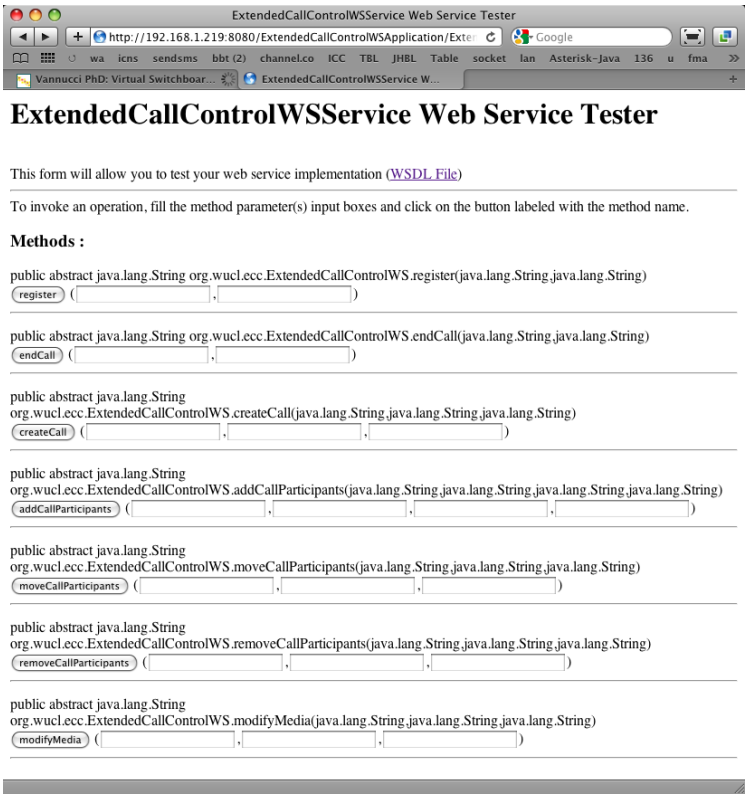


Figure 5.8: Manual Web Service Tester

The technologies used for each entity are as follows:
Web Requestor – Firefox browser
Extended Call Control Web service – Glassfish Java Application Server
VPBX Application Web Server – Lightstreamer Web Server
Streaming HTTP Server – Lightstreamer Server
Stateful Resource + Network Adapter – Java application
Network Emulator – AsteriskNOW

5.4.1 Communication

Communication between the VPBX Application Web Server and the browser is standard HTTP, with the browser downloading the necessary JavaScript from the server to load the application in the browser, and communicate with the Streaming HTTP Server as well as the ECC Web service. The Streaming HTTP Server and VPBX application intercommunicate using AJAX, using proprietary Lightstreamer technology. Invocation of Extended Call Control methods is performed using SOAP. Communication between the ECC Web server and the stateful resource is by means of sockets. The use of sockets is to

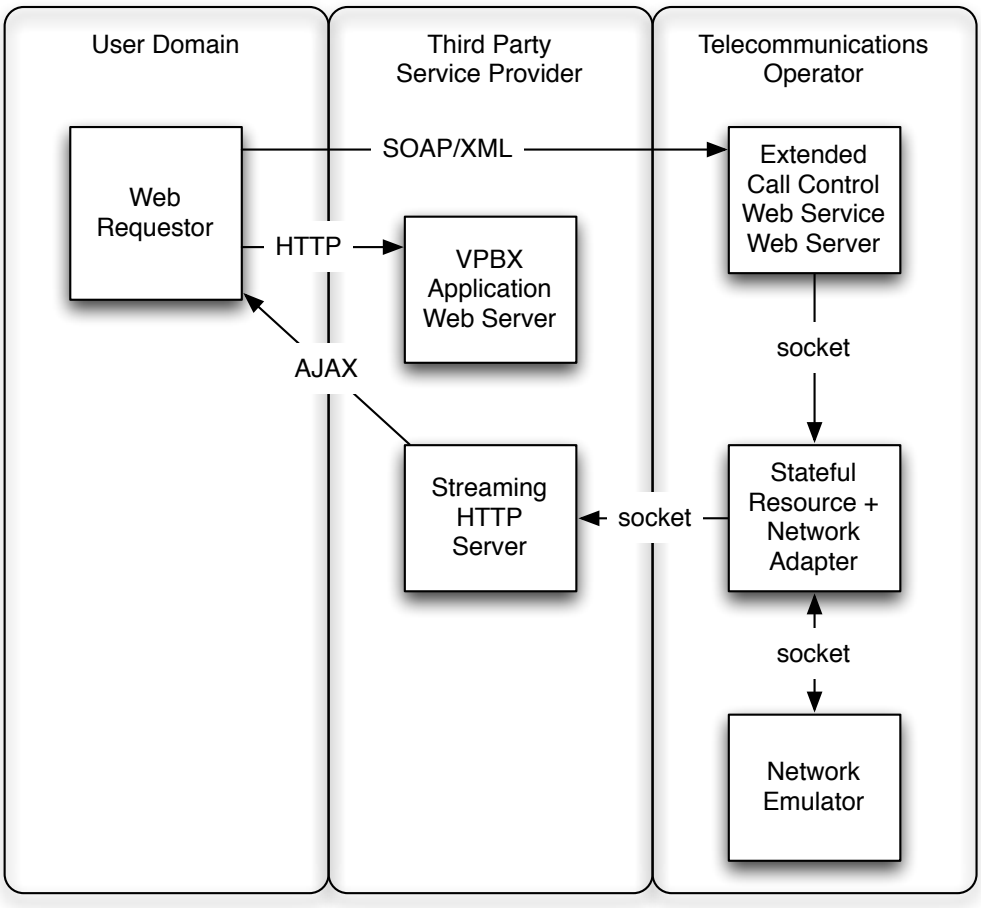


Figure 5.9: Virtual Private Branch Exchange Implementation

illustrate fundamental connectivity and more advanced inter-process communication such as CORBA is not implemented in this proof of concept.

In this implementation the VPBX application interacts directly with the Extended Call Control Web service, however there are a number of alternative methods of consuming such a service. A preferred method would be to use the VPBX Application Web server as a proxy for the Extended Call Control Web service, this would have the added advantage of there no longer being security concerns due to multiple domains, and the third party service provider could further abstract the Extended Call Control Web service by inserting the session specific information on behalf of the user, as illustrated in Figure 5.10.

Communication between the Stateful resource + Network Adapter and the Streaming HTTP Server is performed using sockets, using the *Lightstreamer ARI* protocol. Notifications of changes in state are passed from the Stateful resource to the Streaming HTTP

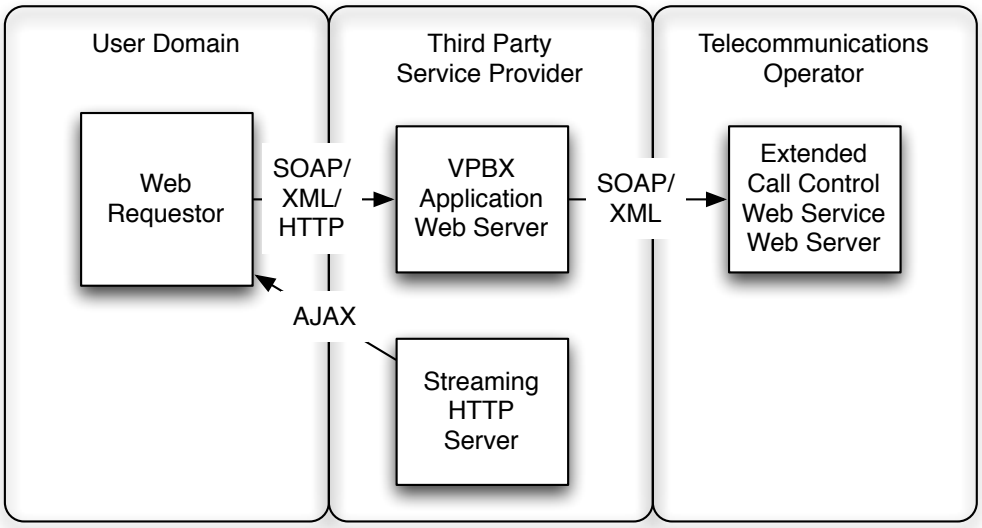


Figure 5.10: Third Party Proxy Web Service

Server, which in turn processes and formats the notification for streaming to the Web Requestor. Note that this separation of state notification and ECC Web service invocation would allow the ECC Web service to support synchronous behaviour if so required, as per the WS-Resource Framework stateful resource pattern (Foster et al. 2004).

The Stateful Resource + Network Adapter use the Asterisk-Java package (Reuter 2010) to interwork with the AsteriskNOW server by means of the *Manager API*. This allows the state of the management and control of extensions within the AsteriskNOW server to be monitored, without interrupting the Asterisk call processing (Reuter 2010).

5.5 Call Model Mapping

Call model mapping has to be performed to interwork the state of extensions within the AsteriskNOW server and the ECC State manager. Through the *Asterisk-Java API* the AsteriskNow server provides a high level Asterisk state view of extensions based on underlying SIP message sequences. The AsteriskNow server does not represent calls as separate objects but rather provides the concept of extensions that are connected to a channel which are linked to other channels. Thus the state of the Network Emulator can be described as channels and associated extensions. Asterisk channels have the following possible states (Reuter 2010):

`BUSY`: Line is busy,
`Dialing`: Digits (or equivalent) have been dialled,
`Dialing_Offhook`: Digits (or equivalent) have been dialled while offhook,
`Down`: Channel is down and available,
`Hungup`: The channel has been hung up and is not longer available on the Asterisk server,
`Offhook`: Channel is off hook,
`Prering`: Channel has detected an incoming call and is waiting for ring,
`Ring`: Line is ringing,
`Ringing`: Remote end is ringing,
`Rsrvd`: Channel is down, but reserved,
`Up`: Line is up.

State transitions within Asterisk are not clearly defined, and a number of Asterisk states do not map to the Extended Call Control call model. From the implementation it was found that the most commonly observed states were `Down`, `Ring`, `Ringing`, and `Up`. Due to the difference in the Extended Call Control call model being a representation of all connections and the AsteriskNOW state being a representation of multiple channels, the state reported has to be contextualised to best represent the overall call behaviour. The mapping used between the two state models is as shown in Figure 5.11.

The AsteriskNOW management API provides event notifications for any event that occurs within the softswitch. These events are processed and reported by the Asterisk-Java API. The Asterisk-Java events are monitored by the Network Adapter for state change notifications that would alter the call state. Each of the ECC methods are mapped to AsteriskNOW. The following use cases and message sequence charts show how the call state models are mapped.

5.5.1 Registration

The registration process, register, shown in Figure 5.12, allows the Web application to create the asynchronous link between the two systems. An identifier is passed back to the application allowing the stateful resource to be identified in future transactions. This allows the Web service to make use of more than one stateful resource. In addition registration causes the state manager to set the required listeners with respect to the extensions registered for the Extended Call Control Web service, as well as needed network triggers. Registration is required to inform the network that there is an application which wishes to perform call control. In the case of there being no controlling application the network has to have a preset manner to handle an incoming call to the VPBX number, such as forwarding

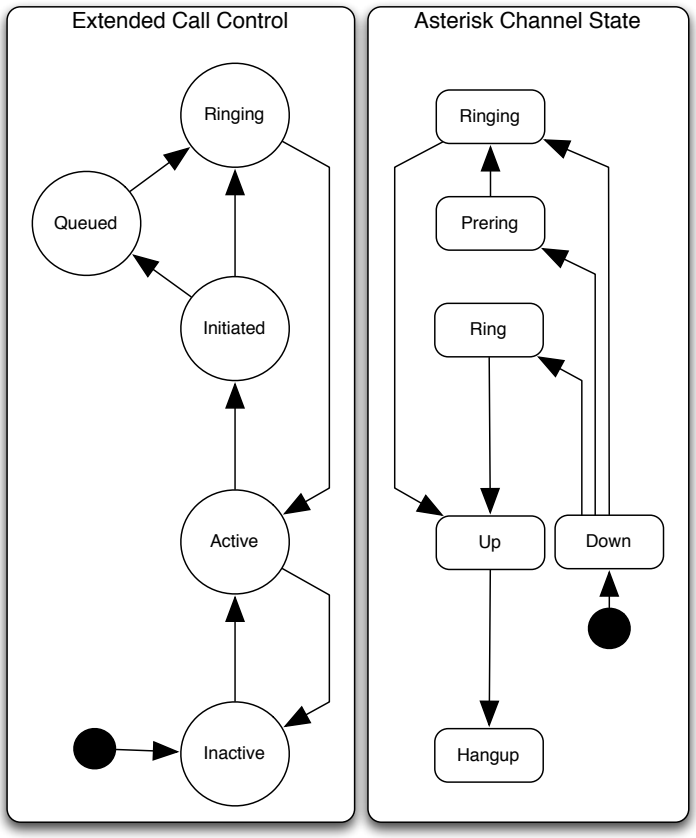


Figure 5.11: Mapping Extended Call Control Call Model to Asterisk

to a set phone number. This functionality is outside of the scope of this research and can be fulfilled by a Web service such as the Parlay-X Call Handling Web service.

5.5.2 Creation of a Call

The creation of a call is shown in Figure 5.13. The implementation of the call creation can differ from system to system, as some vendor equipment supports asynchronous call leg routing, whilst others do not. Figure 5.13, messages 1–25, shows the creation of a two-party call with synchronous leg creation; and messages 26–46 shows the creation of a two-party call with asynchronous leg creation.

In the first sequence, messages 1–25, the user selects the participants to connect, the createCall method is invoked on the ECC Web service and a call identifier, message 5, `callId`, is returned, to the Web application. The creation of the call within the state manager causes the call to transition to the `Inactive` state, as no CONNECTIONS presently exist. This is passed back asynchronously to the VPBX application, as shown in

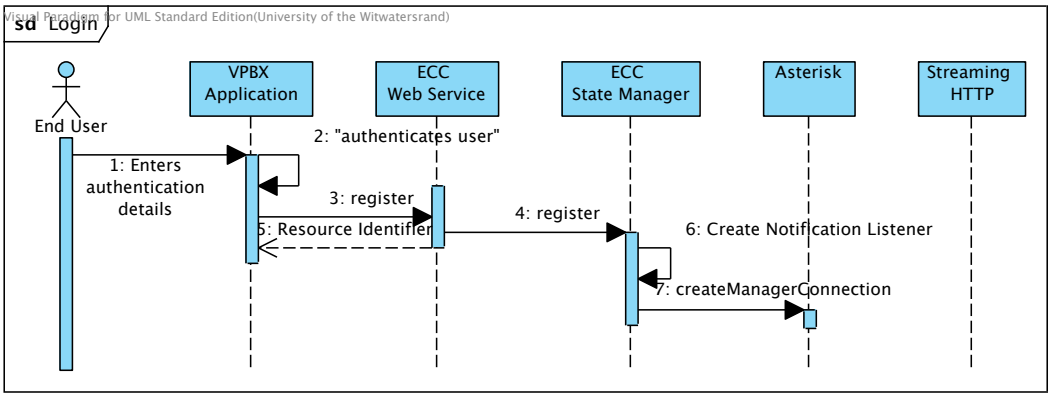


Figure 5.12: Extended Call Control Registration

message 6.

The network adapter creates an OriginateAction, message 7, to connect the first participant. The network adapter makes use of Asterisk meet me applications to interconnect participants. As the channel is created within Asterisk, a `NewChannelEvent` is created by the Asterisk network emulator reporting the channel state as `Down` as shown in message 8. This is mapped by the State Manager to the call state of `Active` and then `Initiated`, and reported to the Web application, message 9. As the extension within the network reports that it is ringing, Asterisk generates a `NewStateEvent` notification, message 10, with the state `Ringing`. The Asterisk state `Ringing` is mapped to the ECC state `Ringing`, and passed to the Streaming HTTP Server, message 11, for asynchronous delivery to the Web application.

When the extension goes offhook, a second `NewStateEvent` is reported with the connection state of `Up`, message 13, this subsequently is mapped to the ECC `Active` state, and the Web application is notified, message 15. Once the first call leg is active the second call leg becomes initiated this causes the ECC call state to transition from `Active` to `Initiated` as a new extension is added (message 19). The same process as before is carried out as `Ringing` is reported, and finally the Web application transitions to `Active`, message 25, as the extension goes off hook and all participants are communicating.

In the second sequence, messages 26–46, the selection of participants is the same as in the first sequence, however the asynchronous leg creation means that the network adapter can create multiple OriginateAction requests in series, as shown in messages 32–33, for each participant. As each OriginateAction is processed and a channel is created within Asterisk, a `NewChannelEvent` is reported. Each notification has an attached identifier which is the

same as provided when invoking the OriginateAction, allowing responses to be correlated to the correct channel. The reported Asterisk channel state of `Down` is mapped by the State Manager to the ECC call state `Active` and the application is notified by the Streaming HTTP Server, message 36.

The receipt of the first Asterisk channel `Up` notification, message 43, does not cause a transition in the State Manager as there are other extensions still ringing. Once all extensions are connected, message 44, the ECC state becomes `Active` and is reported asynchronously to the Web Requestor VPBX application, message 46.

5.5.3 Transferring a Participant

In order for a participant to be transferred the participant has to be active in an existing call, this is determined by the application as a result of knowing the call state, as shown in Figure 5.14. The participants to be transferred are selected by the user, and the Extended Call Control Web service moveCallParticipants method is invoked. This results in the ECC State Manager + Network Adapter issuing a RedirectAction to the Network Emulator. Asterisk meetme `MeetmeLeave` events are triggered when the participant leaves the previous call, message 5, which is passed to the STREAMING HTTP SERVER which notifies the WEB REQUESTOR asynchronously, message 7. Once the participant has sucessfuly joined the selected call. Asterisk creates a `MeetmeJoin` event, as shown in message 8, once both participants are sucessfully connected, resulting in the State Manager causing the state of the selected call to transition through `Initiated` to `Active`.

5.5.4 Adding a Participant

Adding a participant to a call is shown in Figure 5.15. the call is already in the `Active` state, and the participants being added create a sequence similar to a new call, in that the addParticipants method causes the Network Emulator to issue an OriginateAction, as shown in message 4. this would cause the Asterisk Network Emulator to create a `NewChannelEvent` with the state `Down` which would be mapped to the ECC call model state `Initiated`. The Streaming HTTP Server would notify the Web Requestor VPBX application of the change in state, message 10. Once the extension is answered the State Manager + Network Emulator would change the call state to `Active`. The transition for multiple participants is the same as those of when multiple call legs are created simultaneously.

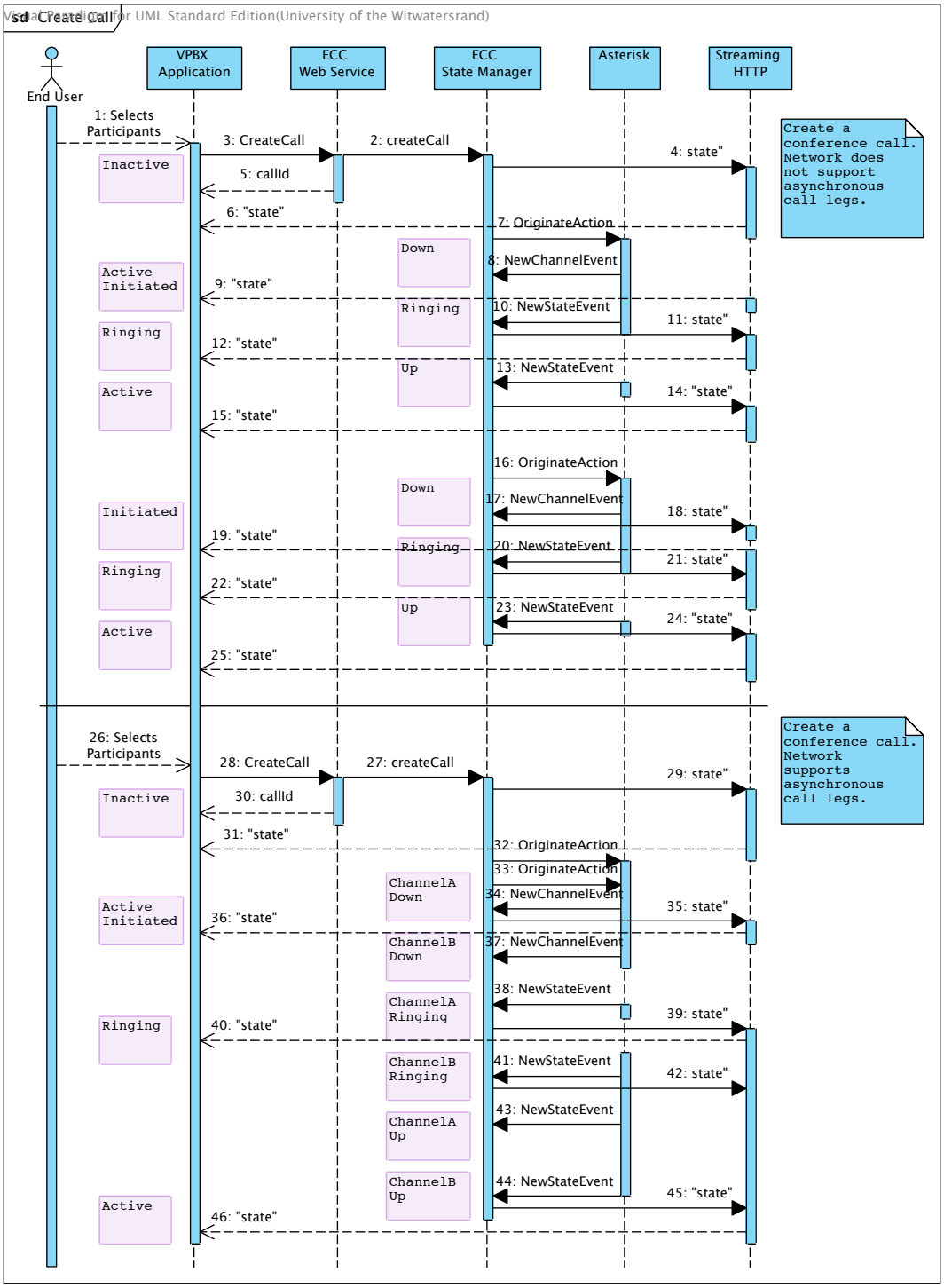


Figure 5.13: Extended Call Control Multi-Party Call Creation

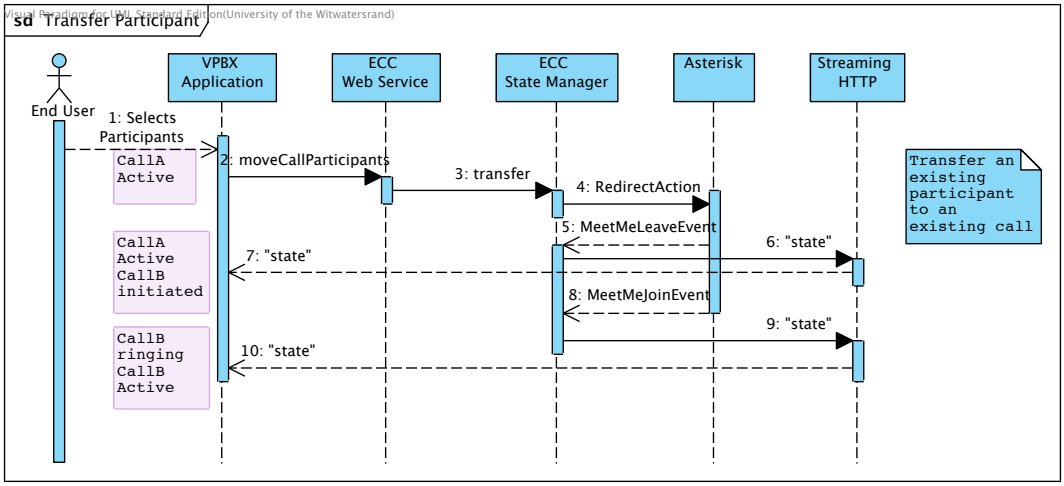


Figure 5.14: Extended Call Control Participant Transfer

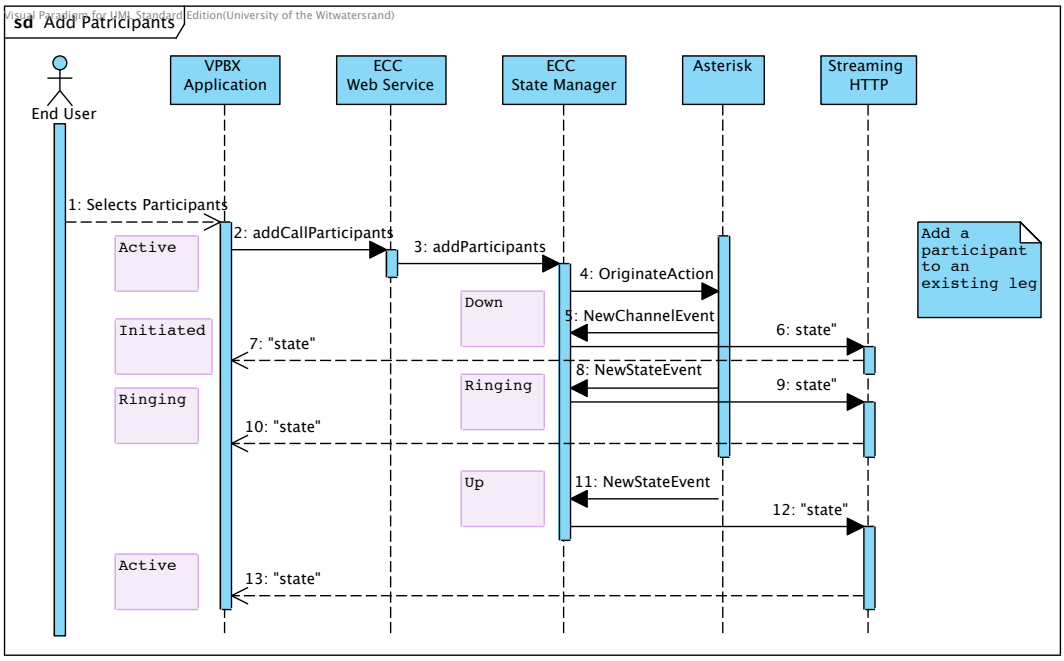


Figure 5.15: Extended Call Control Add a Participant

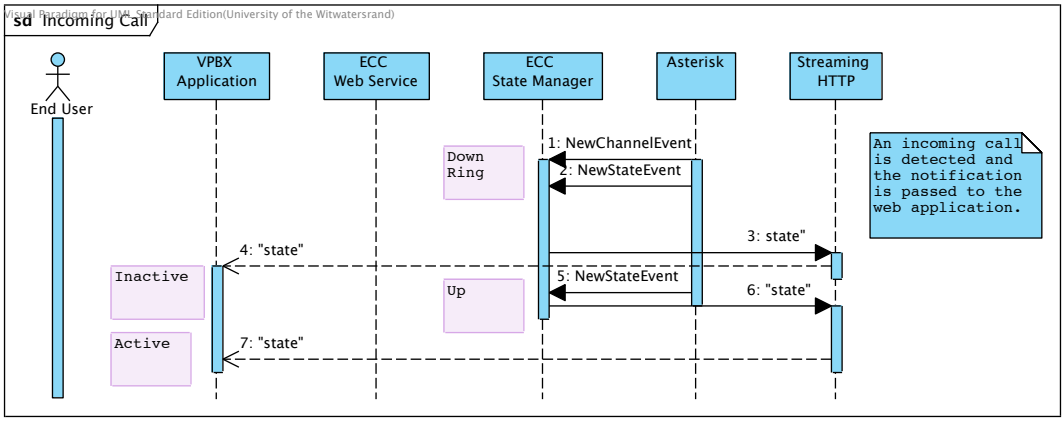


Figure 5.16: Incoming Call Notification

5.5.5 Incoming Call Notification

In the case of an incoming call, as shown in Figure 5.16, a `NewChannelEvent` notification is generated by the Network Emulator of a call to the number of interest, in this case a preset Asterisk meetme number. The incoming call notification causes the Network Emulator to notify the State Manager, which results in a new CALL object being created. The meetme extension within Asterisk transitions from `Down` to `Ring`, which is translated to the ECC state `Inactive` by the State Manager, message 3, and the application is notified, message 4. As the calling extension is placed in an Asterisk meetme conference, message 5, the extension transitions to `Up`, and the resulting `NewStateEvent` notification is mapped to the ECC `Active` state, message 6–7. At this point there is only the calling party, and the application would signal the operator to include a second party so that the call might be directed. This would follow the sequence as discussed in 5.5.4.

5.5.6 Disconnecting Participants

To remove participants from a call the participants are selected by the user, message 1, as shown in Figure 5.17. The overall Extended Call Control call state remains active whilst participants remain in the call, however notifications are received regarding the state of the participants if the application has requested such information, as shown in message 7. The ECC State Manager signals the Network Emulator to disconnect the participant via means of a `HangupAction` to the Network Emulator, as shown in message 4. The Network Emulator disconnects the extension and generates a `HangupEvent`, with an extension state of `Down`, message 5. The ECC State Manager records the change to

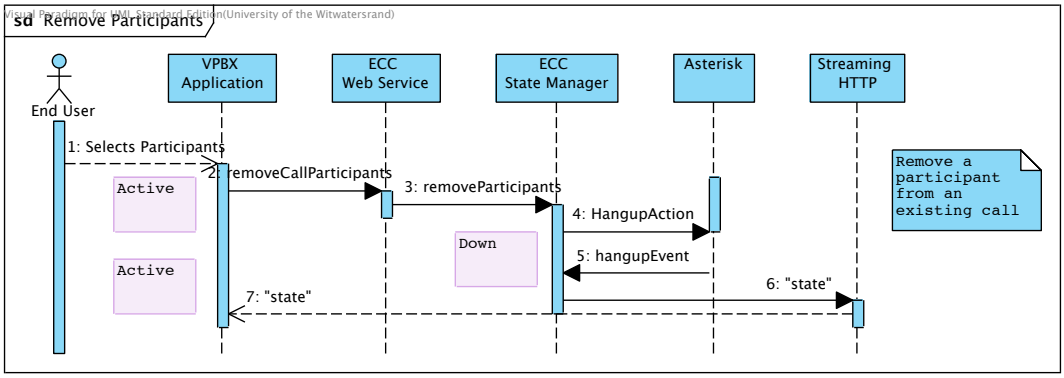


Figure 5.17: Extended Call Control Remove a Participant

the call and the application is informed asynchronously of the removal of the participant, however the overall state of the call does not change. If the last participant is removed from the call, then the call state would transition to `Inactive`, and the call would end.

5.5.7 Ending a Call

To end a call the user selects the call to end and the call identifier is passed by the `Web Requestor` VPBX application to the Extended Call Control Web service, message 1, Figure 5.18. The instruction is passed to the correct State Manager, message 3, and is translated into a number of `HangupAction` requests to the Network Emulator for all the extensions involved with the call. The Network Emulator disconnects the extensions and generates a `HangupEvent` for each extension, with an extension state of `Down`, messages 4–7. The State Manager ensures each participant is removed from the call, and once all extensions are removed, the ECC call state is updated to `Inactive`, and the application informed, messages 8–9.

5.6 Discussion

Implementation of the Extended Call Control Web service and call model can be performed using a number of different Web technologies, in this case JavaScript hosted on a Web application server. The example application, Virtual Private Branch Exchange demonstrates real time call and extension control functionality, incorporating asynchronous notification of changes in state.

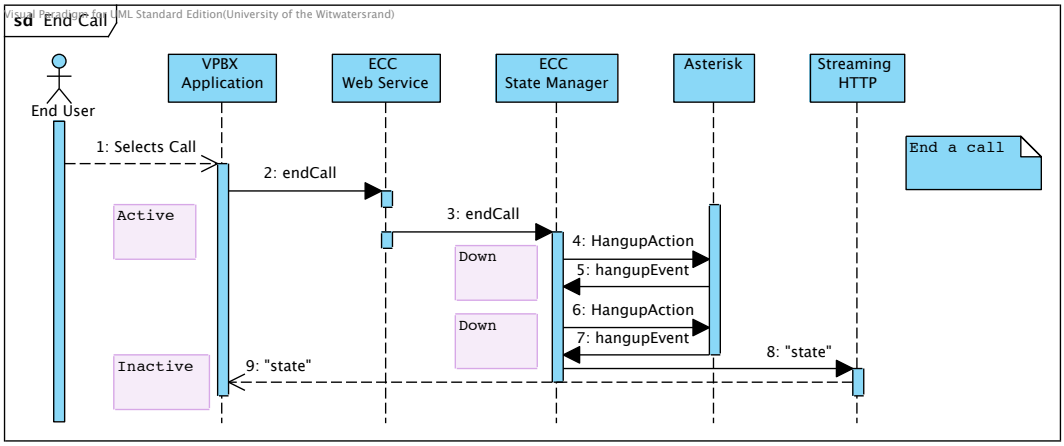


Figure 5.18: Extended Call Control End Call

Advanced control of the network requires knowledge of state of the network, so as to ensure that the Web application provides valid instructions to the underlying network, thus notification of state is a vital component of the implementation of an Extended Call Control Web service.

5.6.1 Web Service

The Extended Call Control API was implemented using a SOAP based Web service, the use of Glassfish provided a convenient container for the service and allowed distributed communication with the state manager. From the implementation it is clear that the Web service and the notifications can be separated, and that notifications can occur in a manner mutually exclusive of the Web server. For adequate call control, the Web service requires the following information to be able to successfully operate on a call in a stateful manner, using a stateful resource: connection identifier, call identifier, and resource identifier. The methods exposed by the Web service determine the level of functionality offered with regards to call control, and by means of common identifiers a successful mashup of services is possible. Unique resource identities allow a Web application to make use of an alternate Web service in the case of a Web server fault, allowing the operator to provide redundancy. The methods used in the Extended Call Control API provide an adequate level of control to implement advanced call control use cases required for a complex service. By catering for multiple (two or more) participants in the API methods, the number of invocations on a Web server can be reduced.

Asynchronous invocation of the Web service methods allow the Web application to receive

a call and connection identifier that is generated by the state manager, thus ensuring the independence of the Web service from the state manager.

The use of a resource identifier removes the need for the Web server to relate requests to a particular session, thus providing the possibility of further simplifying the Web service to be a RESTful Web service.

5.6.2 Network Emulator

The use of Asterisk for the proof of concept implementation provided adequate functionality to demonstrate the use of the Extended Call Control call model. The Asterisk notifications were successfully mapped to the Extended Call Control call model events, demonstrating the usefulness of the Extended Call Control call model in providing an abstracted representation of the call. A large volume of notifications generated by the Asterisk server were irrelevant to the Extended Call Control call model and were filtered by the state manager. The Asterisk extension state mapped to the Extended Call Control call model, and extension notifications were mapped successfully by the state manager. Notifications were generated at a very rapid interval, far faster than the one notification per second as recommended by RFC 4235.

5.6.3 State Manager

The state manager represents the stateful resource. The implementation showed how the stateful resource, which is assumed to be a robust application within the service delivery platform, can successfully map state notifications received from the network to those required by Web applications performing call control. The use of a resource identifier within the Web service allows the correct state manager to be identified, and the state of all associated connections is maintained within the state manager. The use of a notification server is required to interface the Extended Call Control notifications to the Web Application. Each Extended Call Control user would require a separate instance of a state manager to maintain state for the extensions governed by that particular user, implying that Extended Call Control can only be provided by creating a relationship between the user and service provider, in terms of a service level agreement.

5.6.4 Application

The use of a Web JavaScript application ensures that the application can be upgraded without requiring reinstallation from the client side. Web applications can achieve seemingly asynchronous behaviour by making use of AJAX, allowing the browser to defer response handling to an underlying layer. The use of streaming HTTP provides a continuous connection between the Web server and requestor, thereby allowing notifications to be pushed to the application. In the case of Extended Call Control it was found that the nature and volume of notifications was such that only asynchronous streaming HTTP is suitable to update the client state continuously. The display of the calls and participants is required for advanced call control as the individual state of a connection is useful when performing call leg control. Using SOAP based Web services residing in multiple domains is problematic and requires security certificates or altering the security mechanisms on the browser directly. The state of a Web application is dependent on the state messages received and as such would suffer from high latency connections, therefore the use of a network orientated call model overcomes some of these limitations as the state manager is able to confirm the correct use of API methods based on known state.

5.6.5 Notification

Real time asynchronous notification of state is a fundamental requirement of Extended Call Control. Web Application based call control can only be performed successfully if the application is aware of the state of all calls and participants. The Lightstreamer AJAX notification service provided a means of demonstrating streaming HTTP, with a large volume of notifications. Notifications were generated in excess of one per second, and required that the server be licensed, as opposed to the limited unlicensed version. Using a separate server to forward notifications from the state manager to the Web application conformed to the distributed nature of the architecture and would permit redundancy in the case of failure.

5.6.6 Architecture

By separating notifications from the Web service, and using a separate resource from that of the Web server to manage state, the Web service can benefit from Web architecture redundancy, without effecting the operation of the call control. Streaming HTTP for notifications ensured that the Web application was aware of the state of the network, as directly reported

by the state manager. The state manager performed all call mapping and the Extended Call Control call model could map to the Asterisk channel state. By providing identifiers for the state manager, as well as the call and connections Web services can interoperate on the call by means of the common state manager as proposed in (Czajkowski et al. 2004).

5.7 Conclusion

The use of the Extended Call Control Web service and Extended Call Control call model was demonstrated in a proof of concept Web Application for a Virtual Private Branch Exchange. The VPBX Application and associated architecture demonstrates an operator mapping existing network technologies to the Extended Call Control call model. The proof of concept architecture provided the necessary notifications of state for the VPBX application to provide advanced call control. Mapping of the Asterisk states to the Extended Call Control call model states demonstrated the abstraction provided by the call model.

Chapter 6

Conclusion

This chapter provides a summary and conclusion of this research and development of the Extended Call Controll. The novelty of this work, its limitations, and further work remaining is summarised.

6.1 Summary

This section summarises the requirements for Web based advanced call control using stateful Web services. Existing call models and how they apply to the Extended Call Control Web service and Extended Call Control call model are reviewed. A brief summary of the results of the implementation of the Extended Call Control Web service and Virtual PBX application is given.

6.1.1 Requirements for Advanced Web Service Call Control

Telecommunications operators are looking for ways to further allow third parties to make use of the network in an open manner, so that a greater number of applications using the network can be created. Third parties are able to develop services for customers, making use of the telecommunication network. Being able to provide services controlling calls is known as call control, and exposes the fundamental business of an operator. Operators are increasingly exposing their functionality through the Web, making use of Web services. This research is focused on Web based call control, and how the operator can provide advanced functionality similar to that achievable by tightly coupled third party APIs.

Web services provide a means to remove the tight coupling between third party applications and network operators, and provide unmatched interoperability and integration. The limitations of network proxies and firewalls is reduced when using Web services between disparate domains. Web services, in addition, provide telecommunications operators with platform and language independence, removing the need for technologies such as SIP and CORBA. Also by the operator exposing telecommunications functionality through Web services, third parties can enhance the functionality with other Web services, providing an application that can interwork with a conglomeration of multiple dynamically linked Web services.

SOAP based Web services provide a mechanism for third parties and operators to interwork systems regardless of the network transport protocol, and allow systems to recover quickly from failure due to the fact that standard Web services do not require any data to be associated with a particular session.

Control of a telecommunications network by means of a Web application introduces a number of requirements to support such control, specifically how to abstract the view of the network as well as how to provide abstracted APIs that do not require the developer of the application to have detailed knowledge of the operation of a telecommunications network.

Existing Web based call control is generally of a simple message exchange type, such as invoking the network to send an SMS, or making a request for an action and then determining the outcome of the request. Lack of knowledge of the state of the underlying network limits the effectiveness of Web applications and the level of control that can be exerted on the network. The ability to provide complex services is directly proportional to the complexity of the knowledge of state and this research develops a new call model to represent call state in a manner suitable for Web applications. As the application processing spans an increasingly larger time span, so to does the need for preservation of the knowledge of the state of a process. Web applications can provide advanced call control applications due to knowledge of the state of the network, and offer third party developers more complex control of resources than what is currently offered by network operators.

This research set out to provide various levels of control and abstraction of network functionality, which it achieved through the Extended Call Control call model and Web service. The ability for an operator to map existing network technologies to the Extended Call Control call model is demonstrated, providing abstraction of complex systems to a level suitable for Web applications. Many existing Web based call control methods cater only for two party connections, as the level of complexity when dealing with multiple parties simul-

taneously increases the requirements on state to a level that is burdensome for standard Web applications and Web services. The control of multiple parties is therefore a requirement for advanced call control and reinforces the requirement for a call model to represent the state of multiple parties as in the Extended Call Control call model. A Web application that is performing advanced call control logic for the control of the telecommunication network requires the application to participate in the control of the call for the entire duration of the call, and be able to perform operations on the network in an asynchronous manner, requiring an asynchronous Web service API as well as asynchronous notification of the state of the network.

Telecommunication service architectures can be broadly categorised into three technologies; namely SIP for IP based networks, Parlay for IP and circuit switched networks and CAMEL for IN type circuit switched networks. The mapping of these technologies is demonstrated for the Extended Call Control call model, showing the suitability of the call model for Web based call control.

6.1.2 Stateful Web Services

Whilst some call control Web services make use of conversational state, in this thesis an architecture using a stateful resource to maintain the state of the network resources is proposed, as proposed in the WS-Resource Framework. The use of stateful resources together with an asynchronous Web service provides a means to preserve many of the benefits of a stateless Web service, as dynamic state is not stored in the Web service itself, rather in the state manager which is tightly coupled with the underlying network.

The implementation of a separate resource to provide notifications as to the state of underlying resources permits the Web application to have a current view of the network, without wasting resources having to poll for the status of a call.

A key component of the implementation of stateful resources is the correct identification of the resource that is in consideration, as all responsibility for state is maintained by the stateful resource. The stateful resource identity is passed to the requestor by the Web service, which serves to correlate the request with the correct stateful resource. A stateful resource can be associated with multiple Web services facilitating the robustness that is required by network operators.

The state of the Call is managed by the Web application through the Web service, thus eliminating the need for the Web application to have specific knowledge of the identity or

location of the encapsulated system. The methods available in the Web service determine the level of call control a Web application can exert on the network. This research has identified required functionality common to advanced call control in the Extended Call Control Web service.

6.1.3 Existing Call Models

In order to develop the Extended Call control call model an analysis of existing telecommunications call models was necessary, to determine the best practices and concepts for inclusion in a Web based call model. The fundamental concepts of call control were reviewed such as first and third party call control and the description of the state from a perspective of where state information is stored and how the finite state machines of different parties relate to each other. The symmetric and asymmetric nature of the finite state machine used to describe originating and terminating parties were reviewed with respect to the states permitted in the state machine.

The IN basic call state model was reviewed as the cornerstone of call control, and the basis for all notifications originating within the network, with a mapping between events in the BCSM and the Extended Call Control call model being a requirement of the call model.

JTAPI provided an opportunity to consider a call model that does not specifically cater for suspension of call processing and invocation of application logic, and the states used to describe the processing of the call were useful as a basis for a number of Extended Call Control states, and notifications. The use of an asymmetric call model not differentiating between originating and terminating parties provides a level of abstraction useful for Web based call control, and the use of events to report changes in state to the controlling application is borrowed for the Extended Call Control call model.

The JCC/JCAT call model does not impose a limitation on the number of parties in a model, and represents all parties with the same finite state machine. The use of a full call model to provide a complete view of all parties in the call is borrowed for the Extended Call Control call model. Complexity such as application defined routing through the network is identified as a feature that is not required and is abstracted within the Extended Call Control call model. The use of additional methods to provide extended functionality leads to the independence of the Extended Call Control call model from the API, as sufficient information is shown to be present within the Extended Call Control to cater for various levels of complexity of APIs.

The Parlay API and call model are application centric, in that the only parts of the call that are of specific interest to the application are defined within the call model. Call control can be deleted if no further control of the call is required. The Extended Call Control call model borrows from this in providing a view of the call as perceived by the end user, however it uses a state model that is dependent of network notifications, such that the call state is not dependent on the presence of a Web application.

RFC 4235 defines a call model for support of state dialogues for SIP. The concept of the virtual state model being representative of the entire state of all participants is used within the Extended Call Control call model.

The mapping between the circuit switched and packet switched call models of IN, Parlay and SIP show how the interoperation of call models is possible.

6.1.4 Extended Call Control Web Service and Call Model

Based on the requirements laid out in Chapter 1 existing call control call models were examined in Chapter 3. Abstraction and identification of suitable states for a Web based call model were determined and from these the Extended Call Control call model was created. The events occurring within the network were mapped to notifications causing a transition between states in the Extended Call Control call model. The Extended Call Control call model abstracts the complexity of the state of individual terminal connections, and has the ability to provide such information as required by the Application. The state of the terminal in terms of the media is abstracted, as well as the relationship between the terminal and network. Common identifiers allow unique identification of the call as well as connection. By considering use cases as well as existing call control APIs such as Parlay-X an Extended Call Control API is proposed that meets the requirements for advanced call control as laid out in Chapter 1.

6.1.5 Implementation of Virtual Private Branch Exchange Web Application

Chapter 5 presented a proof of concept application using the Extended Call Control call model. The implemented virtual private branch exchange application as described within the Extended Call Control call model Chapter 4, provided a method of validating the concepts of the call model and associated API. The proof of concept architecture provided the necessary notifications of state for the application to provide advanced call control.

6.2 Extended Call Control Web Service and Call Model Contribution

A Web application providing advanced call control requires knowledge of the underlying network that it is controlling. This representation of the state of the underlying network is done using a call model. In this research an Extended Call Control call model and API for call control Web applications is proposed. The developed Extended Call Control Web service provides third parties with an opportunity to incorporate advanced call control in their Web applications, adding value to existing and new service mashups.

This Extended Call Control call model, synthesised from the analysis of existing call models, provides the following benefits:

- It provides abstraction of underlying resources,
- It is sufficiently detailed to interwork with existing call models.
- It can represent multiple parties in an abstracted manner.
- It can abstract circuit-switched and packet-switched networks. Mappings between the notifications for the Extended Call Control call model and various network service architectures have been performed to demonstrate this ability.

A method of identifying and abstracting telecommunications functionality for advanced call control was shown. Using method discovery to abstract functionality present in message sequence charts for complex call control scenarios the Extended Call Control Web service APIs were determined.

This Extended Call Control Web service API and call model was demonstrated in a proof of concept Web application for a Virtual Private Branch Exchange. The architecture of the implementation facilitates separation of the Web service and the call model, facilitating the distribution of functionality and redundancy.

Provision for identification of the call after creation as well as identification of a stateful resource permitted asynchronous stateful Web service operation, providing the Web application with the ability to perform mid call operations such as moving a party from one call to another during the course of the call.

The use of streaming HTTP provided the Web application with state notifications ensuring that the state of the underlying network and the controlling application were identical, a requirement for the Web application to provide control of the network.

Maintenance of state and the corresponding mapping of the state was performed by the state manager, which reduced the complexity of the implementation of the Web service, and provided distribution of functionality.

6.3 Research Limitations

The Extended Call Control Web service assumes Third Party call control. Thus the network is assumed to be in control of the destination leg of the call, requiring that the entity responsible for the mapping and implementation of the Extended Call Control Web service has sufficient knowledge of the destination legs progress to update the Extended Call Control call model, and to perform network level call control on that leg. The functionality available to the Extended Call Control Web service is dependent on the underlying telecommunications operator equipment and service architecture.

The network operator has to create triggers within the network to flag the processing of calls for extensions falling within the control of an Extended Call Control Web application. Service level agreements between the Web Application user and the network operator would have to be performed before extensions could be identified for the Extended Call Control Web service.

As the element responsible for management and mapping of network call state to Extended Call Control call state requires information about each party in a call, the memory requirements for such a system can grow at a rapid rate. Memory requirements are not a limiting factor for asynchronous Web services, as Web applications would be distributed amongst a number of such elements. Rather reducing the quantity of Web application requests is of a higher priority, a feature which asynchronous Web services provide compared to synchronous Web services. Implementation of such an element would require telecommunications operator level hardware, such as a Parlay Gateway. The distribution of resources as proposed in the implementation permits state to be maintained in multiple entities, however such state would be lost should the state manager fail. Protection of state and robustness of implementation were not considered in this proof of concept implementation.

Various browsers implement and interpret JavaScript in a non reliable manner and JavaScript applications can behave slightly differently. In addition security policies on browsers can change depending on the browser and end user settings. Signed security certificates for the Web application and robustness of the Web application were not considered in this proof of concept.

The Virtual Private Branch Exchange Application and proof of concept implementation considers every call to be a multi party call, requiring a large amount of resources within the Asterisk server emulating the network. Treating two party calls as multi party calls within a telecommunications network is wasteful of resources, and this research does not detail how a telecommunications operator could implement such a service on a service delivery platform.

6.4 Future Work

The Extended Call Control is a SOAP based Web service, however RESTful Web services are gaining popularity for a number of reasons. Some of these reasons include the following:

- The simpler manner in which the service can be accessed without having to use SOAP encapsulating technologies, or development toolkits.
- The transparency of the requests via HTTP, allowing firewall inspection without SOAP packet unpacking.
- Authentication using industry-standard certificates and common identity management systems, such as a Lightweight Directory Access Protocol server.

The choice to implement a RESTful interface for advanced call control as opposed to a SOAP based Web service is a subject that requires further investigation.

Security of Web services is a topic that requires thorough research. Security and authentication are not the focus of this research and further consideration is to be given to this topic before an implementation can be performed. In addition authorisation to use a Web service such as the Extended Call Control Web service and the requirements for such authorisation requires understanding of Service Level Agreements and the resources for such.

By means of the Extended Call Control Web service additional advanced call control applications can be developed to provide functionality that was previously only available to a PBX. An example of such an application could be a Web based call centre application. Performance and scalability of such a Web Application is of great interest, especially when compared to existing call control Web services such as provided by Twilio.com and Voxeo.com.

The problem of concurrency and synchronising call state over high latency connections has not been a topic of this research, and requires further study, especially in the case of

a disconnection in which the Web application could lose connectivity from the streaming notification server, in which case the Web application is no longer performing call control. Possible methods of ensuring concurrent state could include a method to request an update of state, based upon a provided previous state identification.

References

Bayer, M.: 2000, *Computer Telephony Demystified: Putting CTI, Media Services and IP Telephony to Work*, 1 edn, McGraw-Hill Professional.

Bittner, K.: 2000, Why use cases are not "functions", *The Rational Edge* . http://download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/dec00/WhyUseCasesAreNotFunctionsDec00.pdf Last accessed: 06-2010.

Caprio, G. D., Minerva, R. and Moiso, C.: 2004, A New Foundation for Future Networks: Web Services + Network Identity + Context Awareness, *Proceedings of 9th International Conference on Intelligence in Next Generation Networks, ICIN*, Vol. 1, Bordeaux, France, pp. 7–12.

Caprio, G. D. and Moiso, C.: 2003, Web Services and Parlay: an Architectural Comparison, *Exp in Search of Innovation* **3**(4). http://exp.telecomitalialab.com/articolo.asp?id=4&idart=342 Last accessed: 06-2005.

Czajkowski, K., Ferguson, D. F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S. and Vambenepe, W.: 2004, The WS-Resource Framework, *Technical Report 1.0*, Computer Associates International. http://www.globus.org/wsrf/specs/ws-wsrf.pdf Last accessed: 01-2010.

da Silva, J. S. T., Hassan, K., Glitho, R. and Khendek, F.: 2004, WEB Services for Conferencing in 3G Networks: A Parlay Based Implementation, *Proceedings of 9th International Conference on Intelligence in Service Delivery Networks, ICIN*, Bordeaux, France.

Dobrowolski, J., Grech, M., Qutub, S., Unmehopa, M. and Vemuri, K.: 1999, Internet Draft: Call Model For IP Telephony, *Technical report*, IPTel Working Group. http://www.cs.columbia.edu/sip/drafts/draft-dobrowolski-call-model-00.txt Last accessed: 05-2009.

Dobrowolski, J., Montgomery, W., Vemuri, K., Voelker, J. and Brusilovsky, A.: 1999a, Internet Draft: Call Model Integration Framework. http://tools.ietf.org/html/draft-vemuri-cmi-framework-00 Last accessed: 03-2009.

Dobrowolski, J., Montgomery, W., Vemuri, K., Voelker, J. and Brusilovsky, A.: 1999b, Internet Draft: IN Technology for Internet Telephony Enhancements. http://tools.ietf.org/html/draft-dobrowolski-iptel-in-00 Last accessed: 01-2010.

ECMA: 2004, Standard ECMA-269: Services for Computer Supported Telecommunicaiton Applications (CTSA) Phase III. http://www.ecma-international.org/publications/standards/Ecma-269.htm Last accessed: 04-2010.

Esposito, D.: 2002, *Building Web Solutions with ASP.NET and ADO.NET*, first edn, Microsoft Press, Washington, USA.

ETSI and Parlay: 2005a, Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control; Sub-part 2: Generic Call Control SCF (Parlay 5), *Technical report*, European Telecommunications Standards Institute, The Parlay Group.

ETSI and Parlay: 2005b, Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control; Sub-part 3: Multi-Party Call Control SCF (Parlay 5), *Technical report*, European Telecommunications Standards Institute, The Parlay Group.

ETSI and Parlay: 2007, Open Service Access (OSA); Parlay X Web Services; Part 2: Third Party Call (Parlay X 3), *Technical report*, European Telecommunications Standards Institute, The Parlay Group.

Faynberg, I., Shah, N. J., Gabuzda, L. R. and Kaplan, M. P.: 1996, *The Intelligent Network Standards: Their Application to Services*, McGraw-Hill Professional.

Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W. and Weerawarana, S.: 2004, Modeling Stateful Resources with Web Services, *Technical Report 1.1*, Computer Associates International.

Freeman, A., Jones, A. and Corp, M.: 2002, *Microsoft .Net Xml Web Services Step by Step*, Microsoft Press, Redmond, WA, USA.

Fricke, B. and Hanrahan, H.: 2006, The Development of a Structured Approach to Service Provisioning in a Parlay Environment, *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): The network @ work*, Western Cape, South Africa.

Graf, M.: 2000, An Introduction to the Java Telephony API (JTAPI), *Technical report*, IBM Research Division. http://www.zurich.ibm.com/csc/distribsys/j323/jtapi-tutorial.pdf Last accessed: 03-2005.

Graham, S., Niblett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Tuecke, S., illiam Vambenepe and Weihl, B.: 2004, Whitepaper: Web Services Notification (WS-Notification), *Technical Report 1*, International Business Machines Corporation.

Hagel, J.: 2002, Loosely Coupled: A Term Worth Understanding, *Technical report*. http://www.johnhagel.com/blog20021009.html Last accessed: 01-2010.

Hanrahan, H.: 2007, *Network Convergence: Services, Applications, Transport and Operations Support*, first edn, Wiley, New Jersey, USA.

Hirsch, F., Kemp, J. and Ilkka, J.: 2006, *Mobile Web Services: Architecture and Implementation*, Wiley, West Sussex, England.

Hogg, K., Chilcott, P., Nolan, M. and Srinvasan, B.: 2004, An evaluation of Web services in the design of a B2B application, *Proceedings of the 27th conference on Australasian computer science*, Australian Computer Society, Inc., pp. 331–340.

IBM Web Services Architecture Team: 2000, Web Services architecture overview: The next stage of evolution for e-business, *Technical report*, IBM. www-106.ibm.com/developerworks/webservices/library/w-ovr/ Last accessed: 07-2008.

ITU-T: 1993, Intelligent Network Distributed Functional Plane Architecture, *Technical Report ITU-T Recommendation Q.1204*, International Telecommunication Union.

Jain, R., Anjum, F. M., Missier, P. and Shastry, S.: 2000, Java Call Control, Coordination, and Transactions, *IEEE Communications Magazine* pp. 108–114.

Jain, R., Bakker, J.-L. and Anjum, F.: 2005, *Programming Converged Networks*, first edn, John Wiley & Sons, Washington, USA.

Jepsen, T. C. (ed.): 2001, *Java in telecommunications: solutions for next generation networks*, John Wiley and Sons Ltd., Chichester, UK.

Johnston, A. B.: 2003, *SIP : Understanding the Session Initiation Protocol*, second edn, Artech House, Boston, MA, USA.

Jønvik, T. E., Vanem, E. and Thanh, D. V.: 2003, Business opportunities for Telecom operators with XML Web Services, *Proceedings of 8th International Conference on Intelligence in next generation Networks, ICIN*, Vol. 5A, Bordeaux, France, pp. 21–25.

JTAPI: 2002, JSR-43 Java Telephony API (JTAPI) Specification 1.4 Final Release 2, *Technical report*, Java Community Process.

JWG: 2002, The Parlay Group, *Technical report*, Parlay 4.0 :Parlay X Web Services White Paper.

Kaye, D.: 2003, *Loosely Coupled: The Missing Pieces of Web Services*, first edn, RDS Associates Inc, Washington, USA.

Kruchten, P.: 2003, *The Rational Unified Process: An Introduction*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Mahemoff, M.: 2006, *Ajax Design Patterns*, O'Reilly Media, Inc., CA, USA.

Marshall, W., Ramakrishnan, K., Miller, E., Russell, G., Beser, B., Mannette, M., Steinbrenner, K., Oran, D., Andreasen, F., Pickens, J., Lalwaney, P., Fellows, J., Evans, D. and Kelly, K.: 2001, Internet Draft: SIP Extensions for supporting Distributed Call State, *Technical report*, SIP Working Group. `http://tools.ietf.org/html/draft-ietf-sip-state-02` Last accessed: 03-2010.

McCarthy, P.: 2005, Ajax for Java developers: Build dynamic Java applications: Ajax paves the way for better Web applications. `http://www.ibm.com/developerworks/webservices/library/j-ajax1/` Last accessed: 05-2010.

Newcomer, E.: 2002, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, first edn, Pearson Education, Indianopolis, USA.

Powell, M.: 2002, Using ASP.NET Session State in a Web Service, *Technical report*, Microsoft Corporation. `http://msdn.microsoft.com/en-us/library/aa480509.aspx` Last accessed: 01-2009.

Reuter, S.: 2010, Channelstate (asterisk-java 1.0.0.ci-snapshot api), *Technical report*, Asterisk-Java. `http://asterisk-java.org/development/apidocs/org/asteriskjava/live/ChannelState.html` Last accessed: 06-2010.

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and Schooler, E.: 2002, SIP: Session Initiation Protocol, RFC 3261, *Network Working Group* . `http://www.packetizer.com/rfc/rfc.cgi?num=3261` Last accessed: 09-2009.

Rosenberg, J., Schulzrinne, H. and May, R.: 2005, Internet Draft: An INVITE Initiated Dialog Event Package for the Session Initiation Protocol (SIP). `http://tools.ietf.org/id/draft-ietf-sipping-dialog-package-06.txt` Last accessed: 07-2008.

Rosenberg, J., Schulzrinne, H. and Mayh, R.: 2005, An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP), RFC 4235, *Network Working Group* . `http://www.packetizer.com/rfc/rfc.cgi?num=4235` Last accessed: 09-2009.

Singh, M. P. and Huhns, M. N.: 2005, *Service-oriented Computing - Semantic, Processes, Agents*, John Wiley & Sons, Ltd, West Sussex, UK.

Sun Microsystems: 2002, Java Call Control (JCC) Application Programming Interface (API): Version 1.0b: Overview of the API, *Technical report*, Sun Microsystems, Inc. `http://java.sun.com/products/jain/api_specs.html` Last accessed: 09-2009.

Sun Microsystems: 2003, JCAT API: Standard Java™Interface to the Java CAll control exTensions (JCAT) : Release 0.3.1, *Technical report*, Sun Microsystems, Inc. `http://java.sun.com/products/jain/api_specs.html` Last accessed: 09-2009.

Vannucci, D. E. and Hanrahan, H. E.: 2005a, Analysis of State Models for Telecommunication Services, *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): IP Generation leading the charge for convergence*, George, South Africa.

Vannucci, D. E. and Hanrahan, H. E.: 2005b, Stateful Web Services in Intelligent Networks, *ICT 2005 - 12th International Conference on Telecommunications*, IEEE Region 8.

Vannucci, D. E. and Hanrahan, H. E.: 2006, IN-OSA Call Model Mapping for Circuit Switched Interworking with IMS, *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): Next Generation Services - The Network @ Work*, Western Cape, South Africa.

Vannucci, D. E. and Hanrahan, H. E.: 2007a, Extended Call Control Telecom Web Service, *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): Next Generation Services - business models@work*, Mauritius.

Vannucci, D. and Hanrahan, H. E.: 2007b, OSA/Parlay-X Extended Call Control Telecom Web Services, *Proceedings of 11th International Conference on Intelligence in Service Delivery Networks, ICIN*, Bordeaux, France, pp. 185–190.

Wang, F., Xu, G. and Chen, D.: 2010, Study of web services asynchronous invocation mechanism in information resource integration, *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, Vol. 13, pp. V13–14 –V13–17.

Weerawarana, S., Curbera, F., Leymann, F., Storey, T. and Ferguson, D. F.: 2005, *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

Appendix A

Papers

The following papers resulted from this research:

Vannucci, D. E. and Hanrahan, H. E.: 2005a, Analysis of State Models for Telecommunication Services, *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): IP Generation leading the charge for convergence*, George, South Africa.

Vannucci, D. E. and Hanrahan, H. E.: 2005b, Stateful Web Services in Intelligent Networks, *ICT 2005 - 12th International Conference on Telecommunications*, IEEE Region 8.

Vannucci, D. E. and Hanrahan, H. E.: 2006, IN-OSA Call Model Mapping for Circuit Switched Interworking with IMS, *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): Next Generation Services - The Network @ Work*, Western Cape, South Africa.

Vannucci, D. E. and Hanrahan, H. E.: 2007a, Extended Call Control Telecom Web Service, *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): Next Generation Services - business models@work*, Mauritius.

Vannucci, D. and Hanrahan, H. E.: 2007b, OSA/Parlay-X Extended Call Control Telecom Web Services, *Proceedings of 11th International Conference on Intelligence in Service Delivery Networks, ICIN*, Bordeaux, France, pp. 185–190. Presented in Appendix B.

Appendix B

OSA/Parlay Extended Call Control Telecom Web Services

OSA/Parlay-X Extended Call Control Telecom Web Services

David E Vannucci & Hu E Hanrahan
Centre for Telecommunications Access and Services *
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg, South Africa
Tel:+27117177229 Fax:+27114031929
{d.vannucci, h.hanrahan}@ee.wits.ac.za

Abstract – This paper investigates the requirements for a Parlay-X Extended Call Control Web Service, and identifies some of the current shortcomings of call control specifications. A call model suitable for Extended Call Control is developed and tested against A Virtual Private Branch Exchange application to illustrate how advanced call control can be facilitated using such a Web Service. A methodology to develop Extended Call Control methods is proposed based on identification of use cases.

Index Terms – OSA/Parlay, Extended Call Control, Web Service, Convergence, Call Model, Virtual PBX

I. Extended Call Control

Extended Call Control is a proposed Web Service for advanced control of telecommunications networks. This Web Service was proposed in September 2005 by Ericsson as a possible solution to the current lack of advanced call control available to Web Service developers [1]. Extended Call Control is currently being developed by the Parlay Group, and a number of Parlay-X API Accelerator meetings have been held to determine the requirements for such a Web Service.

Extended Call Control aims to allow a large set of previously unavailable operations such as individual control of call participants (or legs), control of network resources such as interactive voice response units and functionality such as conference support. This Web Service has to foremost conform to the paradigms of Web Services, such as maintaining abstraction [2], provide loose coupling between the application and the Web Service and a suitable level of abstraction.

Extended Call Control is intended to provide facility for notification of both subscribed notifications and triggered network level notifications. In addition the web application has control and state knowledge over the entire duration of the call, as opposed to current call control Web Services where there is no notification of the call once created. In the current Parlay-X Third Party Call which incorporates elements of Extended Call Control, the actual call is established asynchronously within the network and no notifications are set to the application regarding the state of the call progress within the network [3], [4].

*The Centre is supported by Telkom SA Limited, Siemens Telecommunications, Vodacom South Africa, Telsaf Data and the THRIP Programme of the Department of Trade and Industry.

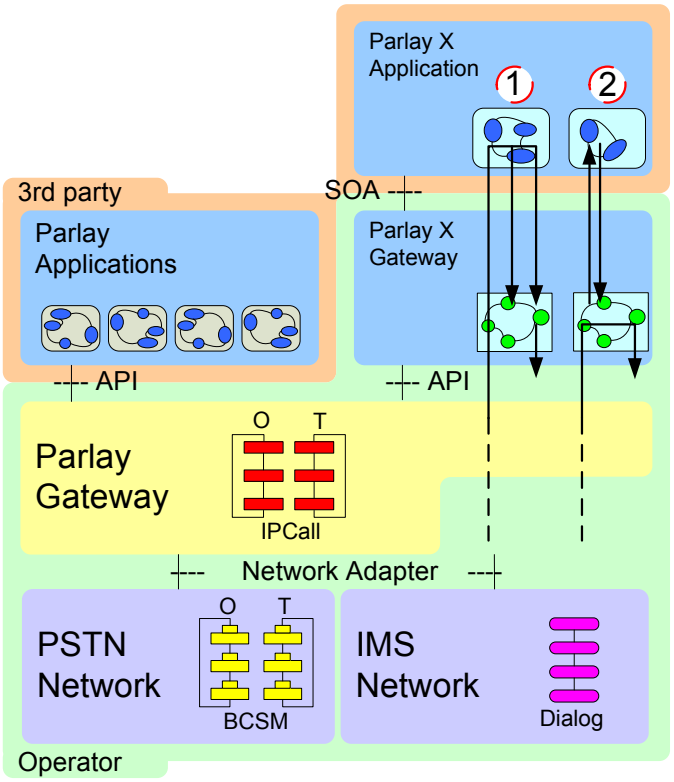


Fig. 1
Parlay X Architecture

Figure 1, from [5], shows the Parlay and Parlay-X architecture, where Parlay-X can be mapped to Parlay APIs or alternatively mapped directly to underlying network services. A Parlay-X Web Application is shown to illustrate the difference between an Extended Call Control Web Service and a normal Parlay-X call control Web Service, where the Extended Call Control application shown by the circle numbered 1 has control over the call for the entire duration of the call, whereas the normal Web Service, circle 2, operates in a request response manner with polling to determine the state of participants.

In this paper we consider the requirements for an Extended Call Control Web Service, allowing greater co-operation between Internet developers and telecommunication service providers. There is a clear need for Extended Call Control, however the requirements are

not fully defined in the standards. This paper provides an innovative method to develop an Extended Call Control Web Service in an extensible manner. In addition current state models for such a service are incomplete, often foregoing intermediate information such as whether the participant's phone is ringing. We propose a call state model suitable for an Extended Call Control Web Service. A reference example in section V is provided to illustrate how Extended Call Control can be used for a virtual private branch exchange application, and the call model used to reflect the state of the progress within the network.

II. Why Is Extended Call Control Necessary?

OSA/Parlay abstracts network resources (as shown in Figure 1) so that service application developers are not required to understand network protocols such as MAP, SIP, INAP and ISUP [6]. However the OSA/-Parlay interfaces are very rich in functionality and still require a fundamental understanding of telecommunication call control, messaging and database operations [6]. By providing abstracted Extended Call Control, fine grained call messaging and complex data operations are abstracted to the extent that these advanced operations can be provided in single method calls. For example creating a call does not require the application to register with the framework, authenticate itself, and create objects to handle the call. Rather instead a single method is invoked on the Web Service.

The aim of Extended Call Control is rapid service development by developers having Web Service skills, and not the specialised skills required for IN programming, allowing the number of application developers to move into the millions [7].

Parlay Call Control consists of five call control related Web Service specifications:Third Party Call, Call Notification, Call Handling, Audio Call, and Multimedia Conference. Complex control of calls requires combinations of these Web Services; however such functionality is difficult to combine when creating applications which operate outside of the intended use cases of the Web Services. The Parlay group has been creating common data types and revising the current call control specifications to better allow interworking of these Web Services, however data type limitations can hamper extensibility of call control Web Services.

III. What Are The Current Issues With Extended Call Control?

More powerful call control requires the application to know the state of parties and the operations within the network. This requires the use of state and an asynchronous model that, while in partial conflict with the Parlay-X programming model of simplicity and stateless behaviour, is seen as necessary.

Existing Call Control Web Services have a number of shortfalls, including:

1) In Call Notification not all call events are reported, and a call could fail due to unavailable resources or routing failure and the application would not know why the call failed [8].
2) Each call event notification is treated as a separate occurrence and the application cannot keep control over the call after the event handling [8].
3) Third Party call control sets up a call between two parties only [9]. However by allowing more parties in the initial call setup one could provide conference call setup capabilities.
4) To receive information regarding the status of the call, the application is required to explicitly poll the Call Control Web Service, creating a number of unnecessary messages, and periods of uncertainty [9], [10], [11]. Note that current draft specifications include a URL correlator as a callback reference.
5) The Multimedia Conference Web Service requires the application to add participants sequentially [11], and can cause a delay in the starting of the conference, since each request is handled synchronously. The conference organiser may be billed for time that is not utilised effectively.

By contrast Extended Call Control maintains control of the call over the entire call life cycle, thus requiring a more detailed model of the call and associated parties than what is currently provided in any given call control Web Service. This implies that the Web Application has substantial logic and no longer simply defines rules. An application using the Extended Call Control Web Service would have to be able to operate in a stateful manner, receive event notifications, and be able to be invoked on a network triggered event notification. Applications would also have to be able to change event subscriptions over the life of the call. Web Services have inherent delays and asynchronous behaviour is required to overcome this. When Call Control is passed from one application to another, say in the case of transferring control of a conference call, a unique identifier is required for each call. Support of multi-party calls also requires new call models [12]. As an Extended Call Control Web Service would allow manipulation of call media, a number of potential base Parlay mappings arise, and existing call control Web Services data structures are inadequate. In section IV we propose a call model suitable for Extended Call Control.

IV. Call Model

As discussed in section III, advanced call control requires knowledge of the state of the resources within

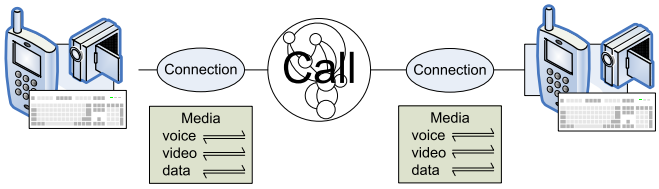


Fig. 2
EXTENDED CALL CONTROL CONNECTION MODEL

the network. Telecommunications service architectures uses call state models to keep track of the progress of each session and the services that are used during the sessions. The call model represents all the essential features of a session [13, pg. 39], and is a high level, technology independent abstraction of the call [5], [14].

Telecommunication services such as conference calling and call centre queueing require many messages to be exchanged, and control of the call is normally in place for the duration of the call. These services usually require the application to implement a call model to interpret these messages based on the last known state of the session [15]. Thus implementation of a call state model allows a far richer set of service functionality than that offered by a simple request response type [5].

An operational model similar to that of the ECMA [16] can be adopted to illustrate the relationship between the call object and associated connections of the call, as shown in Figure 2 .

Connections have the following attributes:

- Connection Identifier - Each connection has a unique identifier for a given call. This identifier can be based on the address of the participant. There are as many connections as participants.
- Call Identifier - Each connection has a reference to the identifier of each call with which it is involved. One connection could be involved in more than one call, for example a conference call.
- Media Stream - This attribute is as defined by Parlay Multimedia Call Control SCF. Media stream has data types, and those data types have a flow direction.

The proposed call state model for the call object in the connection model is shown in Figure 3. State transitions are observed by the service logic through event reports, caused by either service logic instructions or network notifications.

The following are the connection state definitions:

- `Inactive` - In this state the application is registered with the Extended Call Control Web Service, however no connections exist. Service logic is waiting for either application instruction or network notification.
- `Active` - The state where there is a relationship between a call and service logic, this can include zero or more connections. This state describes only the logical state of connections. Media streams are described in the connection information. A change in a connection would cause an update in state information.
- `Initiated` - A state in which the network is in the process of establishing a connection. In this state the call is in a pre-delivery state and service logic can accept, reject or redirect the attempted connection. Only once the connection is fully established would the call state transition to Ringing or Queued.
- `Queued` - A state in which call progression is suspended or made inactive by the network whilst a connection is being established. For example when a call is parked due to the line being busy, or when a call is queued waiting for an agent in a call centre application.
- `Ringing` - In this state the connection is waiting for confirmation from the participant.
- `Error` - It is possible for all states to transition to the Error state except for Inactive. From an Error state the call can resume its Active state such as when adding another party is attempted but fails, or move into an Inactive state.

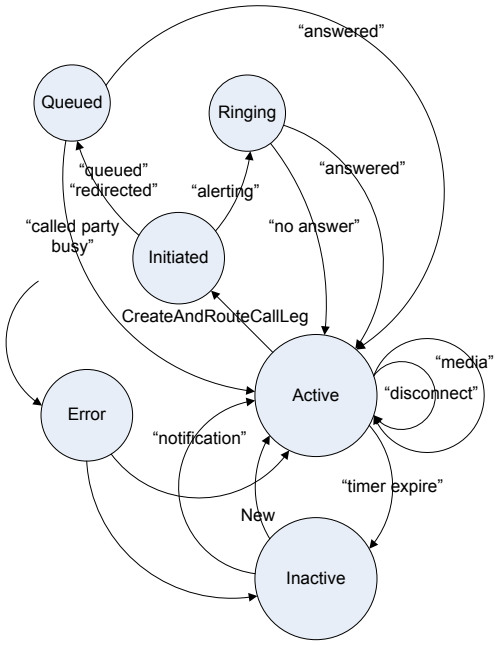


Fig. 3
EXTENDED CALL CONTROL CALL MODEL

V. VIRTUAL PBX EXAMPLE

An example of an Extended Call Control Web Application is a virtual receptionist's switchboard. This application would allow a company to operate in a completely

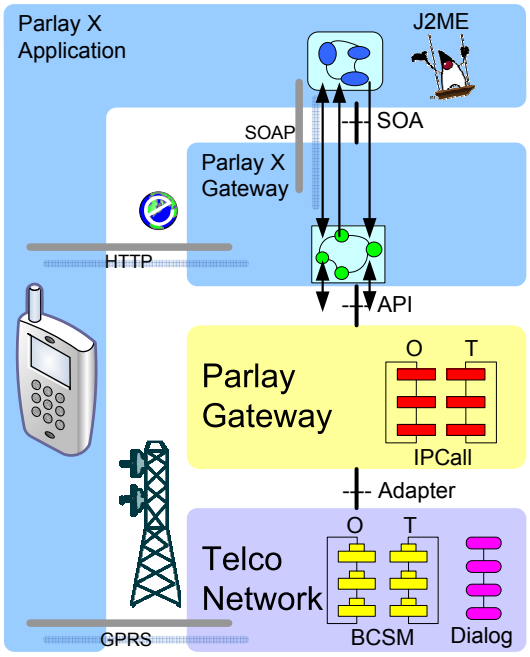


Fig. 4
PARLAY-X EXTENDED CALL CONTROL APPLICATION

distributed manner, with the receptionist having a Web Interface to control the company "extensions", which are standard mobile numbers.

The application would have knowledge of all the extensions and be able to provide advanced call control for any of them as required. It would be possible for the application to be embedded onto the mobile phone itself thus allowing the terminal to provide direct application layer service signalling to the Parlay X gateway as proposed in [17], as shown in Figure 4.

A number is assigned to the switchboard, and when callers call in to the main number, the receptionist's mobile phone rings, and the Web based application also reports an incoming call. The receptionist is able to perform standard and advanced private branch exchange functionality via the Web Application, including:

- Transfer the caller to a salesman's mobile number, either first conferring with the salesman or doing a blind transfer. Requires: Call Leg Control, Call Creation.
- Create a conference call between the caller, support and account salesman, and pass control of the conference to the salesman. Requires: Conference Call Creation and Control.
- Which extension is currently connected, and to whom? Requires: Call Status.

In the following example, as shown in Figure 5, we illustrate how these fundamental concepts can abstracted by an Extended Call Control Web Service and mapped to base Parlay APIs.

1: The Parlay-X Web Service creates an object implementing the IpAppConfCallControlManager interface. The call state transitions to `Inactive` since no connections exist at this time.

2: This message enables the Parlay-X Web service to receive notifications of new call events to the company number, afterwards passing those on to the application.

3,4,5: The conference is created, waiting for an incoming connection and an associated object is created. The Web Service requests to be notified of parties leaving the conference.

6: A caller calls the company switchboard number and the VPBX application is alerted that a call is incoming. This notification causes a transition to the `Active` state.

7,8,9: The caller is automatically assigned to a subconference by the Parlay gateway and this is obtained by the Extended Call Control Web Service, and a call leg object is created to represent the caller. The calling party is then joined to the conference by attaching the media. Note that when there is a change in media, the state model would reflect the change in connections.

10,11: The receptionist's mobile number is then added as a participant of the conference and a new IpMultiMediaCallLeg object is created. The call state transitions to `Initiated`, and later to `Ringing` due to the network notification.

12: Once the receptionist answers the phone, a notification is issued by the network, and the Extended Call Control Web Service notifies the VPBX application. The call state model transitions to the `Active` state with both connections established.

At this stage there is communication between the caller and the receptionist. The caller requests to be put through to both sales and technical support. The receptionist then puts the caller on hold.

13: The Extended Call Control Web Service splits the caller and receptionist so that there is no longer any communication between the two parties.

Once the caller is on hold the receptionist calls the salesman and technical assistant so that they can be informed of the situation.

14,15,16,17: Calls are created to both sales and technical. This results in a transition to the `Initiated` state again, as new connections are being formed.

18,19: Once parties answer, the VPBX application is notified, and the conference allows the parties to interact.

The receptionist is instructed to put the caller through. The `Active` call state is updated.

20: The caller is moved to the same conference as the sales and technical and the receptionist is free to disconnect.

21: The receptionist disconnects and the VPBX application is notified.

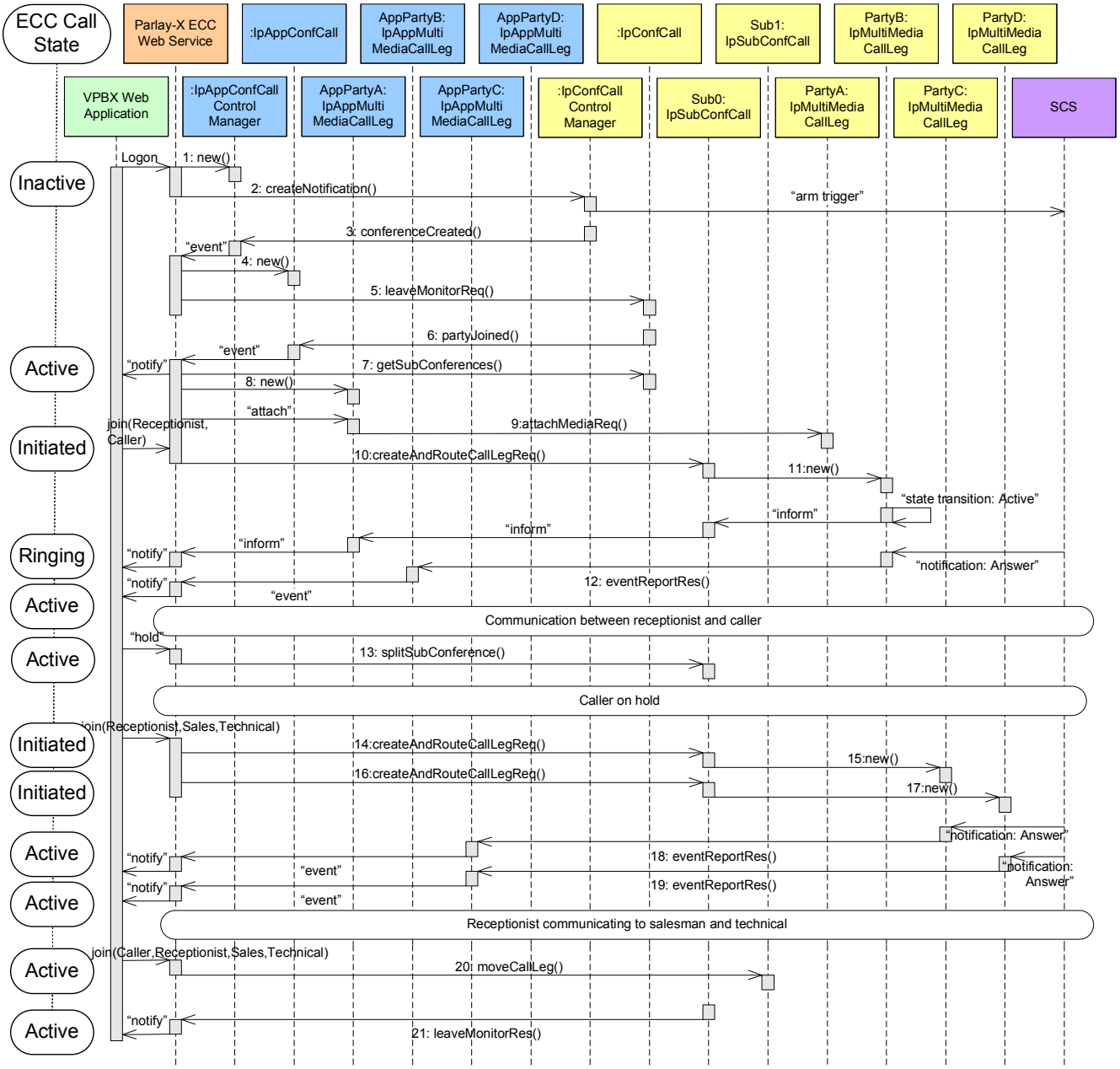


Fig. 5

VIRTUAL PBX EXAMPLE

VI. HOW IS EXTENDED CALL CONTROL ACHIEVED?

There are two approaches for an Extended Call Control specification, firstly extending the existing call control specifications and creating coherent cross specification data structures, or creating a new Extended Call Control specification, as shown in Figure 6.

Regardless of the approach used, basic requirements for Extended Call Control emerge when one considers an example Extended Call Control Web Service as shown in Figure 5, and when taking into consideration that the Web Service is in itself a type of Parlay application as shown in Figure 1.



Fig. 6

APPROACHES TO EXTENDED CALL CONTROL SPECIFICATIONS

Most important, the Web Application has to be able to maintain the state of the call and connections, and as such a call identifer is required that is unique for each call and a connection identifier. This would allow control to be passed from one Web Application to another, and

in the case of a failure be able to recover the session. This is being addressed by the draft Parlay-X call control Web Services where a session identifier and participant identifer are included [3]. In addition a web application using the Extended Call Control Web Service would have to be able to subscribe to receive notifications, in an asynchronous manner, thus providing knowledge of state transitions to the application as soon as they occur. This requirement has been addressed in the draft Parlay-X call control specifications, however a limited number of notifications has been chosen. The call has to be represented in a manner that, whilst maintaining a suitable level of abstraction, does not prevent the application from differentiating between the various parties and controlling parties or their connections. We believe a call model as shown in Figure 2 allows such control.

A. Extended Call Control Methods

Using the base Parlay APIs to determine characteristic message sequences in any given application, such as Multiparty of Conference Call Control. Use cases can be identified that lead to Extended Call Control methods. For the VPBX example, first the call and callback objects are created. These operations are encapsulated in a single method *Logon*, including associated notifications of completion, errors and failures. The Application should not be required to specifically create any call control objects, only indicate required connections to the Extended Call Control Web Service. Depending on the service agreement between the web application and the Web Service, charging plans are set automatically by the network. As is shown in messages 10 and 11, additional call legs are enabled as required by the Web Service, and a single *Connect* method is required to determine that the caller and receptionist have to be connected. Notifications arising from direct web application requests and relating to progress of the call would be reported as is shown when the receptionists phone rings and is answered to establish communication between the two parties. As is shown in message 13, the mapping to base Parlay APIs depends on the service level agreement of the service and the operator implementation, as one could also use the detachMediaReq() to place the caller on hold. In messages 14 and 16 new call legs are created as the receptionist's connection already exists. Thus, allowing more than one party in a Connect() method would provide an extensible number of parties.

VII. Conclusion

Extended Call Control aims to allow advanced control of a telecommunications network via a Web service. In this paper we have considered the requirements for an Extended Call Control Web service, allowing for greater co-operation between Internet developers and telecommunication service providers. This paper provides an innovative method to develop an Extended Call Control Web Service in an extensible manner, and proposes a call state model suitable for such a Web service. A reference example of a virtual private branch exchange application is provided. Current shortfalls with existing Call Control Web Services is examined, and key requirements for a new Web service identified.

References

[1] Ericsson. Meeting #32: Parlay X Call Control improvement, C5-050482. Technical report, Joint Working Group (Parlay, ETSI TISPAN Project OSA, 3GPP CT5), September 2005.

[2] Appium. Meeting #33: Parlay X 3.0 Enhanced Call Control, C5-050616. Technical report, Joint Working Group (Parlay, ETSI TISPAN Project OSA, 3GPP CT5), October 2005.

[3] ETSI. Open Service Access (OSA); Parlay X Web Services; Part 2: Third Party Call (Parlay X 3). June 2007.

[4] ETSI. Open Service Access (OSA); Mapping of Parlay X 2 Web Servicees to Parlay/OSA APIs; Part 2: Third Party Call Mapping; Sub-part 2: Mapping to Multi-Party Call Control. December 2005.

[5] D. E. Vannucci and H. E. Hanrahan. Extended Call Control Telecom Web Service. In *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): Next Generation Services - business models@work*, Mauritius, September 2007.

[6] H. Hanrahan. *Network Convergence: Services, Applications, Transport and Operations Support*. Wiley, New Jersey, USA, first edition, 2007.

[7] Z. Lozinski. Parlay/OSA and the Intelligent Network. February 2005.

[8] ETSI. Open Service Access (OSA); Parlay X Web Services; Part 3: Call Notification (Parlay X 2). December 2006.

[9] ETSI. Open Service Access (OSA); Parlay X Web Services; Part 2: Third Party Call (Parlay X 2). December 2006.

[10] ETSI. Open Service Access (OSA); Parlay X Web Services; Part 11: Audio Call (Parlay X 2). December 2006.

[11] Parlay ETSI. Open Service Access (OSA); Parlay X Web Services; Part 12: Multimedia Conference. March 2005.

[12] Ericsson BT Appium, ETRI. Meeting #34: Requirements Parlay X Etended Call Control, C5-060016. Technical report, Joint Working Group (Parlay, ETSI TISPAN Project OSA, 3GPP CT5), February 2006.

[13] R. Jain, J.-L. Bakker, and F. Anjum. *Programming Converged Networks*. John Wiley & Sons, Washington, USA, first edition, 2005.

[14] M. Graf. An Introduction to the Java Telephony API (JTAPI). Technical report, IBM Research Division, March 2000. `http://www.zurich.ibm.com/csc/distribsys/j323/jtapi-tutorial.pdf`.

[15] J. Dobrowolski, W. Montgomery, K. Vemuri, J. Voelker, and A. Brusilovsky. IN Technology for Internet Telephony Enhancements, December 1999. INTERNET-DRAFT draft-dobrowolski-iptel-in-00.txt.

[16] ECMA. Standard ECMA-269: Services for Computer Supported Telecommunicaiton Applications (CTSA) Phase III. March 2004.

[17] B. Fricke and H.E. Hanrahan. The Development of a Structured Approach to Service Provisioning in a Parlay Environment. In *Proceedings of Southern African Telecommunication Networks and Applications Conference (SATNAC): The network @ work*, Western Cape, South Africa, September 2006.