The Deployment, Design and Implementation of Voice Mail Feature based on DFC

Ruiguo Li

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in Information and Systems Science

Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada, K1S 5B6

©August, 2002

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-79791-0

Canada

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

**The Deployment, Design and Implementation of Voice Mail
Feature based on DFC**

submitted by

Ruiguo Li

in partial fulfillment of the requirements for the degree of
M.Sc. in Information and Systems Science

Professor B. Pagurek, Thesis Supervisor

Chair, Department of Systems and Computer Engineering

Carleton University

©August, 2002

Acknowledgements

I would like to thank my thesis supervisor, Professor Bernard Pagurek, for his guidance and support in this research. I would also like to thank him for believing in my ideas and devoting time and energy in challenging and inspiring discussions. I would like to thank to Professor Woodside for a lot of help. I would like to thank AT&T for providing the thesis project and financial aid.

I would like to thank my colleges in the Network Management Lab. for their friendship and support. I would like to thank Vicky Bell for her help.

Finally, I would like to thank my family for their love, support and understanding during the course of my studies.

Abstract:

Feature interaction has been identified as a major problem in the PSTN and it will be worse in IP telephony, since more new features are added in IP telephony. The two current predominant IP telephony protocols, H.323 and SIP, have not addressed this problem. Distributed Feature Composition (DFC) is a virtual architecture for the description of telecommunication services recently proposed by AT&T to modularize features, structure feature composition, analyze feature interactions and separate services and transmission layers. The modularized features, dynamic assembly of features in a pipe and filter pattern according to their priority in DFC makes analysis of feature interaction much easier. However, currently all features in the DFC run on the routers inside the network. This wastes a great deal of power in the end user devices, usually PCs, and significantly reduces the performance of the system when it is busy. This thesis will focus on the deployment of features in the end user device and how to manage these features. A new FDDP version of the DFC is proposed, with the addition of a local feature manager, a proxy of the user's line interface box and a Blind Call Transfer (BCT) feature. The FDDP version of the DFC has the following advantages:

- a) Dynamic loading of telephony features from the central server when they are needed;
- b) Features running inside the network can run on the end user device without any change;
- c) Users can subscribe to features on the network and in their own devices for improved reliability of services and performance;
- d) The modularity of the DFC features is maintained and all the features implemented in original DFC can be reused in the FDDP version of the DFC .

As an example, the voice mail system is designed, implemented and deployed both in the end user device and inside the network in FDDP version of DFC.

Acronyms

BCT	Blind Call Transfer
CFB	Call Forwarding on Busy
CFNA	Call Forwarding on No Answer
CPL	Call Process Language
CW	Call Waiting
DFC	Distributed Feature Composition
ECLIPSE	Extended Communications Layered on IP•Synthesis Environment
FB	Feature Box
FFB	Free Feature Box
BFB	Bounded Feature Box
FDDP	Feature Deployed at Dual Places (inside the network and at the end user device)
FSM	Finite State Machines
HTTP	HyperText Transfer Protocol
ICS	Incoming Call Screening
IETF	Internet Engineering Task Force
IP	Internet Protocol
ITU-T	International Telecommunications Union – Telecommunications Standards Sector
LI	Line Interface
OCS	Outgoing Call Protocol
PSTN	Public Switched Telephone Network

SIP	Session Initiation Protocol
SMB	Spontaneous Message on Busy
TI	Trunk Interface
UA	User Agents
UAC	User Agent Client
UAS	User Agent Server
UML	Unified Modelling Language
URL	Uniform Resource Locator
VM	Voice Mail
VMC	Voice Mail Center

Contents

Acknowledgements.....	I
Abstract.....	II
Acronyms.....	III
Contents.....	V
List of Figures.....	IX
Chapter 1 Introduction.....	1
1.1 Background.....	1
1.2 Motivation and Objective.....	3
1.3 Thesis Contribution.....	4
1.4 Thesis Organization.....	5
Chapter 2 H.323 and SIP Protocols.....	6
2.1 H.323.....	6
2.1.1 Introduction to H.323.....	6
2.1.2 H.323 Architecture: Components.....	6
2.1.3 H.323 Zone.....	8
2.1.5 Protocols Specified by H.323.....	8
2.1.6 H.323 Summery.....	11
2.2 SIP.....	12
2.2.1 Introduction to SIP.....	12
2.2.2 SIP Entities.....	13
2.2.3 SIP Messages.....	15

2.2.4 SIP Summary.....	18
2.3 General Summary.....	19
Chapter 3 Distributed Feature Composition	
Architecture.....	20
3.1 Introduction.....	20
3.2 DFC Components.....	21
3.2.1 Boxes.....	21
3.2.2 3.2.2 Router.....	22
3.2.3 Global Data.....	23
3.3 Feature Composition and Routing.....	23
3.3.1 Usage.....	23
3.3.2 Box Categories.....	24
3.3.3 Precedence Rule.....	25
3.3.4 Routing.....	26
3.3.5 Signaling/Media Separation.....	30
3.4 Limitation.....	30
3.5 Summary.....	30
Chapter 4 ECLIPSE.....	32
4.1 Introduction to ECLIPSE.....	32
4.2 Components.....	32
4.3 Routing.....	34
4.4 Data and provisioning.....	36
4.5 Signal/Media separation.....	37

4.5.1	Media representation in DFC.....	38
4.5.2	Mboxes.....	40
4.5.3	Mbox model.....	42
4.5.4	Mbox model commands.....	43
4.6	Box programming.....	45
4.6.1	Box class hierarchy.....	46
4.6.2	Port class hierarchy.....	48
4.6.3	Finite state machine class hierarchy.....	51
4.7	Monitoring and visualization.....	56
4.8	Limitations.....	57
4.9	Summary.....	57

Chapter 5 DFC with Features Deployed at Dual Places

(FDDP version DFC).....	59
5.1 Introduction.....	59
5.2 Deployment of Features in End User Device in Original DFC.....	60
5.3 FDDP version DFC.....	64
5.3.1 Components.....	64
5.3.2 Basic Scenarios.....	67
5.3.3 Example usages.....	68
5.4 Summery.....	71

Chapter 6 Development of Voice Mail System based on

FDDP version DFC.....	72
6.1 Introduction.....	72

6.2 Voice Mail System.....	72
6.2.1 Usage diagrams.....	74
6.2.2 Sequence diagrams.....	77
6.3 Implementation.....	91
6.3.1 Voice Mail Feature.....	92
6.3.2 Voice Mail Center.....	100
6.4 Modification of LI box	111
6.5 Test.....	112
6.5.1 Test for leaving message on busy.....	113
6.6 Summary.....	115
Chapter 7 Discussion: Some Important Issues in IP	
Telephony.....	116
7.1 Feature interaction.....	116
7.2 Deployment of Features.....	120
7.3 Performance and Scalability.....	122
7.4 Security.....	123
7.5 Evolvability.....	124
Chapter 8 Conclusions and Future Work.....	126
8.1 Conclusions.....	126
8.2 Future Work.....	128
Appendix A.....	130
Appendix B.....	133
References.....	141

List of Figures

Figure 2.1. Layout of components in H.323 network and connection with other networks.....	7
Figure 2.2. H.323 Protocol Stack on the terminal.....	9
Figure 2.3. An example of SIP INVITE message.....	16
Figure 2.4. An example of SIP RESPONSE message.....	18
Figure 3.1. Components in the DFC architecture.....	21
Figure 3.2. A linear usage.....	24
Figure 3.3. A nonlinear usage with a bounded feature box.....	29
Figure 4.1. A DFC usage showing components in ECLIPSE network.....	33
Figure 4.2. DFC usage on the signal layer and corresponding Mbox on the media layer.....	39
Figure 4.3. Models for the media layer in Figure 4.2.....	42
Figure 4.4. Box class hierarchy.....	46
Figure 4.5. DFC Port class hierarchy.....	48
Figure 4.6. Class hierarchy for SignalPathTermination.....	50
Figure 4.7. FSM Class Hierarchy.....	52
Figure 4.8. Simple FSM without nested FSM.....	54
Figure 4.9. A FSM with a nested FSM.....	55
Figure 5.1. Signaling processes in the signal path setup phase.....	61
Figure 5.2. Running a router and all features on the end user device.....	63
Figure 5.3. Features run on dialed zone at the end user device.....	64
Figure 5.4. FDDP version DFC architecture.....	65

Figure 5.5. Usage of VM at the end device and in the network.....	68
Figure 5.6. A usage of CFB in the end user device.....	70
Figure 5.7. A usage of ICS and CFB.....	70
Figure 6.1. Deployment of VM in end user device and inside the network.....	73
Figure 6.2. Usages for the voice mail feature and voice mail center.....	75
Figure 6.3.a. Sequence diagram for a voice mail usage:	
Leaving messages on busy (Part 1)	78
Figure 6.3.b. Sequence diagram for a voice mail usage:	
Leaving messages on busy (Part 2)	79
Figure 6.4. Sequence diagram for a voice mail usage:	
Leaving a message on no answer.....	84
Figure 6.5. Sequence diagram for a voice mail usage:	
Leaving a message on disconnect.....	88
Figure 6.6. Sequence diagram for retrieving messages.....	90
Figure 6.7. Relationship between the CallerPort, CalleePort, BoxPort and other Objects.....	92
Figure 6.8. Voice mail box finite state machine.....	93
Figure 6.9. Transparent2LinkActFSM.....	95
Figure 6.10. Open2LinksFSM.....	97
Figure 6.11. TransparentActFSM.....	100
Figure 6.12. Voice Mail Center Box Finite State Machine.....	102
Figure 6.13. Voice Mail Center Line Finite State Machine.....	103
Figure 6.14. OpenExtChanFSM.....	105

Figure 6.15. MailBoxLineFSM.....	106
Figure 6.16. MailBoxBusyFSM.....	107
Figure 6.17. AnswerFSM.....	108
Figure 6.18. MediaAnswerFSM.....	109
Figure 6. 19. LeaveRetrieveMsgFSM.....	110
Figure 6.20. Trace file for leaving message on busy.....	115
Figure 7.1. Usages using CW and CFB.....	118
Figure 7.2. Possible feature interaction between OCS and CFB.....	119

Chapter 1 Introduction

1.1 Background

Internet telephony provides real-time voice and/or multimedia communication over the Internet. It has experienced significant growth in recent years due to its low cost, highly efficient use of bandwidth and potential advanced services. However, it is far from mature. A number of new challenges are introduced by Internet transmission, such as packet loss, delay, repetition and jitter [1]. There are also common problems existing in both IP telephony and the traditional Public Switched Telecommunication Network (PSTN), such as feature (service) interaction, service composition, service deployment, and so on [2]. In this thesis, the terms “feature” and “service” are interchangeable as they are roughly synonymous, although “service” carries the connotation of being larger.

Currently there are two dominant sets of protocols standard for Internet telephony. One is the H.323 set of protocols proposed by the International Telecommunications Union - Telecommunications Standards Sector (ITU-T) and the other is Session Initiation Protocol (SIP), proposed by the Internet Engineering Task Force (IETF). Both protocols provide mechanisms for customer call establishment and teardown, customer call control and support for advanced services.

H.323 is an ITU umbrella of protocols that provides multimedia communications, such as real-time audio, video and data, over packet networks, including Internet protocol-based (IP) networks. It reuses some well-proven protocols in telecommunication, such as Q.931, H.245 and so on, instead of developing all the protocols from scratch. Therefore H.323-compliant products have a large portion of the current IP telephony market, although it is believed to be very complicated and difficult to implement. Supplementary services in H.323 are specified in the H.450.x series. Each of the services has its own specification.

SIP is used to initiate a session (call) between the users. It is a client-server protocol similar to Hypertext Transfer Protocol (HTTP) in both syntax and semantics. It is developed from scratch. New thinking, ideas and research results have been put into SIP. SIP has more advantages in terms of its flexibility when adding new advanced services, and the relative ease of implementation and debugging [3] as compared with H.323. To specify various IP telephony services, the Call Processing Language (CPL) was introduced. CPL is a scripting language. End users can specify the behavior of call agents that execute on their behalf on the SIP server when a call arrives.

Feature interaction, which refers to situations in which instances of the same features or different service features affect each other [4], has been a major problem for traditional PSTN networks. It is believed that it would be worse as more and more new features are added in the next generation telecommunication networks [5, 7]. However, neither H.323 nor SIP addresses feature interaction and service

composition problems. Distributed Feature Composition (DFC) is a virtual architecture for the description of telecommunication services. It was designed for feature modularity, structured feature composition, analysis of feature interactions, and separation of service and transmission layers [2, 8, 9,10]. A DFC feature is designed as a module and is independent of other DFC features. To avoid bad feature interactions and preserve good feature interactions, a precedence rule is used as the only constraint to guide the feature composition. In research done prior to this thesis, the features have been deployed on routers inside the DFC network.

1.2 Motivation and Objective

In the long term, the greatest advantage of Internet telephony will be the enabling of a large number of advanced services [11, 12]. As more and more features are invented and added, the management of these features, to avoid the bad feature interactions and preserve the good ones, will become crucial for the success of IP telephony technology. In recent years, the astonishing advances in the processor power of the end user device have made it possible to run features in the end user device. This is reflected in the recent trends towards the deployment of IP telephony features in the end user devices [12-14].

In term of service management, DFC is one step ahead of the other two protocols. The centralized deployment of features in the DFC causes performance problems when the number of features running on the network is large [15], and also wastes the processing power of the end user devices. Where should a feature reside

in IP telephony? There are three possible ways to deploy a feature. One is to deploy the feature inside the network; one is to deploy the feature in the end user device; and another is to deploy the feature both inside the network and in the end user device. An examination of the deployment of features in both the end user device and inside the network in the DFC architecture is the main motivation of the thesis. The objectives of the thesis are two-fold:

- 1) To study the possibility of deploying features in the end user device in the DFC architecture; and
- 2) To modify the current architecture to allow the end user to load telephony features from the central server; to run features which originally resided inside the network on the end user device, without any change; and to allow users to subscribe to features on their own devices and on the network. This last will help to reduce the workload on the network, improve the overall performance of the system and keep the modularity of the DFC features.

1.3 Thesis Contribution

In this thesis, a new DFC architecture with features deployed in dual places (FDDP version DFC) is created with addition of a local feature manager, a blind call transfer feature (BCT) and a proxy of the line interface box for a user, to the original DFC architecture. The local feature manager resides in the end user device. The BCT and the proxy reside on the router inside the network. The new architecture provides significant flexibility allowing features to run at the end user device, inside the

network or both depending the nature of the features. Voice mail feature is also designed and implemented.

1.4 Thesis Organization

The remainder of the thesis is organized as follows. The second chapter briefly overviews the currently predominant IP telephony protocols, H.323 and SIP, to give some sense on service (feature) deployment and service (feature) management. The third chapter will give a detailed illustration of the DFC, including feature construction, feature management, routing protocols, media management etc. The fourth chapter will give a detailed review of the Extended Communications Layered on IP•Synthesis Environment (ECLIPSE) project. In the fifth chapter, the original DFC architecture is modified to allow features running in the end user device by addition of a local feature manager, a BCT feature and a proxy of user's line interface box software components without sacrificing the merits of the original DFC. Based on the new architecture, the deployment of features in the end user device or on the network is discussed. In the sixth chapter, a voice mail feature and voice mail center are designed and implemented, based on the new DFC architecture. Detailed design diagrams are presented. In the seventh chapter, some important issues related to the IP telephony architecture, especially service architecture, are discussed. Finally the eighth chapter presents the conclusions of the thesis and provides suggestions for future work.

Chapter 2 H.323 and SIP Protocols

2.1 H.323

2.1.1 Introduction to H.323

H.323 is an umbrella specification which describes the architecture and operations for real-time voice, video, and data communication over packet-based networks, including the Internet. The original version of H.323 (H.323v.1) was ratified by the Study Group 16 of the Telecommunications Sector of the International Telecommunication Union (ITU-T), as “Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service” [16]. Since then, H.323 has been evolving with the addition of more protocols and more services. The current version is H.323 version 3. H.323 version 4 is under development. The H.323 protocol stack is designed to operate above the transport layer of the network. Therefore H.323 can be used on top of any packet-based network transport, like the Internet, ATM and Frame Relay, to provide real-time multimedia communications.

2.1.2 H.323 Architecture: Components

The basic components defined by H.323 are terminals (T), gateways (GW), multipoint control units (MCUs) and gatekeeper (GK) as shown in Figure 2.1.

Figure 2.1. Layout of components in H.323 network and connection with other networks

Terminal

A terminal is an end user device which runs an H.323 stack and supports voice and/or other real-time multimedia applications. It can be either a personal computer (PC), or an H.323-enabled IP telephone. Its main function is to communicate with other terminals, including both point-to-point calls and multipoint conferences.

Gateway

A gateway is used to connect an H.323 network to the other networks, such as PSTN. It translates protocols for call setup and release, and converts media formats between different networks.

Multipoint Control Unit

An MCU provides a multipoint conference service. It consists of two logical components: a multipoint controller (MC) and a multipoint processor (MP). The MC controls the conference resources, determines the common audio and/or video coder/decoder (CODEC) available to all the participants, and carries out other coordination activities. The MP is used in audio and video stream mixing.

Gatekeeper

A gatekeeper is thought to be the only intelligent part in the H.323 network. It has the following functions: address resolution, bandwidth management, admission and access control of endpoints, billing and possibly routing calls. It is optional in the H.323, but if it is present in the network, the terminals and gateways must use its services.

2.1.3 H.323 Zone

A zone consists of all terminals, gateways and MCUs managed by a single gatekeeper as shown in Figure 2.1.

2.1.4 Protocols Specified by H.323

The protocols under the H.323 umbrella are listed here. For audio CODECs, G.711 is mandatory, and G.729 and G.723 are optional. For video CODECs, H.261 is mandatory, and H.263 is optional. H.225 is for registration, admission, and status (RAS), and call signaling. H.245 is for control signaling. The H.245.x series is for supplementary services. Real-time transfer protocol (RTP) and real-time control protocol (RTCP) are

also under the umbrella. The H.323 protocol stack running on the terminal is shown in Figure 2.2.

Figure 2.2. H.323 Protocol Stack on the terminal.

Audio CODEC

An audio CODEC encodes the audio signal before the terminal transmits it and decodes the audio code after the terminal receives it. G.711 is required and others are optional.

Video CODEC

A video CODEC encodes video before the terminal transmits it and decodes the video code after the terminal receives it. As supporting video in H.323 is optional, these video CODEC protocols are optional.

H.225

H.225 is used for registration, admission, and status management (RAS) between endpoints (terminals and gateways) and gatekeepers. The following procedures are involved: gatekeeper discovery (GRQ), endpoint registration, endpoint location and admission control. The gatekeeper discovery process is a process in which the endpoints search for a proper gatekeeper to register. Endpoint registration is a process in which the endpoints join a zone and register themselves to the gatekeeper. Endpoint location is a process in which the address of an endpoint is determined and its alias name is given. Due to network conditions such as traffic, bandwidth, and server CPU work load, a terminal may or may not be allowed to enter into a zone, and a terminal may even be disengaged from a gatekeeper and its zone if the traffic is too high. All these jobs are carried out in the admission control.

H.225 is also used as a call signaling protocol. It is a subset of Q931 and is used to connect two H.323 endpoints over a reliable call-signaling channel. For example, H.225 protocol messages are carried over TCP in the Internet.

H.245 Control Signaling

H.245 control signaling exchanges H.245 messages between the endpoints. The messages include terminal capabilities, flow rate control messages and messages to open and close logical channels. The endpoints adjust their operations according to the messages.

T.120

T.120 is a protocol stack based on a layered architecture. The lower layers of the stack provide common transport functionality to data conferencing applications defined in the top layer.

Real-Time Transport Protocol

Real-time transport protocol (RTP) delivers real-time audio and video from endpoint to endpoint. In the IP network, the RTP transports data using the user datagram protocol (UDP). RTP adds the payload-type, sequence number and time stamp to the headers of the packets, and examines these fields. The UDP multiplexes the packets and makes a checksum of the received audio and video.

Real-Time Transport Control Protocol

Real-time transport control protocol (RTCP), as its name implies, controls the real-time transport of audio and video multimedia streams. RTCP provides feedback on the quality of the data received and also carries a transport-level identifier for the media source transported by RTP. Based on the feedback, the source can adjust its operations. With the source identifier, the receiver synchronizes audio and video.

2.1.5 H.323 Summery

H.323 reuses quite a number of the well-proven protocols used in telecommunication networks, which makes it relatively easy to bridge traditional telephone networks with media rich packet-based networks. There is a great potential for

new services and applications that take advantage of the capabilities of both networks. These services can range from value-added traditional telephone services, such as call transfer, and diversion to new services such as integrated messaging (e-mail, voice mail, fax, instant messaging, etc) [1,11].

In terms of services, H.323 specifies supplementary services through the H.450.x series protocols, leaving the service architecture open to the IP telephony system developers. H.450.x does not address how to implement features, how to deploy features or how to manage feature (service) interactions. H.450 adopts a hierarchical structure for the development of new services. A general framework for supplementary services is defined in H.450.1. H.450.1 provides an essential mechanism for end-to-end control signaling between peer service entities. ITU-T specifies Call Transfer in H.450.2, Call Diversion in H.450.3, Call Hold in H.450.4, Call Park/Pickup in H.450.5, Call Waiting in H.450.6, Message Waiting in H.450.7, Name Identification in H.450.8, and Call Completion on Busy Subscriber in H.450.9.

H.323 is a large, complex, and flexible standard. The complexity and the flexibility of H.323 make its implementation difficult and prone to errors [17,18]. It may also cause interoperable problems in the communication between the IP telephony products from different IP telephony vendors.

2.2 SIP

2.2.1 Introduction to SIP

The Session Initiation Protocol (SIP) is a signaling protocol for initiating, managing and terminating voice and video sessions across packet networks. It is being developed by the SIP Working Group, within the Internet Engineering Task Force (IETF), and published as RFC 2543 [18].

SIP is a client–server protocol similar to Hypertext Transfer Protocol (HTTP), both in syntax and semantics [18,19]. SIP users are identified by email-like SIP URLs: user@host. The user portion of the SIP URL can be a user name or a telephone number, and the host portion can be a domain name or numeric network address. An SIP transaction consists of a request from a client and one or more responses returned by one or more servers. Each SIP entity can be a client and also a server depending on whether the entity issues a Request or receives a Request.

The SIP protocol can use both TCP protocol and UDP protocol to send its signals. UDP provides faster and better scalability than TCP. However, SIP protocol has to provide its own mechanism for reliability if UDP protocol is used for signaling. As the SIP protocol is an application level protocol. It does not depend on the underlying transport layer and network layer protocols. Therefore SIP can not only be used for the Internet, but also ATM, frame relay, and so on [18,20].

2.2.2 SIP Entities

An SIP network is composed of four types of logical SIP entities: User Agent (UA), Proxy Server (PS), Redirect Server (RS) and Registrar.

User Agent

A user agent is an endpoint entity. User agents initiate and terminate sessions by exchanging requests and responses. RFC 2543 defines the user agent as an application, which contains both a user agent client (UAC) and a user agent server (UAS). A UAC is a client application that initiates SIP requests. A UAS is a server application that contacts the user when an SIP request is received and that returns a response on behalf of the user.

Proxy Server

A proxy server receives requests from callers or other servers and forwards them to another server (next hop server), which has more precise location information about the callee. The next-hop server might be another proxy server, a UAS, or a redirect server. The primary function of proxy is call routing, i.e. the determination of the set of servers to traverse in order to complete the call.

An SIP proxy server can also fork a request, sending copies to multiple next-hop servers simultaneously. This allows a call set-up request to try many different locations. The first location to answer is connected with the calling part. This can be viewed as some kind of “find” feature in the other telecommunication networks.

Redirect Server

A redirect server also receives requests, and determines a next-hop server. However, instead of forwarding the request there, it returns the address of the next-hop server to the client.

Registrar

A registrar is a server that accepts REGISTER requests for the purpose of updating a location database with the contact information of the user specified in the request.

2.2.3 SIP Messages

There are two types of SIP messages: Requests sent from the client to the server, and Responses sent from the server to the client. The SIP protocol adopts most of the syntax and semantics of the HTTP protocol. Every SIP message is composed of three parts: start-line, headers and message body. There is a blank line separating the headers from the message body.

Request Messages

As in HTTP, the client requests invoke methods on the server. The SIP protocol defines six methods for its request messages. These methods allow clients to request the server to act according to the request. The six methods are INVITE, ACK, OPTIONS, BYE, CANCEL and REGISTER. INVITE initiates a call. ACK confirms a final response for INVITE. BYE terminates a call. OPTIONS queries the other side's capabilities.

CANCEL is used to cancel a pending request. REGISTER is used to register the client's current location with the Registrar.

A client sets up a call by issuing an INVITE request. A typical SIP INVITE message is shown in Figure 2.3. The start-line of the message is a request line that includes a method token, INVITE, an SIP URL or a general URL, and a protocol version. The header fields in the request provide information about the call.

INVITE sip:rli@sce.carletonca.ca SIP/2.0

Via: SIP/2.0/UDP here.com: 5060

From: Joe <sip:Joe@company.com>

To: Ruiguo Li <sip:rli@sce.carleton.ca>

Call-ID: 12345@company.com

CSeq: 1 INVITE

Subject: Hello

Contact: Joe<sip:Joe@company.com>

Content-Type: application/sdp

Content-Length: 888

v=0

o=user1 53655765 2353687637 IN IP4 128.3.4.5

c=IN IP4 224.2.0.1/127

t=0 0

m=audio 3456 RTP/AVP 0

Figure 2.3. An example of an SIP INVITE message.

The Via field traces the path of the request in order to allow intermediary SIP servers to forward the replies along the same path. The To and From fields contain the callee's and caller's address, respectively. The Call-ID header field contains a unique call identifier. The Cseq header field contains a request method and a single decimal sequence number that is unique to one Call-ID. The Subject field provides a summary of a customer call. The Contact field provides a URL where the user can be reached. The Content-Type field indicates the media type of the message body. The Content-Length shows the size of the message body.

Response Messages

An SIP server replies to an SIP request with one or more SIP responses. There are six classes of status codes and two types of responses defined for the response message. The six classes are:

- 1xx: Informational (request received, continuing to process the request).
- 2xx: Success (the request was successfully received, understood, accepted and the action was done successfully).
- 3xx: Redirection (further action needs to be taken in order to complete the request).
- 4xx: Client Error (the request contains bad syntax or cannot be fulfilled at this server).
- 5xx: Server Error (the server failed to fulfill an valid request).
- 6xx: Global Failure (the request cannot be fulfilled at any server).

The two types of responses are provisional (1xx) and final (2xx, 3xx, 4xx, 5xx and 6xx). Provisional responses are used by the server to indicate progress, but they do not

terminate SIP transactions. Final responses terminate SIP transactions. An example of a response message is shown in Figure 2.4 below. An SIP response message has a response status code and a reason phrase. The status code is intended to be read by the machine, and the reason phrase is intended to be read by the human user. The first line is the status line. It consists of a protocol version, a numeric status code, and its associated textual phrase. Other fields in the response message are similar to those in the request message.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP sce.carleton.ca: 5060
Via: SIP/2.0/UDP company.com: 5060
From: Joe <sip:Joe@company.com>
To: Ruiguo Li <sip:rli@sce.carleton.ca>;tag=113448
Call-ID: 12345@company.com
CSeq: 1 INVITE
Content-Type: application/sdp
Content-Length: 888
```

Figure 2.4. An example of a SIP RESPONSE message.

2.2.4. SIP Summary

SIP is an application level protocol defined in RFC2543 for establishing multimedia communications. It is independent from the underlying transport layer and network layer protocols. It can interoperate with ISDN and other networks through gateways. It is developed from scratch. Therefore it is free of extra, seldom-needed

software components or protocols, while H.323 carries many extra protocols and/or software components to make sure that it is interoperable with the other standards in the series. SIP is a much smaller and less complicated standard that is based on the architecture of existing popular protocols such as HTTP. SIP uses a simple format for commands and messages. Therefore it is easy to debug. Extending the ability of SIP is also easier because of its simpler HTTP-like message format [21].

2.3 General Summary

Currently H.323 and SIP are the predominant protocols in the IP telephony. However, neither of them has clearly addressed feature interaction, deployment of features and feature composition. These problems are expected to be more serious in the next generation telecommunication networks as more and more advanced features are added.

Chapter 3 Distributed Feature Composition Architecture

3.1 Introduction

Feature interaction problems will become worse as more and more advanced services (features) are invented and added to the IP telephony. Powerful intelligent end user devices, multimedia transmission and mobility have the potential to create enormous services which are not available today [1, 6]. It is expected that the feature interaction problem will be the major thread to the reliability and evolvability of the IP telephony system. To succeed, the IP telephone architecture has to have a strategy to manage feature interactions, to allow good feature interaction and prevent bad feature interaction. Distributed Feature Composition (DFC) is a virtual architecture for the description of telecommunication services developed by AT&T. It was designed for feature modularity, structured feature composition, analysis of feature interactions, and separation of the service and transmission layers [8-10]. It is AT&T's solution to the feature interaction problem.

3.2 DFC Components

Figure 3.1 below shows the components in the DFC architecture. For simplicity, the multimedia components have not been included. Those components will be discussed later, in chapter 4. Each component in Figure 3.1 will be discussed here.

Figure 3.1. Components in the DFC architecture [23].

3.2.1 Boxes

A box (shown in Figure 3.1 as a square with a single border) is a call-processing component in the DFC system. It can be a line interface (LI) box, a trunk interface (TI) box or feature box. It also includes media related boxes, such as a resource interface (RI) box and a media switch (Mbox), which will be discussed in chapter 4.

A line interface box is an interface to a telephone line. A trunk interface box is an interface to a trunk to connect to the other telephone systems, such as PSTN.

A feature box is an implementation of a feature. Each feature is a largely independent module. The feature boxes (F) can have any number of ports, depending on their various functions. A feature box has full control of all the call signals it sends or receives and processes the signals received without external assistance. When a feature box does not need to function, it can behave transparently, i.e., the feature box receives the signal and passes it on to the next box without processing the signal. When an action-triggering signal is received, a feature box can absorb and process the signal, generate new signal (s), re-route calls and process media streams. Each box in the DFC has one or more ports, shown in Figure 3.1 as black dots. These ports are used to send and receive featureless calls. These featureless calls are called internal calls in the DFC. The internal call connects the two ports on two different boxes. Internal calls are provided by the port-to-port virtual network.

3.2.2 Router

The router of the virtual network in the DFC is different from the normal router in the network. It not only routes internal calls to the destinations associated with their target addresses, as any network router does, but it also assemble features by routing internal calls to and from feature boxes. All the boxes including LIboxes, TIboxes and feature boxes run on the router.

3.2.3 Global Data

Assembling the connection between the caller's LI box and the callee's LI box requires data on feature subscriptions, feature precedence and the dialing plan, as well as configuration data. This is all global data, as indicated by the double rectangles in Figure 3.1. Figure 3.1 also shows global data called operational data, which is used by the feature boxes. For example, the Call Forwarding on Busy feature box retrieves its subscriber's forwarding address from the operational data. Each feature that the user subscribes to can only access its own operational data as preset by the user. This strict partition of the operational data by features keeps feature modularity.

3.3 Feature Composition and Routing

3.3.1 Usage

In DFC, a customer call creates and is responded to by a usage in which boxes and internal calls are dynamically assembled. A usage is analogous to an assembly of pipes and filters in the Pipe and Filter Pattern [22], with pipes being internal calls and filters being feature boxes. The dynamic assembly of boxes and internal calls in a usage has great advantages, including the fact that feature boxes are largely independent and can be composed freely in many different combinations; the set of features is easily extended; and the interactions between features can be analyzed with relative ease.

As a detailed demonstration of how a call is realized in the DFC, one linear usage is illustrated in Figure 3.2 and one non-linear usage in Figure 3.3.

Figure 3.2. A linear usage

3.3.2 Box Categories

In the DFC architecture, boxes are categorized into two large types: bound boxes and free boxes. A bound box is a unique, persistent, addressable individual. Bound boxes are shown with double borders in Figures 3.2. The advantage of bound boxes is that they make it possible to have joins in usage graphs. A free box is an anonymous, interchangeable copy of its type with no persistence outside its tenure in a usage. The free box is shown with a single border in Figure 3.2.

In Figure 3.2, DFC internal calls are expressed as consecutive arrows from the port of line interface box A that placed the call, to the port of line interface box B that received the call, connecting all the feature boxes between them. In each call, there are three logical zones: source, dialed and target, where the features are assembled. They come from the source zone via the dialed zone and to the target zone, along the usage. These zones can be physically distributed or on the same physical device. In Figure 3.2,

user A subscribes to a Spontaneous Message on Busy (SMB) feature at the source zone and Credit Card Call at the dialed zone, and user B subscribes to a Voice Mail (VM) feature and a Call Forward on Busy (CFB) feature at the target zone. When the callee is busy, the SMB will call the callee periodically until the connection between the caller and callee is established or connection is aborted due to time-out. The purpose of this section is to introduce the basic concepts of DFC. The specific feature interaction case will be discussed in the Chapter 7.

3.3.3 Precedence Rule

The order of the features in each usage is governed by the precedence rule. It is the only constraint in the DFC where the features are explicitly related to one another. It sets the priority of each feature. Features whose triggering signals travel downstream (from the source zone, via the dialed zone, to the target zone) should have the highest priority at the source, lowest priority at the target zone and middle priority at the dialed zone. Features whose triggering signal travels upstream should have the priority in the opposite order. The feature that has higher priority reacts to the triggering signal earlier. For example, a busy signal usually originates in the target line interface. So the proper coordination of busy signal treatment features is achieved by placing the higher-priority feature boxes later in the route, i.e. closer to the source of their triggering signal. A busy treatment feature absorbs and responds to a busy signal if its feature is enabled, or passes the signal on to the earlier feature boxes in the route if it is not enabled.

3.3.4 Routing

An internal call starts when the caller port of line interface box A sends a setup message to the DFC router. After the router receives the setup message, the router will return an acknowledge message to the sender. The router will compute a route based on the DFC routing algorithm. The setup message has five fields of interest to the router. They are: source: address, dialed: string, target: address, command: {new, continue, update}, and route: sequence of (seq) routing_pair. Each routing pair has a first component of type box_type and a second component of type zone = {source, dialed, target}. How to route a call will be explained by example. The setup message sent by LI A has a source field containing userA, a dialed field containing the dialed string, and a command field containing new. The other two fields are empty. Upon receiving the signal, the router first extracts the target line address userB from the dialed string, and put it into the target field. Next, the router, instructed by command new, computes a new route and puts it into the route field. User A subscribes to one feature (SMB) in the source zone, so the first pair of the route is (SMB, source), which has lowest priority. Since Credit Card Call needs to authenticate the user, which inevitably involves the database operations, CCC is better deployed somewhere on the network where the database server runs. The dialed string matches the triggering pattern of one feature CCC in the dialed zone (for example, 0xxxxxxx, first 0 indicates the call is a Credit Card Call), so the next pair of the route is (CCC, dialed). In the target zone, user B subscribes to two features, VM and CFB. Both of these features are busy signal treatment features. The order in which these two features react to the busy signal becomes very important. If VM is deployed closer to the target line interface box than CFB, VM will react to the busy

signal earlier than CFB. VM will ask user A to leave a message in the user's mailbox. Most of the time, the mailbox is not busy. The VM will block the CFB feature, causing bad feature interaction. If CFB is deployed closer to the target line interface box, CFB will react to the busy signal earlier than VM. Upon receiving the busy signal, CFB will forward to LI box C. If LI box C is free, a call will be connected from user A to user C. If LI box C is busy, then CFB will receive a busy signal. Therefore CFB is assigned a higher priority. The last two pairs of the route are (VM, target) and (CFB, target). Now a route has been built. The router has to find a box to which it can route the internal call. It strips the first pair off the route, and since SMB is a type of free box, it routes the internal call to an arbitrary fresh box of that type. The feature boxes in the upper part of Figure 3.2 do not need to control the routing initially. So when each box prepares a setup message for an outgoing call, it simply copies the entire setup message from its incoming call, and changes the command field from new to continue. The continue command tells the router not to re-compute anything in the route. As the chain unfolds, one pair of the route is deleted and each free box is added to the usage. Finally, in the last internal call, the route is empty so the router routes to the bound box LI B.

After the setup message reaches the LI B, if the user is already engaged in another call, the LI userB will send back a busy signal. Upon receiving a busy signal, the CFB springs into action. CFB retrieves the forwarding address (userC) from the operational data provided by the subscriber and puts it in the target field, changes the command to Update in the Setup message and then sends it to the router. The router computes the route to user C. The router removes from the route the remnants of the target zone of b,

and replaces them with a newly computed target-zone route for user C. If the connection with user C is successful, then the busy signal is absorbed by CFB. If user C is busy too, then the CFB will not absorb the busy signal and the CFB will forward it to the VM. Then VM will ask the caller to leave a message in user B's mail box. If due to some reason the VM is also busy, then the VM has to pass the busy signal to CCC. As a busy signal is not the triggering signal for CCC, CCC simply passes the busy signal on to SMB and behaves transparently. Upon receiving the busy signal, the SMB will send the Setup message to the router again and again with the time interval specified in the operational data until a successful connection is made or the caller aborts the call.

From the call setup phase until the call teardown phase, the call has a two-way signaling channel. The call can also have any number of media channels, each carrying any medium. Each media channel in a call must be opened and closed explicitly by signals on the signaling channel.

Figure 3.3 shows a non-linear usage with a bound feature box for Call Waiting. User C subscribes to CW and an arbitrary feature, F, in both his source and target zones. The usage is formed based on the following scenario. First user C makes a successful call to user A. CW behaves transparently after the setup phase. Later, user B tries to call user C. In the set-up phase, when the router computes the route to user C, the router finds that CW is a bound feature box and a CW instance already exists, so the router routes the call to CW on its way to user C, where the call is accepted at the third port. Acceptance of a call at its third port causes CW to spring into action. It first signals back to the line

interface box for user b (LIb) and the state of the call moves into the alert state; CW then alerts the line interface box for user C (LIc) by sending a text message to the LIc in the current implementation of the DFC, which will be discussed in detail in chapter 4. CW then monitors the signal channel from LIc for flash messages. Each time CW receives a flash message, it switches the voice channel between user C and user A or user C and user B.

Figure 3.3. A nonlinear usage with a bounded feature box.

In Figure 3.3, user C subscribes to CW in both the source and target zones, to make sure that the customer calls from user C and to user C go through the same box, because the CW box is involved in two calls: in one call, user C is the caller and in the other call, user C is the callee. In Figure 3.3, the customer call from user C to user A goes

through CW because user C subscribes to CW in the source zone. The customer call from user B to user C goes through CW because user C subscribes to CW in the target zone.

3.3.5 Signaling/Media Separation

DFC separates signaling path from media path as some other modern telecommunication systems, such as ISDN, do. A media path is chosen as the shortest path between the two end points. The media part of the Extended Communications Layered on IP•Synthesis Environment (ECLIPSE) will be discussed in more detail in the next chapter.

3.4 Limitation

The modularized features, pipeline fashion assembly of features and precedence rule in DFC makes the analysis of feature interaction much easier. However, human experts who have significant knowledge of all the features still have to consider all the scenarios about all the possible involved features to assign each feature a unique priority.

3.5 Summary

In DFC, features communicate with each other using only featureless internal calls that guarantees features are independent of each other. The feature box must not change when its environment changes; for example, if new features are added. All the signals concerning the signal and media channels go through each feature box in the usage, so each feature box can process its triggering signals if the box receives the signals. The precedence rule is the only constraint to guiding the feature composition that

makes feature interaction analysis much easier and also makes the system more extensible and evolvable.

Within the IP community, the SIP protocol and its accompanying CPL language can be regarded as defining an architecture within which services are created [7]. But the services accommodated are largely confined to the initiation phase of user communication. H.323 only has services protocols.

Compared with the above alternatives, DFC is significantly more general, modular, extensible and evolvable.

However, the modularity of the system and pipeline fashion assembly of features has brought in a fair amount of overhead, which contributes to the deterioration of the performance of the system. Another defect of the DFC is that all features are deployed inside the network, increasing the system's workload and wasting the intelligent end devices' significant processing power.

Chapter 4 ECLIPSE

4.1 Introduction to ECLIPSE

The Extended Communication Layered on IP•Synthesis Environment (ECLIPSE) project is based on the DFC architecture. “The primary goal of the ECLIPSE project is to determine whether the DFC can serve as the service architecture of next-generation telecommunication networks” [23, 24]. To be a successful service architecture, it must accommodate various kinds of features, including the value added features of traditional networks and new IP telephony features, and manage feature interactions well. DFC was designed for abstraction and modularity.

In ECLIPSE, the signaling path and media path have been separated. Bandwidth shortage is the major problem for many multimedia Internet applications, therefore media transmission should follow the shortest possible path. Signaling is less expensive, but reliability is more important. Java RMI is used for the signaling in DFC.

4.2 Components

The ECLIPSE implementation of DFC has two major subsystems [25]. One subsystem provides routing, signaling, control, and global data; it is a straightforward

implementation of DFC with a slight modification. The architecture in ECLIPSE is more distributed than pure DFC because it introduces the concept of location [24]. The signaling subsystem is layered on top of the media subsystem. The media subsystem transmits, receives and processes media. Since transmission has to follow the shortest path, the media path can be different from the signaling path. The interface between the signaling layer and media layer is designed to be stable. Commands and messages specified in the interface will be discussed later in this chapter.

Figure 4.1 A DFC usage showing components in ECLIPSE network.

Figure 4.1 shows all the components in the ECLIPSE network. The DFC usage is on the signaling layer of the ECLIPSE network and controls its media layer. The strong dotted line separates the software components. F1 to F6 are IP telephony features. L1a is a line interface box. TI is a trunk interface box. MB is a media switch box where media are processed. ECLIPSE routers are completely different from IP routers, because they are application programs that route the internal calls to the target address and assemble the feature boxes for each usage. They have no direct involvement with any lower layer activities. They use IP routers to transport the packets in the IP network. Provisioning manager processes provisioning requests from each entity in the ECLIPSE network, and administrates the whole system and updates the central data server accordingly.

4.3 Routing

Each router in the ECLIPSE network is similar to the way in which the local office in the PSTN runs independently. It only supports a subset of all ECLIPSE network customers and only holds data of that subset. All routers work together to compute the logical routes and route the internal calls between the feature boxes.

In ECLIPSE, a lazy evaluation is used in the computation of logical routes for optimal solution. Here is a typical lazy evaluation situation in which a source customer on one router calls a target customer whose data resides at a different router. The router at the source zone can compute the route at the source zone of features from the local subscription data responding to a new command. Then the features at

the source zone are connected. Then a Setup message goes to the router at the target zone via a dialed zone if either the caller or the callee subscribes to features on the dialed zone. The router at the target zone computes the rest of the route for the features in the target zone, using only the local subscription data at the target zone.

In ECLIPSE, the feature boxes are created and deleted in response to the start and end of a call by a box factory and a clean up function in the router. The code for each of feature boxes is stored in the central server. It is dynamically loaded to the routers when it is first needed and stays there afterwards. An instance of the feature box is constructed from the loaded code using Java Reflection. Each feature box runs as an independent thread and communicates with each other using port-to-port signaling through unbounded queues.

As mentioned in chapter 3, feature boxes can be categorized as free boxes or bound boxes. The box factory in the router treats free and bound boxes differently. For free boxes, a new instance is created when a setup signal is received. For bound boxes, a check is made to see whether a box is already instantiated; if so, that box is sent the relevant setup signal; if not, a new box is created.

Line-interface and trunk-interface boxes are persistent in ECLIPSE.

4.4 Data and provisioning

Initially all the data is stored in the central data server in the provisioning manager. Part of the data comes from the registration data submitted by the router when the router instances are created. Part of the data comes from the registration data submitted by the line interface boxes and trunk interface boxes. The customer's feature subscription data is another source of data, when the user subscribes to features through the web server. Other data, such as precedence and dialing pattern, is the global data which has existed since the provisioning manager began. Part of the data is loaded to the router when the router needs the data. Here the data stored is in the router.

Subscription data

Subscription data is implemented as a hash table that stores the information of the customer, the features subscribed to which the customer subscribes, and the feature operational data. Each router only stores subscription data for the customers associated with the router. The provisioning manager stores the subscription data for all customers.

Precedence

The precedence of a feature is implemented as an integer. Its value depends on the characteristics of the feature and its relationship with the other features. The precedence data for a series of features is global data stored in both the router and the provisioning manager, but independent from the router.

Dialing patterns

Dialing patterns are the few patterns customers use when they are dialing for special services. For example, if we use 1-800 number service in the dialed zone, we need to dial 1800xxxxxxx. Dial patterns are also stored in both the router and the provisioning manager.

Configuration data

Configuration data is the information about the association of line interface boxes and trunk interface boxes with a specific router. There is configuration data for the provisioning manager too. It stores all the information about the routers and line and trunk interface boxes.

4.5 Signal/Media separation

In ECLIPSE, the signal channel and media channel are separated. Signal channel implementation is based on the DFC, with some modification. Media are mainly introduced in the ECLIPSE project. Principally speaking, a DFC system can support any number of media. However, the ECLIPSE media implementation is designed to support a relatively modest number of media, less than ten. Users of the most commonly used media, such as voice, video, text, images and audio (high-quality sound), are interested in ECLIPSE. However this does not set a limit on the number of media types which can be used; when necessary, other media can be added through the current well-defined interfaces.

The implementation of each medium is separate. All media types are treated in the same way in the signaling layer in the DFC network with regard to opening and closing channels and routing the media stream. The only place to produce, interpret, split, merge or translate multiple media in a DFC system is in an interface box or a device hiding behind an interface box.

4.5.1 Media representation in DFC

In a DFC, a call can have any number of bi-directional channels, each of which carries a medium. Each channel has an identifier and two channel terminations. A channel termination is composed of a port and a channel-identifier. Each interface box not only has the normal DFC ports, which are now referred to as internal ports, but also has at least one external port. An external port is the place where an external line, external trunk, or resource joins the interface box. External ports enable us to specify media processing capacity within interface boxes. An external channel is a medium channel between an external port and a device, resource, or separate network. An external channel has only one fully identified channel termination; its other termination (at a device, resource, or separate network) has only a channel identifier.

The media channel representation in a box is formalized as a set of links. Each link is a unidirectional media connection between two channel terminations; the channel terminations must have distinct ports on the same box.

To distinguish components in the signal layer from those of the media layer with similar names and functions, the components in the signaling layer will be prefixed with a capital D, signifying that is from the DFC, such as Dboxes, Dports, Dlinks, Dchannels, and Dcalls.

Figure 4.2. DFC usage on the signal layer and corresponding Mbox on the medial layer. The same devices are shown in both layers [10].

Figure 4.2 above shows the Dchannels, Dlinks and Dcalls in a DFC usage. A Dchannel, shown as a dashed line, follows a call or external line and connects two boxes or one line interface box and an external device, labeled with a channel

identifier. A Dlink is shown as a dashed arrow within a box. A Dcall is shown as a solid line between the Dboxes.

4.5.2. Mboxes

In the media layer, the media switch is encapsulated in a software component called an Mbox. The ports of the media switch are called Mports, and the bidirectional media connections between Mports of different Mboxes are named Mcalls. When a Dbox designed to use a certain medium is created, it is given the address of an Mbox of that medium. The assigned Mbox implements all processing functions for that medium specified in the interface between Mbox and Dbox.

The Mbox assignment does not change during the lifetime of the Dbox. The dotted lines in Figure 4.2 show the signaling connections between the Dboxes and the Mboxes assigned to do their media processing. Besides a media switch, an Mbox also includes a controller and a model. The controller receives and responds to commands from Dboxes. The model is a representation of the current media representation of all the Dboxes to which this Mbox has been assigned. The model contains Dchannels and Dlinks, which have been created by the controller in response to Dbox commands, and the status of the Dchannels and Dlinks. In addition to maintaining its model, a controller performs the following three functions: creates and destroys Mcalls; controls media transport; and controls time division multiplexing of media streams from different customer calls. The following paragraphs will provide further details on these functions.

In the usage shown in Figure 4.2, the user b subscribes to a Call Forward on Busy (CFB) feature, to which MBox1 is assigned, in the target zone, and user c subscribes to an F feature, to which MBox2 is assigned, in the target zone. If user b is busy, then the CFB forwards the call to user c. Here a Dchannel connects the two Dboxes with different Mboxes, the Dchannel needs to appear in the models of both Mboxes, and its implementation requires an Mcall between the two Mboxes. The controllers of the two Mboxes co-operate in creating and destroying the Mcalls required by their models. For example, in Figure 4.2 the Dchannel between (g,7) and (h,7) is created using an Mcall between Mport p of MBox1 and Mport q of MBox2.

To maintain a smooth, stable media flow through the Mbox, the media output at any Mport of the Mbox should be equal to the sum of the inputs from the remaining Mports of the same Mbox. The controller constantly computes the input and output of all Mports from the Mbox's model and adjusts the media switch to ensure that each Mport's output matches the computed result. For example, the output from Mport p should be equal to the sum of the input from Mport n and input from Mport o.

The creation of external Dchannels requires Mcalls; for example, the Mcalls from Mports *n*, *o*, and *r* in Figure 4.2. For DFC and the model, external Dchannels are opened at provisioning time and persist until the customer call is torn down. However, in the real implementation, the Mcalls creating them are intermittent since there are many calls going through the same Mbox most of the time and each

occupies its own time slot. The controller gathers and stores all the necessary information about every external Dchannel at provisioning time, and maintains a corresponding Mcall.

4.5.3 Mbox model

A model in an Mbox is shown by a dynamic graph. Figure 4.3 is an expansion of the media part of Figure 4.2, showing the models within the Mboxes.

Figure 4.3. Models for the media layer in Figure 4.2 [10].

The nodes in the graph represent Dchannel terminations. A node on the border of the model has an Mport, while a node inside the model has not. There are two types of edge: a plain line refers to a Dchannel, while an arrow refers to a Dlink. Depending on the assignment of Mboxes to Dboxes, there are two types of Dchannels. The first type is an internal Dchannel connecting two Dboxes, both of which have been assigned to the same Mbox. The second type is a cross-border

Dchannel connecting two Dboxes, each of which has been assigned to a different Mbox. For example, the Dchannel between (d,5) and (e,5) is an internal Dchannel residing in the Mbox's model. Neither of its terminations lies on the border of the model. The Dchannel between (g,7) and (h,7) is a cross-border Dchannel, existing in the models of both Mboxes. In each model, the remote Dchannel termination (the one whose Dbox has been assigned to a different Mbox) is on the border of the model and has an Mport allocated to it. The remote Mbox is recorded in the border node's label. The Dchannel is created with an Mcall between the two Mboxes, joining the Mports at the border nodes.

4.5.4 Mbox model commands

There are six commands that express all the manipulations of a particular media that a Dbox can perform by itself. These commands open and close Dchannels and Dlinks. The Dbox issues these commands to its Mbox and the Mbox will create representations of the Dchannels and Dlinks in its model and execute the commands. A Dbox issues a command to its Mbox, and must wait for the reply to the command (either ack or nack) before sending another command.

The command

`openlink(db:Dbox,from:Dchanne_termination,to:Dchanne_termination)`

is issued by a Dbox to add a Dlink between two of its own Dchannel terminations when the Dbox receives a `OpenLink` message .

The command

`closelink(db:Dbox,from:Dchanne_termination,to:Dchanne_termination)`

is issued by a Dbox to remove a Dlink between two of its own Dchannel terminations when the Dbox receives a CloseLink message.

The commands

`open2link(db:Dbox,ct1:Dchanne_termination,ct2:Dchanne_termination)`

`close2link(db:Dbox,ct1:Dchanne_termination,ct2:Dchanne_termination)`

are not absolutely necessary, since these two functions can be achieved by using `openlink` and `closelink` twice. Due to the fact that we normally need to add two links in the same Dbox, one in each direction, or close these two links at the same time. These two commands are more convenient to use. The `open2link` command is just like the `openlink` command, except that it opens two Dlinks between the two Dchannel terminations, one in each direction. The `close2link` command is just like the `closelink` command, except that it closes two Dlinks between the two Dchannel terminations, one in each direction. When the Dbox receives an `Open2Link` or `Close2Link` message, the corresponding command will be executed.

The command

`openchan(db:Dbox,mb1:Mbox,ct1:Dchanne_termination,mb2:Mbox,ct2:Dchanne_termination)`

is issued by a Dbox to open a Dchannel when the Dbox receives an `OpenChan` message. The `OpenChan` command causes the Mbox to add a Dchannel to the Mbox model, along with the Dchannel terminations if they are not already present. The two Mbox arguments can be the same or different. If the two Mbox arguments are the

same, then the internal Dchannel will be added in the same Mbox. If the Mbox arguments are different, the cross-border Dchannel will connect these two Mboxes.

The command

```
closechan(db:Dbox,ct:Dchanne_termination)
```

is issued by a Dbox to close a Dchannel when the Dbox receives a CloseChan message.

4.6 Box programming

The ECLIPSE IP telephony system is large and complicated. All software components (objects) in the signaling layer are implemented with Java. Significant abstractions are used to simplify the design and implementation of the ECLIPSE feature boxes. First, UML FSM diagrams are used in the design that allows a programmer to represent a feature as an assemblage of Finite State Machines (FSMs). Second, the software objects of key DFC concepts are standardized, which makes the programmer's implementation easier, to some degree. Each feature box can be expressed as the composition of several FSMs in sequential order and/or nested fashion.

The software objects are organized according to interfaces and class hierarchies. Here are some of the most important software object class hierarchies defined in the DFC/ECLIPSE.

4.6.1 Box class hierarchy

Box is the most important concept and the basic unit in DFC/ECLIPSE. To differentiate from MBox, ECLIPSE refers to the box on the signaling layer as DBox (here D is from DFC). A box class hierarchy encompasses the interface and feature boxes, as shown in Figure 4.4 below.

Dbox is the super class of all box classes. Each DBox has several ports to communicate with other components in DFC. It creates one or more threads to perform box and port operations. DBox will be garbage collected when all threads have been returned and all port connections have been destroyed. This situation can occur when the box port of a box receives a Reset message, or when the box's operations complete normally.

Figure 4.4. Box class hierarchy.

The DBox is extended by line interface box (LBox), trunk interface box (TBox), resource interface box (RBox) and feature box (FBox) classes. LBox and TBox were discussed in DFC architecture (chapter 3). A resource interface box is newly introduced in ECLIPSE. It provides an interface to media-processing resources such as announcement players, message recorders and text-voice converters.

Feature boxes are the most important part of the DFC/ECLIPSE. There are two interfaces: FreeFeatureBox (FFBox) and BoundFeatureBox (BBox). Each feature box can implement one of these two interfaces, depending on the nature of feature. It can be a free feature box or a bound feature box.

For example, the Call Forwarding on Busy feature box (CFB) implements an FFBox. When a router needs an instance of CFB in a usage, a new instance of CFB is created. If two parties in two different places call a third party, there will be two routes to the third party. There will be two instances of CFBs, one on each route. These two boxes operate completely independently.

While the Call Waiting feature box (CW) is a bound feature box, it is bound to the subscriber's address and is not interchangeable. When a router needs an instance of CW in a usage, the router needs to check if an instance of CW associated with the subscriber's address already exists. If it does exist, the router has to connect the call to the existing CW box. If it does not exist, a new instance of CW will be created and incorporated in the usage.

4.6.2 Port class hierarchy

Ports are an essential part of DFC. They are used in the communications between the components. A port class hierarchy is shown in Figure 4.5.

Figure 4.5. DFC Port class hierarchy.

A port is an interface specifying the basic input and output methods and monitors the functions of these input and output activities.

The SwitchPort of the router has two functions: receiving messages from the boxes and sending messages to the boxes. The messages received are stored in a FIFO queue, waiting to be processed.

The BoxPort of the feature box is used to receive messages from the router. It is the peer port of the router's switch port.

LinkablePort is an abstract super class that serves to group the common characteristics of callee and caller ports. It has a defined link state which indicates the status of a link with its peers, whereas other port classes such as BoxPort and SwitchPort don't. The finite state machine in LinkablePort keeps track of the communication state and reacts to the signals received.

CallerPort implements the Caller interface, sending messages to the boxes at the downstream of a usage. It has an output() method. This method not only sends out messages but also initializes the instance of CallerPort at the initial stage of call setup, when a setup message is received by the box port. The setup message will, in turn, be passed on to the switch port by the caller port.

CalleePort implements the Callee interface. This port sends messages to the boxes at the upstream of a usage. It is also initialized by outputting an acknowledge message when a setup message is received by the box port, at the initial stage of call setup.

Dual port implements both Caller and Callee interfaces, and can therefore be used as a CallerPort or a CalleePort as needed.

SignalPathTermination relates to the media channel. It is the common interface implemented by LocalSignalPathTermination and PeerSignalPathTermination classes. It specifies only one method: GetMonitorProperties(). The class hierarchy diagram for SignalPathTermination is shown in Figure 4.6 below.

Figure 4.6. Class hierarchy for SignalPathTermination.

LocalSignalPathTermination represents an endpoint of a signaling path between peer linkable ports on adjacent boxes. A SignalPathTermination is associated with a linkable port. SignalPathTerminations send and receive signaling messages, using the linkable port with which they are associated. Once a SignalPathTermination has been created on a port in a box, the box can then output messages to the SignalPathTermination just as if it were a port itself. A SignalPathTermination also provides the box with control over the media channel associated with the SignalPathTermination. There is a one-to-one mapping between a

SignalPathTermination and a ChannelTermination. The ChannelTermination is a medium channel termination. The SignalPathTermination represents an endpoint upon which signaling messages associated with a particular media channel are sent and received. The line interface boxes do not have direct access to ChannelTerminations; however, operations on signal path terminations can affect their associated media channels based on the model and controller pattern in the media channel management.

PeerSignalPathTermination is an abstract class representing a termination on the signaling path.

LocalPeerSignalPathTermination is a subclass of PeerSignalPathTermination. It implements all the methods specified in its super class. It works as a peer termination of LocalSignalPathTermination and receives the signaling message from the LocalSignalPathTermination.

4.6.3 Finite state machine class hierarchy

A finite state machine is the core of feature boxes. The class hierarchy of finite state machines is shown in Figure 4.7.

FSMOperations is an abstract class that specifies the common operations of FSMs. A finite state machine extends FSMOperations. It is based on UML finite state

machine abstraction customized for DFC. It supports both internal transitions and message transitions.

Figure 4.7. FSM Class Hierarchy.

Internal transitions are transitions that are not triggered by the arrival of a message. An internal transition happens when the current state changes from one to another without a triggering event. Message transitions are triggered by the arrival of a specific message. Either type of transition can define an optional guard condition, based on the values of state variables alone, or with the value and type of the message in the case of message transitions. The guard condition for a transition must be evaluated to be true before the transition can be ready. The transitions can invoke any actions, such as update state variables, output a message, etc., when the transition fires. States can define actions to be performed just before the state is entered and after the state is exited. An initial internal transition is defined with its start state as the state machine's actual initial state, and its end state as the user-defined initial state.

This ensures that an enter action defined for the initial state will be executed when the finite state machine is run.

Nested FSMs can also be defined for states. The current state enters a nested FSM after its parent state's entry action has been performed, and before its exit action is performed. A box programmer can define transitions whose start or end states are states defined in a nested FSM. The box programmer must define the initial state of the specific FSM he is developing.

The finite state machine carries out two important operations: run to completion (`run()` method), or single step (`run(message)` method). When either method is initially invoked, the FSM attempts to fire as many internal transitions as possible at any level.

Using the `run()` method, when the FSM arrives in a state with no enabled outgoing internal transitions, it waits for messages to arrive on ports associated with the current state and any nested states, and evaluates any outgoing message transitions. If no messages are available, then the thread calling the `run()` method will be blocked until a message arrives. If no message transition can fire for the message, then an `FSMException` is thrown. If there is a message transition for that message, a message transition will be fired (`run(message)` method operates). The FSM will then attempt to fire a sequence of as many internal transitions as possible. It will then either return, if there are no more message transitions defined for the current state or

any nested states, or it will attempt to fire a message transition for the next message, and so on.

Using the `run(message)` method, the other software objects have to provide the message with which to evaluate message transitions. The FSM will not wait for messages to arrive using this method. When the FSM arrives in a state with no enabled outgoing internal transitions, it evaluates any outgoing message transitions with respect to the message provided as a method argument. If no message transition can fire for the message, then an `FSMException` is thrown. If there is a message transition for the message, the message transition will be fired. The FSM will then attempt to fire a sequence of as many internal transitions as possible. It will then return.

Figure 4.8 Simple FSM without nested FSM.

Figure 4.8 shows a simple FSM without nested FSM. Its initial state is state1. Before FSM enters this state, an entry action, action1, is performed. If the triggering message arrives and the guard condition evaluates it to be true, then, consecutively, exit action, action2; transition action, msg-action; and entry action, action 3 are fired and the FSM enters state2. There is an internal transition from state2 to state3 and from state3 and state4 respectively. Therefore the FSM fires all the transitions and related actions in the order of intern-action1, action4, intern-action2 and action5.

Figure 4.9 shows a FSM with a nested FSM. a1 is performed first, then the FSM enters state1 and creates a nested FSM, nFSM. Next, a3 is performed and the FSM enters ns1.

Figure 4.9 A FSM with a nested FSM.

In the nested state, ns1, if message m1 arrives and guard condition g1 evaluates it to be true, the FSM enters ns2; if message m2 arrives and guard condition g2 evaluates it to be true, the FSM enters ns3. No matter whether the FSM is in ns1, ns2 or ns3, if message m3 arrives and guard condition g3 evaluates it to be true, a2, a5 and a6 will be performed and the FSM will enter state2. If message m4 arrives and guard condition g4 is true, a7 will be performed and FSM will return to state1. The parent state of the nested FSM can be a history state or normal state. If state1 is a history state, the FSM will return to the nested state from where it left. If state1 is not a nested history state, the FSM will create a new instance of the nested FSM.

4.7 Monitoring and visualization

Monitoring is a very important part of the ECLIPSE system. It provides information about the tasks, such as fault management, security enforcement, performance analysis, visualization, billing, customer care, and data mining. Traditionally these tasks are supported by a separate, special-purpose system which results in redundancy and unnecessary overhead. The ECLIPSE system has developed a single, scalable, flexible, extensible monitoring architecture.

To support scalability, the monitoring architecture is designed to be fully distributed and hierarchically structured. The monitor's probe is located in every box and the monitors are organized hierarchically. There is also a central monitor monitoring the whole system. To support extensibility it is possible to define the monitor's subclasses for monitoring and handling new events. A number of the

ECLIPSE programming abstractions provided for service developers already have monitors embedded in them. For example, the probes in the Box and Port abstractions notify a monitor when a box or port is created or destroyed. A network visualization subsystem is developed in ECLIPSE. This visualization system provides a dynamical picture of a DFC usage as it evolves over time.

4.8 Limitations

Besides those limitations from DFC architecture, ECLIPSE has also some limitations itself as it is still under development. The current implementation only supports media communication between Lboxes on the same router linked to the same Mbox. Time-out is not implemented in the FSMs. There are some bugs in the implementation of Lbox and features such as Blind Call Transfer.

4.9 Summary

A distributive, stable, flexible and media efficient IP telephony system based on the DFC architecture has been developed at application level. In this system, the signaling layer and media layer are separated. The software components in the signaling layer are implemented with object-oriented language Java. The features are modularized as independent feature boxes. The features are implemented as an assembly of several FMS. All the features are deployed on routers inside the network. This feature deployment scheme causes performance problems and does not make good use of the strong processing power of the end user device. The media path in

ECLIPSE follows the shortest possible path between the two end points. ECLIPSE is an ongoing project. It has not accomplished all aspects of the design. For example, it does not support media communication between Lboxes on different routers, or different Mboxes. It also has some bugs in some components.

Chapter 5 DFC with Features Deployed in Dual Places (FDDP version of the DFC)

5.1 Introduction

DFC/ECLIPSE has great advantages in term of feature modularity, feature composition and management of feature interaction over other IP telephony service architectures or protocols. However, in the DFC/ECLIPSE, all features reside inside the network. This feature deployment scheme wastes the strong signal processing and storage power of the end user devices. It causes performance problems, especially when the number of the features is large [15]. Some features, such as voice mail, are security sensitive. Deployment of features at the end user device reduces the number of the features inside the network and protects the user's privacy. In this thesis, the original DFC architecture is modified with the addition of a local feature manager (LFM), a proxy of the user's line interface box and a Blind Call Transfer feature (BCT). In this new DFC architecture, the feature deployment scheme is more flexible than that of the original DFC, and the features can be deployed either inside the network, at the end user devices or both, depending on the characteristics of the features. This new DFC architecture utilizes features deployed in dual places (network and end user device). It is called the FDDP version of the DFC.

5.2 Deployment of Features in End User Device in Original DFC

In the original DFC (ODFC), all the features are deployed on the routers inside the network. The possibility of deploying the features in the end user device has been investigated, and several possible methods are laid out in this thesis. Some typical examples are listed below.

An application program can be written in the end user device which will create instances of each of those features that are supposed to run in the end user device, as shown in Figure 5.1. All these features run in the end user device, but are managed by a router inside the network. Another method is to run a router with the features the user subscribes to in the end user, as shown in Figure 5.2. A further method is to put the dialed zone in the end user device, as shown in Figure 5.3, and run the features the user subscribes to there.

Figure 5.1 shows the signal path set up processes. Fr is an arbitrary feature running on the router inside the network. F1 and F2 are features subscribed to by user b and run in user b's end device. To connect to more features, the router has to send a Setup message over the IP network to the feature running in the end user device. After processing the message, the feature sends an Acknowledge message to the router over the IP network. Obviously sending signals through the network is very expensive in terms of performance. Compared to the time taken sending signals over the Internet, the time taken for a function in the feature to process each message is

very short, for most features. There are a few exceptions, such as 1-800 number translation. These kinds of functions have to search a large database and may take a relative long time.

Figure 5.1. Signaling processes in the signal path setup phase.

This feature deployment scheme is also not suitable for those features that need operational data to function, since all operational data is stored in the router in the current version of DFC from AT&T.

F1 and F2 will not function when the end user device is not connected to the Internet, which may cause reliability problems, depending on the nature of F1 and F2.

More detailed discussion on service reliability will be given later in this chapter.

Another problem is that it is very difficult to evolve and upgrade the system. In the DFC, only the router is allowed to load the features from the provisioning manager. Therefore, when new versions of the features are developed and put in the provisioning manager, the application program running in the end user device will not be able to load these features to upgrade the features running in the end user device.

To reduce the total number of Setup signal trips between the router and the end user devices and solve the operational data problem, a router can be run in the end user device, as can all the features (F1 and F2) to which the user subscribes, as shown in Figure 5.2.

This deployment scheme does have many advantages besides providing a solution to the above problems. These advantages include dynamic loading of telephony features from the central server when they are needed; running features that used to run inside the network on the end user device without any change; and keeping the modularity of the DFC features. However, there is also a reliability problem in this feature deployment scheme, as with the previous case, because the end user device may not be connected to the network all the time. When the end user device is not connected to the network, all the features the user is subscribed to will not function.

Figure 5.2. Running a router and all features on the end user device.

Figure 5.3 shows a usage with another feature deployment scheme in which features are deployed in the dialed zone and the dialed zone is put in the end user device. In the DFC, there is no clear description of what else besides those database checking related features, such as 1-800 number translation, credit card call etc., will run in the dialed zone. If an application program is used to create those features running in the dialed zone, then it must provide the same feature management as the router does. Therefore we suppose that there is a router running in the dialed zone to manage the features. There is also a reliability problem at present, as with running all the features on the router in the end device. If the end user device does not connect to the network, the call signal will be cut off in the middle of the call signal path. This deployment scheme is the worst of all these three schemes, since the dialed zone is designed to be the place where database-related services, such as the 1-800 number translation service, reside. Putting database related features in the end user devices causes security problems and difficulty in maintaining the consistence of the multiple databases in the different end user devices.

Figure 5.3. Features run on dialed zone at the end user device.

5.3 FDDP version of the DFC

In this section, the FDDP version of the DFC architecture and each of its new components will be discussed. Basic scenarios showing how components in the new architecture work together to setup the call are presented. The reliability of services (features) will be discussed by means of examples.

5.3.1 Components

Figure 5.4 shows the components in the modified DFC architecture, i.e. the FDDP version of the DFC architecture. The new architecture has three more components: a local feature manager, a proxy of the user's LI box and a Blind Call Transfer feature. The FDDP version of the DFC consists of two parts: a system

network, which is the same as the original DFC network, and end devices. The feature boxes reside either on the system network or in the end user devices, or both. The local feature manager, the user's real LI box (LIb) and those features subscribed with the real LI box address, run in the end user device. The Blind Call Transfer feature and the proxy of the user's LIb are deployed on the router inside the system network. We will discuss the new components in FDDP version of the DFC in detail.

Figure 5.4. FDDP version DFC architecture.

Local feature manager (LFM)

The Local Feature Manager is a router. It is responsible for the management of the features running in the end user devices, creating the feature boxes, dynamically assembling feature boxes, loading features from the central server and updating the features. Some methods of the router have to be overridden, as the responsibilities of the local feature manager are slightly different from those of the router. These methods include creating trunk interface boxes, handling trunk setups or tearing down related messages, since we do not allow end user devices to be the gateway connecting to the other networks.

Proxy of user's LI box

The proxy of user's LI box is a LI box. The only reason for the introduction of the proxy of the user's LI box is that the user can subscribe to those features that have to be run inside the network, using the proxy's address. A detailed discussion on why some features have to be deployed inside the network will be given later in this section.

Blind Call Transfer (BCT)

Blind Call Transfer is a feature that transfers the call to another phone once it receives a call setup signal, according to the operational data preset by the user. Here it is a mandatory feature for those users who would like to run features in their end user devices. BCT is implemented as a free feature, since many callers may call the same callee at the same time. Each instance of BCT serves each call. The primary

reason for the introduction of BCT in the new architecture is that it can redirect the call to the user's real line interface box in the end user device, so that all the features along the call path can process the call. These features include the features running inside the network subscribed to with the user LI box proxy's address, as well as features running in the end user device subscribed to with the user LI box's address. Here, a convention for the proxy's address and user LI box's address is used. As the proxy represents the real user LI box to the other users in system network, we use the user name as the proxy's address and use the user's name plus 0 as the address for the real user LI box. This convention makes it very convenient for BCT to redirect the call. However, it is not strictly necessary. We can also use any other combination of characters as the address for the real user LI box, as long as the address is not used by another user, and set it as an operational data for the BCT feature.

5.3.2 Basic Scenarios

In this section we will show how the components in the FDDP version of the DFC work together to set up a call in which the features inside the network and in the end user device are involved. In the FDDP version of the DFC as shown in Figure 5.4, user A subscribes F1 feature. User B subscribes to some features - F2 and BCT - on the router B with his LIb's proxy (LIbp). User B also subscribes to some features - F4 and F5 - on the local feature manager, running in the end user device, with his real LIbox, LIb. When user A calls user B, the routers route the call from LIa to user B's feature F2 and then BCT. After BCT receives the Setup signal, it returns an ACK message and then redirects the route to user B's real LI box, LIb, without passing

through the proxy LIbp. Next, BCT sends a Setup signal to the local feature manager (LFM), and LFM passes the Setup signal to F4 on LFM. F4 sends ACK to BCT. The call connection continues until it finally reaches LIb. The signals go from user B to user A, through the same path in reverse order. LIb is a normal line interface box that interacts with user B. All the features residing inside the network and the features in user B's end device process user B's call.

5.3.3 Examples of usages

In the IP telephony network, the end devices may not be connected to the network all the time. Therefore the reliability of the services in the end user devices has to be considered. As Internet technologies develop rapidly and the performance and quality of computers continues to improve, in the future the computer may be connected to the Internet all the time, or most of the time, in the same way that conventional telephones are now. However, in the following paragraphs, the reliability of several services such as Voice Mail (VM), Call Forward on Busy (CFB) and incoming call screen will be discussed based on the assumption that the end user devices may not be connected to the network all the time.

Figure 5.5. Usage of VM at the end device and in the network.

Voice mail

Voice mail is one of the most popular telephone features. There are three possible ways to deploy it: in the end device, inside the system network, or both. As the storage capability of the end user device has become larger and larger in recent years, it is possible to store a large amount of messages in the end user device. If VM is deployed in the end user device, significant storage space inside the network is saved, the workload inside the network is reduced, the user's telephony service cost is reduced and the security of the voice mail is also improved. However, when the end user device is not connected to the network, in order to guarantee the reliability of the service, voice mail inside the network is necessary as a backup. Figure 5.5 shows the usage diagram of voice mail.

Call Forward on Busy

Call Forward on Busy (CFB) is another popular telephony feature which provides the busy signal treatment. This feature only springs into action when it receives a busy signal. If the CFB is deployed inside the network, when the end user device is not connected to the network, the CFB will not receive a busy signal since the busy signal is generated from the user's LIbox in the end user device. If the CFB does not receive a busy signal, it will not function; it will behave transparently as if it does not exist. When the end user device is connected to the network, the CFB will function no matter whether it is deployed inside the network or in the end user device. To save network resources, it is better to deploy the CFB in the end user device. Figure 5.6 below shows a usage diagram of the CFB deployed in the end user device.

In this usage, when user a calls user b, if user b is already engaged in a conversation on the phone, a busy signal will be sent to the CFB, which will then forward the call to LIc, which has been specified in the operational data for the CFB. Here F is an arbitrary feature subscribed by user a and user c.

Figure 5.6. A usage of CFB in the end user device.

Figure 5.7. A usage of ICS and CFB

Incoming Call Screen (ICS)

Incoming Call Screen is another feature that can be deployed in the end devices. If the end device is not connected to the network, the incoming call will not reach the user. When the device is connected to the network, the ICS feature is better deployed in the end user, where it screens out the unwanted calls before the other

features process the call signals for the user, as the ICS has been assigned highest priority for the downstream signaling at the LFM.

There are many more features can be deployed in the end devices [25].

5.4 Summary

In recent years, the performance of semiconductor devices has improved at a dramatic rate. As the end user devices become more and more intelligent and powerful, there is a strong trend towards moving IP telephony services (features) to the end user devices. The advantages of deploying features in the end user devices include reducing the workload in the network, making use of the strong processing power of the end devices, reducing the user's telephone cost and enhancing the security. However, in the original DFC/ECLIPSE, all features reside inside the network. The FDDP version of the DFC proposed in this thesis provides the flexibility for deployment of features both in the end devices and inside the system network. The new DFC architecture also has the following advantages: a) dynamic loading telephony of features from the central server when they are needed; b) features run inside the network can run in the end user device without any change; c) users can subscribe to features both in the network and in their own devices, to achieve reliability of services and improved performance; d) the modularity of the DFC features is retained.

Chapter 6 Development of Voice Mail System based on the FDDP version of the DFC

6.1 Introduction

In the last chapter, the FDDP version of the DFC was proposed. This new architecture allows features to be deployed in the end user device (EUD), inside the network or in both the end user device and inside the network, according to the nature of the features. The voice mail system is used as an example to show the deployment of features both in the end user device and inside the network. The development of the voice mail system formed part of our project with AT&T, and the voice mail feature is also identified as one of the more challenging IP telephony features [26].

6.2 Voice Mail System

The voice mail system is ideally deployed in the end user device, since it can make use of the computing and storage power of the end user device and save resources such as CPU and storage space on the IP telephony network. However, as mentioned earlier, the end user device may not be connected to the network all the time, and a voice mail system inside the network is needed as a backup.

The voice mail system consists of a voice mail feature (VM) and a voice mail center (VMC). If the end user is not available, VM will transfer the call to the voice

mail center where the call will be handled and the message stored in the callee's mailbox. The voice mail center is designed as a resource interface (RI) box. An RI box is a collection of LI boxes that can accommodate a number of calls simultaneously, by dynamically assigning an LI box to each call [10,23]. Since there may be more than one caller at the same time and each call is independent, it is necessary to have an instance of the VM feature to direct each call to the voice mail center. Therefore VM is designed as a free feature by implementing the FFBox interface.

Figure 6.1. Deployment of VM in end user device and inside the network.

Figure 6.1 shows VM deployed both in the end user device and inside the network. The voice mail center corresponding to the VM in the end user device is deployed in the end user device. The voice mail center corresponding to the VM inside the network is deployed inside the network.

6.2.1 Usage diagrams

The voice mail system has to function well in the following two cases. In case one, the end user device is connected to the network. In the other case, the end user device is not connected to the network. The first case consists of further two sub cases. In first sub case, the callee is busy talking to other people on the phone when the caller calls. In the other sub case, the callee does not pick up the phone.

6.2.1.1 Leaving a message when the end user device is connected to the network

Figure 6.2.a shows the scenario for leaving a message in the voice mail center when the end user device is connected to the network. The voice mail systems deployed in the end user device and inside the network. One instance of the voice mail feature is deployed in the Llb target zone and one is deployed in the target zone of Llb's proxy.

User a calls user b, but user b is not available because user b is busy talking to user c, or user b does not answer. The VM at the end user device is designed to function when either of the above two cases happens. It redirects the call to the VMC in the end user device.

(a)

(b)

(c)

Figure 6.2. Usages for the voice mail feature and voice mail center. a: leave a message when the end device is connected to the network, b: leave a message when the end device is not connected to the network. c: retrieve a message.

User a calls user b, but user b is not available because user b is busy talking to user c, or user b does not answer. The VM at the end user device is designed to function when either of the above two cases happens. It redirects the call to the VMC in the end user device.

When the user b is busy, the LIb will send out a busy signal (Upnack) after it receives a setup signal. As the VM at the end user device is closer to the LIb box than that inside the network, it will absorb the signal and act.

When user does not answer, VM will function based on the time-out. To guarantee the VM at the end user device instead of the VM inside the network to function, the operational data (time-out) for the VM at the end user device is set to 20s and that for the VM inside the network is set to 30s. With the above time-out setting, when the end user device is connected to the network, the VM at the end user device will function. The VM inside the network and the BCT behave transparently and simply pass on the messages.

6.2.1.2 Leaving a message when the end user device is not connected to the network

Figure 6.2.b is the usage diagram for leaving a message when the end user device is not connected to the network. In this case, the VM inside the network should function. It can not receive busy signal generated by the LIb since LIb does not exist anymore. It will redirect the call to the voice mail center inside the network

after it receives the teardown message from the adjacent feature in the downstream direction of the call.

6.2.1.3 Retrieving a message

Figure 6.2.c shows the scenario for retrieving a message. User b calls the voice mail center to retrieve his messages.

6.2.2 Sequence diagrams

6.2.2.1 Leaving a message when the end user device is connected to the network

Leave Message on Busy

Figure 6.3 shows the detailed sequence diagrams for the usage of the voice mail feature when the callee is busy. VMi and VMe are VM inside the network and VM at the end user device respectively.

The first part of the figure starts with LIa sending a Setup message to LIb, and ends with the VM receives Downack message. It shows that LIa tries to connect to LIb, while the callee is on the phone. When LIb is busy, if it receives a Setup message, it will reply with an Upnack message. After the VM feature receives the Upnack message, it sends a voice mail center address message (VMCAddrMsg) to the LIa to leave a voice mail center address to the LIa, then it sends out a Teardown message to the adjacent features to teardown the current call path. Upon receiving the Teardown message, the adjacent feature passes the message on and then acknowledges it by sending back a Downack. After receiving the VMCAddrMsg, the

Figure 6.3.a. Sequence diagram for a voice mail usage: leaving messages, when the callee is busy (Part 1).
ax are signal sequence numbers related to VM.

Figure 6.3.b. Sequence diagram for a voice mail usage: leaving messages, when the callee is busy (Part 2).
bx are signal sequence numbers related to VM.

L1a sends out a teardown message and tears down the current call. Then the L1a makes another call to the VMC to leave a message. Messages a6 and a6' are a pair of message and acknowledgement, so are messages a7 and a7'.

The second part of the figure shows the call setup steps for the signal and media channels, the media conversation and the teardown steps of the signal and media channel between the caller and VMC. Upon receiving the VMCAddrMsg, the L1a automatically and directly sets up a call to the voice mail center without the involvement of the VM and BCT features since only user b subscribes to these features while the VMC doesn't. After the signal channel has been successfully established, indicated by L1a receiving an available (Avail) message from the voice mail center, the L1a starts setting up the media channel.

The diagram shows the media channel setup steps and use of the media channel, from the L1a sending the Open message to sending the teardown message. Upon receiving the Avail message, it is time to set up a Dchannel between L1a and VMC. The L1a sends an Open message to the VMC. Upon receiving the Open message, the VMC sends an OpenChan message to the Mbox to allow the Mbox to set up a media channel between the L1a and VMC. After the Mbox sets up the channel, it acknowledges to the VMC by sending an OpenChanAck. The VMC then sends an OpenAck back to the caller. The L1a sends back a Ready message to the VMC. The VMC sends a Wait message back to the L1a. As soon as the VMC has finished sending the Wait message, the VMC loads the greeting from the disk. After

VMC sends out the Accept Message, VMC immediately sends the caller greeting and query through media channel. The greeting will be something like, “Welcome to the AT&T voice mail center. If you want to leave a message, please press the leave button and leave a message after the beep. If you want to retrieve your messages, please press the retrieve button”. At present, if the caller wants to leave a message, he can press the leave button, which sends a LeaveMsg message to the VMC. Once the VMC received the message, it sends a “beep” to the caller. The VMC then waits for the media message from the caller. After hearing the beep, the caller starts to send a media message to the VMC. The VMC receives the media and stores it in the file.

After leaving the media message, the caller hangs up the phone, and the LIa sends a Teardown message to the VMC. Finally both the LIa and VMC send a CloseChan message to the MBox to close the media channel between them, and the Mbox acknowledges. As the code for MBox and Magaco is not available from AT&T, the part of the sequence diagram for closing the media channel is based on the design document from AT&T[10] and a sequence diagram from AT&T[33] instead of the real source code, therefore it may have a minor difference between the current part sequence diagram for closing the media channel and the real one. However, those basic building components of the line interface box are reused here and they interact with the MBox and Magaco to close the media channel. Therefore, the part of the code developed in this thesis does not deal with the part of the sequence diagram for closing the media channel. This fact is also true in the closing channel part of the sequence diagrams in the following sequence diagrams.

Leave Message on No Answer (LMNA)

Even though the callee is online, he may not want to answer the call. In this case, a time-out has to be set so that the VM can teardown the connection between the caller and callee after the time-out and redirect the caller to the VMC. A time-out is not implemented in the current DFC/ECLIPSE finite state machine, but is introduced in the VM feature in this thesis. Figure 6.4 shows a detailed sequence diagram for the usage of the voice mail feature to leave a message on no answer.

The first part of Figure 6.4 starts with L1a sending a setup message to L1b and ends with the teardown of the call to the callee. It shows the steps for setting up a signal channel and media channel between the caller and the callee. Once the caller has received the Avail message, it is time to set up the media channel between the caller and the callee. The media channel is established by setting up a Dchannel between each pair of sequential boxes and two links within the same feature box to achieve bi-direction media connection. The caller sends an Open message to the callee via VMi, BCT and VMe features. Upon receiving the Open message, the VM feature sends an OpenChan message to the Mbox. The Mbox acknowledges by sending back an OpenChanAck message. The VM then sends an Open2Links message to the Mbox, which acknowledges. The VM passes the Open message to the callee. Upon receiving the Open message, the callee sends an OpenChan message to the Mbox to set up a Dchannel between the VM and the callee. After setting up the channel, the Mbox acknowledges to the callee. The callee answers the Open message by sending back a OpenAck message to the caller through VMe, BCT and VMi

features. Upon receiving the OpenAck message, the caller sends a Ready message to the callee through the above features. After receiving the Ready message, the callee sends back a Wait message to the caller through the features. As soon as the VM receives the Wait message, a timer begins counting down. If the callee answers the phone within the time-out, the call will be a normal call. Otherwise, the VM will send a VMCAddrMsg to the Lla, which will change the call to a voice mail call once the system has closed the signal and media channels. The part of the sequence diagram for closing media channel is may not exactly same as the real one due to the reason mentioned earlier.

The second part of the diagram shows the setup steps for signal and media channels, media conversation and leaving a message, and the teardown steps of the signal and media channel between the caller and VMC, which are the same as the second part in Figure 6.3.

To be continued.

6.2.2.2 Leaving a message when the end user device is not connected to the network

Figure 6.5 shows the detailed sequence diagrams for the usage of the voice mail feature when the callee's end device is not connected to the network. The software objects in the gray area are the objects supposed to be run at the end user device. DFC does not allow running an independent router without registering the router to the central server through the Internet since all the data the router needed are from the central server. LFM is a router. Therefore when the end user device is not connected to the network, the user cannot run the LFM and those objects running on the LFM, such as features and LI boxes.

The first part of the figure starts with LIa sending a Setup message to LIb, and ends with LIa receiving the Downack message. It shows that LIa tries to connect to LIb, while the callee's end device is not connected to the network. After VMi receives a Setup message, it replies with a UpAck message and continues the call setup by passing Setup message to the next feature BCT. BCT continues the call setup by sending Setup message to the router. The router parses the Setup message and finds the callee is not online then sends back an UnAvail to the BCT. After BCT receives the UnAvail, it sends a Teardown message to VM. After the VM feature receives the Teardown message, it sends VMCAddrMsg to the LIa to leave a voice mail center address to the LIa and then acknowledges the Teardown message by sending back a Downack message to BCT. After sending the Downack to BCT, VM sends a Teardown message (d8) to the LIa box. After receiving the VMCAddrMsg,

the Lla sends out a Teardown message to VM (d9). Lla sends a Downack (d8') to acknowledge the Teardown message (d8) from VM. VM acknowledges the Teardown message (d9) by sending back a Downack(d9'). The current call is torn down. It is time for Lla to make another call to the VMC to leave a message.

The second part of the diagram is to setup a call between the caller and VMC and leave a message with the same steps as part2 in figure 6.3 except that the VMC is inside the network.

Figure 6.5. Sequence diagram for a voice mail usage: leaving a message when callee's end device is not connected to the network. dx is signal sequence number related to VM.

6.2.2.3 Retrieving messages

The message retrieval sequence diagram is shown in Figure 6.6. To retrieve messages stored in the VMC, the user has to call it. After the signal and media channels have been established, the VMC will send a greeting and query to the caller: "Welcome to the AT&T voice mail center. If you want to leave a message, please press the leave button and leave a message after the beep. If you want to retrieve your messages, please press the retrieve button." Once it has received a Retrieve message, the VMC will send the information about the voice mail to the caller and ask the caller if he or she would like to retrieve the information or end the call. If the caller answers "yes" in the acknowledge message, the VM will send all the new messages to the caller. After sending all the new messages to the caller, the VMC sends a Teardown message to the caller to teardown the call. Each of Lib and VMC sends CloseChan to the MBox to close the media channel.

Figure 6.6. Sequence diagram for retrieving messages. ex is the signal sequence number.

6. 3. Implementation

In the DFC, features are implemented by assembling a number of finite state machines in the sequence and/or nested fashion. As a finite state machine in DFC defines only one `run()` method as mentioned in the chapter 4. Therefore the documentation for the implementation of DFC objects is different from the documentation of normal application software, ie., a comment for each method. Here the documentation follows the way used by AT&T for documentation of objects in the Eclipse project.

In this thesis, if it is possible, the objects in the software library from AT&T are reused so that we do not reinvent the wheel. In the implementation of the VM feature, `CallerTeardownFSM`, `CalleeTeardownFSM` and `Open2LinksFSM` are reused. In implementation of voice mail center, `OpenExtChanFSM`, `Open2LinksOnPortFSM`, `AnswerFSM`, `MediaAnswerFSM` are reused. The LI box is modified to accommodate the additional functions to leave and retrieve messages, due to the addition of the voice mail system. The details of the modification of the LI box will be given later in this chapter.

Before we examine a specific FSM, it is necessary to introduce some notations. $P1?m$ represents that Message m input from port $P1$; $P1!m$ represents that port $P1$ outputs Message m . $S1 \xrightarrow{P1?m1 \text{ [guard]} / P2!m2; \text{action}} S2$ means that at state $S1$, if $P1$ receives $m1$ and guard evaluates to true, then transition enabled, port $P2$ sends out $m2$ and action is executed resulting in state $S2$

6.3.1 Voice Mail Feature

The voice mail feature is implemented based on the previous sequence diagrams, Figures 6.3, 6.4 and 6.5. The voice mail feature is designed to redirect the call to the voice mail center, where the caller can leave a message, when the callee's end device is connected to the network, if the callee is busy talking to other people on the phone or the callee does not want to answer the phone; or when the callee's end device is not connected to the network.

The voice mail feature box consists of a voice mail box finite state machine (VoiceMailBoxFSM), which controls all the operations of the feature, a caller port and a callee port and a box port. To show the communication between the above ports and other objects, Figure 6.7 is drawn. CallerPort is the port used to communication with the next feature, CalleePort is used to communication with the last feature along the calling path.

Figure 6.7. Relationship between the CallerPort, CalleePort, BoxPort and other objects. LF is the last feature and NF is the next feature on the calling path.

The voice mail box finite state machine further consists of a Transparent2LinksAct finite state machine, and a CallerTeardownFSM and a CalleeTeardownFSM, developed by AT&T, are reused here to teardown the caller port and the callee port. The Transparent2LinksAct finite state machine further consists of a TransparentAct finite state machine and an open2Links finite state machine. To show the correlation between the signal sequence diagrams and the FSMs, the signal sequence numbers are put in brackets in the FSM diagrams.

VoiceMailBoxFSM

Figure 6.8. VoiceMailBoxFSM

The VoiceMailBoxFSM is used in the first stage in the setup of the signal path, the creation of sub-finite state machines for further signal processing and the media channel setup and teardown of the call. As shown in Figure 6.6, there are four states in the VoiceMailBoxFSM.

At the Init state, when the box port receives a setup message from the router, the callee port sends an Upack message to acknowledge that the setup message has been received. The caller port then sends out the “continue setup” message and the current state changes to the LinkCaller state.

In the LinkCaller state, if the caller port receives the Upack message, the VM will follow the sequence diagram shown in Figure 6.3. VM will send the voice mail center address message (VMCAddMsg) to the caller and the current state moves into the End state.

In the End state, CallerTeardownFSM and CalleeTearDownFSM are created to tear down the caller and callee ports.

In the LinkCaller state, if the caller port receives an Upack message, that means the callee line interface box Lib is not busy, the VM will follow the sequence diagram shown in Figure 6.4, and the current state moves into the TransparentAct state.

After entering the transparentAct state, the voice mailbox finite state machine creates a nested Transparent2LinksActFSM, which selectively allows the messages to pass through the caller and callee ports transparently and open the media signal channel.

In the TransparentAct state, once the callee port receives a teardown message, the current state changes to the End state, where a CallerTeardownFSM and Callee TeardownFSM are created to destroy the caller and callee ports. If the caller port receives a teardown message, the callee port sends out a VMCAAddMsg to the Lla and then the current state changes to the End state.

Transparent2LinksActFSM

Figure 6.9. Transparent2LinkActFSM.

This finite state machine sets up a media connection between two linkable ports, Port1 and Port 2, allowing the messages to be propagated transparently. Port 1 and Port2 are created by casting the caller port and the callee port to the linkable port respectively. When an open message arrives, the FSM attempts to establish a media channel link between the two ports. There are two states: the TransparentAct state and the OpenLink state in the Transparent2LinksActFSM.

In the TransparentAct state, a nested TransparentActFSM is created. In the TransparentAct state, when linkable port 1 or port2 receives an open message, the finite state machine initializes the alias (opener port and openee port) of the two linkable ports and creates two signal path terminals (opener and openee), which correspond to the two alias ports. Between port1 and port2, the first one that receives the open message will be assigned as the opener port and the other will be assigned as the openee port. The opener is the local signal path termination associated with the opener port. The openee is the local signal path termination associated with the openee port. The current state then changes from the TransparentAct state to the OpenLink state.

In the OpenLink state, the Open2LinksFSM is created and if, in this state, the openee port receives an open message, it sends back an Onack message to reject any attempts to set up a link from the openee, since the opener is already engaged in a media channel setup. In the OpenLink state, the status messages from either opener or openee port propagate through transparently. After the media channel links have

been opened and the current state has been changed to the link opened state of the Open2LinksFSM, then an internal transition takes place from that state to the transparent state without any actions. If the opening of the media channel fails, i.e. the current state has been changed to a link unopened state of the Open2LinksFSM, an internal transition from that state to a transparent state occurs. With this internal transition, the opener and openee ports are destroyed.

Figure 6.10. Open2LinksFSM.

Open2LinksFSM

The Open2LinksFSM developed by AT&T is used to set up a bi-directional media link between the two local signal path terminations. There are four attributes in the Open2linksFSM: the LocalSignalPathTermination opener and openee; the LinkablePort openerPort; the openeePort; and eight states: start, OpenChannel, OpeningLink, WaitingForOpenee, ClosingLink, ClosingChannel, LinkOpened and LinkUnopened. The media connection can be ended in the LinkOpened state if everything goes well, or the LinkUnopened state if an Onack is received from the Openee, as shown in Figure 6.10.

The current state changes from the Start state to the OpeningChannel state through an internal transition. In the transition, the opener sends out an open channel message.

In the OpeningChannel state, upon receiving an OpenChan message, the opener outputs a Open2LinksMessage to the openee and the current state changes to the OpeningLink state. In the OpeningChannel state, upon receiving an Open2LinksAck, the openee sends out an open message and the current state changes to the WaitingForOpenee state.

In the WaitingForOpenee state, if the openee receives an Oack message, the opener will pass the Oack message on to acknowledge the open channel request from the caller and the current state changes to LinkOpened. In the WaitingForOpenee

state, if the openee receives an Onack message, then the opener sends back a Close2Links message to the caller and the current state changes to ClosingLink state.

In the ClosingLink state, upon receiving a Close2LinksAck message, the opener sends out a CloseChan message to close the channel. The current state then changes to ClosingChan state.

In the ClosingChan state, when the opener receives a close channel acknowledge, the channel with the opener closes and the current state changes to the LinkUnopened state.

TransparentActFSM

The TransparentActFSM is one of the most important FSMs in the Voice Mail Feature. It consists of two linkable ports, port1 and port2, a Time Event Generator, a timer and one TransparentAct state. It propagates the DFC protocol messages between two linkable ports, except for WaitMessage and AcceptMessage, as shown in figure 6.11. Once the port1 has received the WaitMessage, the timer is started. If within the time-out limit, port1 does not receive an AcceptMessage, the timer will generate a time event. After receiving the time event, the time event handler sends a VMCAddMsg, which has a string attribute holding the address of the voice mail center, to the caller through the port1 signal path termination. Upon receiving the text message, the caller's finite state machines sets up a connection with the voice mail center and leaves a message there.

If within time-out, port1 receives an Accept message, then a flag is set that stops the finite state machine sending the VMCAAddMsg to the caller; and port2 passes an Accept message on to the caller. The caller will then talk to the callee as a normal call.

Figure 6.11. TransparentActFSM.

6.3.2 Voice Mail Center

The voice mail center is implemented as a voice mail center box, which consists of a voice mail center box finite state machine (VoiceMailCenterBoxFSM) and a GUI-based management interface. The VoiceMailCenterBoxFSM further consists of an open external channel finite state machine (OpenExtChanFSM) and a voice mail center line finite state machine (VoiceMailCenterLineFSM). The voice

mail center line finite state machine consists of a mail box line finite state machine (MailBoxLineFSM), which in turn consists of an Open2LinksOnPortsFSM and a mail box busy finite state machine (MailBoxBusyFSM). The busy finite state machine consists of an answer finite state machine (AnswerFSM). The above finite state machines may also consist of some sub-finite state machines. All these finite state machines work together to process signals and set up media connections.

The voice mail center ideally extends the RI box. However, the RI box is not yet implemented in the current version of ECLIPSE. Therefore, in this thesis, the voice mail center is designed and implemented by extending the line interface box that only allows one caller to leave a message at a time. Once the RI box implementation is finished, we can put the current implementation into the RI box (the new voice mail center) with only a slight adjustment, since every call to leave a message in the callee's mailbox follows the same procedures. The above compromise does not affect the research in this thesis to any great extent, as our main goal is to study the deployment, design and implementation of the voice mail feature.

Voice Mail Center Box Finite State Machine

The voice mail center box finite state machine is similar to PhoneGuiDisplayBoxFSM developed by AT&T, has a box port, a dual port, a GUI port and only one state, LineState. The finite state machines nested in these two finite state machine are different. In the LineState, the finite state machine creates a nested finite state machine, a voice mail center line finite state machine

(VoiceMailCenterLineFSM). The voice mail center line finite state machine further creates a nested mail box line finite state machine. The line finite state machine further creates a mail box busy finite state machine. The details of all of the above sub-finite state machines will be shown later. The following figure shows only the states related to the voice mail center box finite state machine.

Figure 6.12 Voice Mail Center Box Finite State Machine

In the Line_State/VoiceMailCenterLineFSM.Line_State/MailBoxLineFSM.Idle_State, after the box port has received a setup message, the finite state machine records the caller's address, the dual port replies with an acknowledge message to the caller, and the current state change to a Line_State/ VoiceMailCenterLineFSM.Line_State/MailBoxLineFSM.Busy_State/MailBoxBusyFSM-.Receiving_Call_State.

In the Line_State/ VoiceMailCenterLineFSM.Line_State/ MaiBoxLineFSM Busy_State, when the box port receives the setup message, the finite state machine rejects the call by replying with an Upnack message through the dual port.

VoiceMailCenterLineFSM

Figure 6.13. Voice Mail Center Line Finite State Machine.

VoiceMailCenterLineFSM is similar to LineFSM developed by AT&T. The finite state machines nested in these finite state machines are different. There are four states: Start_State, Open_Ext_Chan_State, Line_State and End_state in the VoiceMailCenterLineFSM. In any of the above states, if the finite state machine

receives a WinCloseMsg from the guiPort, it will enter the end state and teardown the call.

The finite state machine starts with a Start_State, then changes into an Open_Ext_Chan_State through internal transition.

In the Open_Ext_Chan_State, an OpenExtChanFSM, which is responsible for opening the external media channel to the user, is created. Once the OpenExtChanFSM has finished its job, the state changes from OpenExtChanFSM.End_State to Line_State through internal transition.

In the Line_State, the current finite state machine creates a MailBoxLineFSM and the MailBoxLineFSM creates a MailBoxBusyFSM.

OpenExtChanFSM

This finite state machine is developed by AT&T is reused to set up a UDP media channel to the remote peer. It has three states: Start_State, Wait_Ext_Ack_State and End_State.

In the Start_State, each of the connecting pairs sends its own OpenExtChanMsg that contains an IP address, port number and local media type to the Mbox through its external termination, and the current state changes to a Wait_Ext_Ack_State.

In the Wait_Ext_Ack_State, after receiving the acknowledgement message, which contains the IP address, port number and media type of its remote pair, each of the connecting pairs stores the message. The message will be used later when each sends media message to its peer.

Figure 6.14. OpenExtChanFSM.

MailBoxLineFSM

MailBoxLineFSM is similar to LineFSM developed by AT&T. The finite state machines nested in these two finite state machines are different. A MailBoxLineFSM starts with a Start_State. The current state then changes to the Open2Links_State where Open2LinksOnPortFSM, developed by AT &T, is created. After Open2LinksOnPortsFSM has finished its duties, the current state changes to

Idle_State in MailBoxLineFSM from the Links_Opened_State in Open2LinksOnPortsFSM. Before entering the Idle_ state, an entry action: set gui in idle mode is invoked.

In the Busy_State, if the dual port receives an Open message, it will return an Onack message to the caller. Once the dual port has received a Teardown message, dual port will be torn down before the current state changes from the busy state to the Restart_State. Finally the current state changes from the Restart_State to the Start_State through internal transition.

Figure 6.15. MailBoxLineFSM.

MailBoxBusyFSM

MailBoxBusyFSM starts with an `Initiating_Call_State`. The entry action, `gui.remoteCalling- Mode()`, has been carried out as the current state enters the state. This entry action enables the GUI status label to indicate that someone is calling. Through internal transition, the current state changes to the `Receiving_Call_State`, and an `AnswerFSM` is created. There is an internal transition between `AnswerFSM.Connected_State` and the `Connected_State`. After the signal and media channels are connected, a finite state machine for leaving and retrieving messages (`LeaveRetrieveMsgFSM`) is created at the `Connected_State`. There is also an internal transition between the `AnswerFSM.Disconnected_State` and the `Disconnected_State`. The current state changes to the `Reset_State` from the `Disconnected_State` after running the enter action, `dualPort.teardown()`.

Figure 6.16. MailBoxBusyFSM.

AnswerFSM **Figure 6.17. AnswerFSM.**

The AnswerFSM is developed by AT&T and reused here. It starts in the Start_State. Once a dual port, answerPort, has sent out an Avail message, the current state changes to the UnOpened_State. In the UnOpened_State, after a local signal path termination answerSpt, which is associated with the answerPort, receives an OpenMessage, the current state changes to the Media_Answer_State. In this state, a Media_AnswerFSM is created. There are two internal transitions: one is from the Media_Answer_State/ Media_AnswerFSM. Connected_State to the Connected_State, and the other is from the Media_Answer_State/ Media_AnswerFSM. Disconnected_State to the Disconnected_State.

Media_AnswerFSM **Figure 6.18. MediaAnswerFSM.**

The MediaAnswerFSM is developed by AT&T and reused here. It starts with a start_state. Once the answerSpt has sent out an OpenChan message, the current state changes to the Opening_Chan_State. In this state, if the answerSpt receives an OpenChanAckMsg, it sends out the OAck message, the current state changes to OAck_Wait_State. If the answerSpt receives an OpenChanNackMsg, the current state changes to Disconnected_State. Before the current state enters the Disconnected_State, answerSpt sends out RejectMsg. At OAck_Wait_State, upon the answerSpt receiving a Ready message, it sends out WaitMsg and the current state changes to the Local_Alert_State. There is an internal transition between the Local_Alert_State and the Connected_State. Before the current state changes to the

Connected_State an entry action is executed in which the answerSpt sends out the AcceptMsg.

LeaveRetrieveMsgFSM

Figure 6. 19. LeaveRetrieveMsgFSM

The LeaveRetrieveMsgFSM starts in the Start_state with an entry action: sending greeting and queries to the user such as “Welcome to the AT&T voice mail center. If you want to leave a message, please press the leave button and leave a message after the beep. If you want to retrieve your messages, please press the retrieve button.” If the answerSpt receives a LeaveMsg, a “beep” will be sent through the media channel and the current state will change to the LeaveConfirmed state. The

FSM will wait, receive and save the media message from the caller and the current state will change to the Received state through an internal transition. In the Received state, if the answerSpt receives a Teardown message, the current state will change to End state where the dual port will be teardown. In the Start state, if the answerSpt receives a RetrieveMsg, the voice mail information and query - something like, “You have one new messages. Would you like to read the message?” will be sent to the caller through the media channel and then the current state enters RetrieveMsgConfirmed state. If the answerSpt receives a RetrieveMsg again, the messages will be sent to the caller through the media channel and current state enters Msg_Sent state. After all the messages have been sent, the dual port is torn down and current state changes to End state through an internal transition.

6.4. Modification of LI box

Adding VM into the system introduces three new buttons (Leave, Retrieve and NoRetrieve) and their event handlers on the phone GUI. When each of the three buttons is pressed, a corresponding message is sent to LI box finite state machine. For example, if the Leave button is pressed, a Leave message is sent out.

Another modification is that a new message transition is added in BusyFSM of the LI box at the Call state. At this state, if a VMCAAddrMsg is received, the current state will change to the Disconnect state in which the finite state machine will tear down the current call and set up another call.

6.5 Test

As mentioned in Chapter 4, the current implementation from AT&T does not support two routers. Ms. Yuxiao Wang of our group modifies the router code and the modified code supports two routers on the same Mbox. The modified router code is used in the current test.

The FDDP version DFC and voice mail feature are tested. The FDDP version DFC architecture is tested using some screen snap shot pictures and illustrations in appendix A to provide easily understandable visual presentation showing the results of the FDDP version DFC architecture. The voice mail feature is tested by tracing the messages, sent or received by the feature. There are two possible ways to trace the messages.

One is to use the monitoring system in the ECLIPSE. All messages and events for administration and signaling, their sources and destinations, the sending ports and receiving ports and their ID, are stored in the MonitorManagerLog file. However, the file is extremely large (about 20-40 pages depending on each of the usages), and information in the file is very complicated. All the IDs are meaningless digits and letters, and the messages and events from all the objects in the ECLIPSE are mixed together. It is extremely difficult to read. Significant amount of editing job has to be done to pick up useful information from the large file. This makes message sequence tracing unreliable.

The other is to add print sentences in all classes of the VM system developed in this thesis and print them to log files. With this way, each log file is much shorter and easy to read. However, because we do not have code for Mbox and Magaco, some of the messages appeared on the sequence diagrams may not appeared on the trace file since we cannot add the print sentences to that part of the code. In this thesis, some of the most fundamental, frequent used and tested components developed by AT&T such as CallerTeardownFSM, and CalleeTeardownFSM, are reused as building parts of the VM system. These components interact directly with the MBox and Magaco. The parts (or objects) developed in this thesis do not involve in closing of the media channel, but involve in closing of the signal channel in some cases. In ECLIPSE, a router and all objects running on it run together, so the trace file records all the messages sent and received by the all these objects. The trace files for different scenarios are listed in the appendix B. The messages related to the voice mail system are marked with the sequence number in order to compare with the sequence diagrams. A brief explanation is also added to the trace files. The trace files for all the scenarios match the sequence diagrams very well except a few untraceable messages for closing the media channel. As an example, the trace file for leaving message on busy is discussed here.

6.5.1 Test for leaving message on busy

Figure 6.19 is a session cut from the trace file for leaving message on busy. The trace file matches the sequence diagram for leaving message on busy in figure 6.3 very well.

VoiceMailBoxFSM boxPort receives Setup a1
VoiceMailBoxFSM callee Port sends Upack a2
VoiceMailBoxFSM caller Port sends Setup a3
VoiceMailBoxFSM caller Port receives Upack a4
VoiceMailBoxFSM callee Port sends VoiceMailCenter Address msg a5
CalleeTeardownFSM port sends out Teardown a6
CalleeTeardownFSM port receives Teardown a7
CalleeTeardownFSM port sends out Downack a7'
CalleeTeardownFSM port receives Downack a6'
Lazy cleanup of precedence change
VocieMailCenterBoxFSM box port receives Setup b1
VocieMailCenterBoxFSM dual port sends Upack b2
AnswerFSM answerPort sends Avail b3
AnswerFSM answerPort receives Open b4
MediaAnswerFSM answerSPT sends OpenChan b5
MediaAnswerFSM answerSPT receives OpenChanAck b6
MediaAnswerFSM answerSPT sends OAck b7
VoiceMailCenterLineFSM internTerm receives Ready b8
VoiceMailCenterLineFSM internTerm sends Wait b9
MediaAnswerFSM answerSPT sends Accept b10
LeaveRetrieveMsgFSM sends greetings b11
LeaveRetrieveMsgFSM dual port receives LeaveMsg b12
LeaveRetrieveMsgFSM sends Beep b13

LeaveRetrieveMsgFSM receives MediaMsgs b14

VoiceMailCenterLineFSM dual port receives Teardown b15

Figure 6.20. Trace file for leaving message on busy.

6.6. Summary

In this chapter, the voice mail system is developed. The voice mail system consists of a voice mail feature and a voice mail center. The voice mail feature is deployed both at the end user device and inside the network in the FDDP version of the DFC architecture. The voice mail feature and voice mail center are designed and implemented and tested. The snap shot picture and the trace files prove that FDDP version DFC architecture and VM system function well.

Chapter 7 Discussion: Some Important Issues in IP Telephony

IP telephony has many advantages over traditional circuit switched telephony, such as low cost, highly efficient use of bandwidth and potential for advanced services. It is believed that the next generation of telecommunication will be IP-based. A number of IP telephony architectures, including H.323, SIP, DFC and their derivatives [27-31], have been proposed. However, the IP telephony architectures are far from mature, and all of them are evolving. To be a successful architecture, several important issues such as feature interaction, deployment, performance, evolvability and security have to be considered. Of course, there are other important issues such as the quality of voice, audio and video synchronization, but they are beyond the scope of the thesis and will not be discussed.

7.1 Feature interaction

Feature interaction has been a major problem in the traditional circuit switched network. It will be much worse in the IP telephony network since intelligent end user devices, multimedia transmission and mobility will bring a large number of new services to the IP telephony [6].

In the H.323 protocol, there are no guidelines on the implementation of features, there are only some supplementary service protocols specifying the functions of each service. Since there are no service architecture and design guidelines for the features in H.323, the service architecture is left open to service developers, and it is very difficult to develop a generic strategy to solve the feature interaction problem.

The SIP protocol and the Call Processing Language (CPL) [31] can be viewed as a service architecture of IP telephony. However, there is no clear guideline for implementing features, assembling features and preventing bad feature interactions.

The DFC is designed for feature modularity, structured feature composition, analysis of feature interactions, and separation of service and transmission layers. The components of the DFC are modularized as boxes. It provides very concrete guidelines for feature implementation. A usage is a dynamic assembly of the boxes and featureless internal calls. The order of the features in a usage follows the precedence rule that is the only constraint for the assembly of feature boxes. Therefore the DFC architecture makes feature composition and feature interaction analysis much easier. The following is an example showing how to avoid bad feature interaction between Call Waiting (CW) and Call Forward on Busy (CFB). Figure 7.1 shows two usages using these two features:

Figure 7.1. Usages using CW and CFB.

Both CW and CFB are used to handle the busy signal sent back from the callee. In usage (a), if CFB is assigned higher priority to react to the busy signal from the callee, it resides closer to the callee b. The CFB will absorb the busy signal from the callee, if callee is already engaged in another call, and the CFB will forward the call to the address c. In this case, an unexpected result happens: the CW feature will never receive a busy signal and have a chance to function. This is a bad feature interaction.

It is easy to solve this feature interaction problem in the DFC. We can simply assign higher priority to the CW, thus deploying it closer to the callee b as shown in

usage (b). If the CW receives a busy signal from callee b, it will send a call waiting text indication to callee b. Callee b can accept this call by pressing flash, and the call will then be connected. If callee b is engaged in an important call, he may not want to be interrupted by the incoming call from LIa, and he can ignore the text indication. After 20 seconds, CW will pass the busy signal to CFB. The CFB will then forward the call to callee c.

Here is another example of possible feature interaction. If we replace SMB in Figure 3.2 with outgoing call screen (OCS), we will get the following usage diagram.

Figure 7.2. Possible feature interaction between OCS and CFB.

The basic scenario is that user A subscribes OCS in its source zone and user B subscribes CFB in the target zone. User C is on user A's OCS list. When user A calls user B, if user B is busy, the call might be forwarded to user C causing bad feature

interaction since OCS should not allow user A to call user C. Whether the call is forward to user C depends on how CFB is implemented. If the CFB receives a busy signal from user B, it forwards the call to the user C immediately, then a bad feature interaction occurs. If the CFB receives a busy signal from user B, it first sends a message to user A to ask if it is allowed to forward the call to user C. When the OCS receives the message, the OCS checks if user C is on the OCS list. If user C is on the OCS list, OCS will send back a “No” message to CFB, CFB will not forward the call to user C. If user C is not on the OCS list, OCS will send back a “Yes” message to CFB, then CFB will forward the call to user C, then the bad feature interaction problem is avoided.

The FDDP version of the DFC inherits all the merits of the DFC, including ease of feature composition and feature interaction analysis.

7.2 Deployment of Features

As there are no formal service architectures in H.323 from ITU, many researchers and IP telephony vendors have developed their own service architectures [14, 16-18]. Therefore service deployment is very dependent on each of the researchers and vendors. Some services (features) are deployed in the end user devices, while some are deployed in the gatekeepers. The conference feature is deployed on the MCU inside the network

In SIP, as the end user devices become more and more powerful, there is a tendency to deploy features in the end user devices where possible. However, some features by nature have to be deployed inside the network, such as Call Forward on No Answer (CFNA) [32].

In the original DFC, all the features are deployed in the router inside the network. The end user device is connected through the LI interface box. In the FDDP version of the DFC, features are deployed, based on their nature, either in the end user's devices or inside the network or both. Here are some examples to give a clearer explanation of our deployment scheme.

The Incoming Call Screen (ICS) is deployed in the end user device. The function of the Incoming Call Screen is to screen out calls from certain addresses. If the end user device is connected to the network, when the call setup signal arrives at the end user device, the ICS will attract the address field from the setup message and check if the address is on the unwelcome list. If yes, the ICS will tear down the call; if not, ICS will let that call pass through. If the end user device is not connected to the network, then no calls at all, including those from unwelcome addresses, will reach the user.

Call Forward No Answer is better deployed inside the network. If it is deployed in the end user device, and the end user device is not connected to the network, this feature will not function and the call will not be forwarded to the other

address. If this feature is deployed inside the network, no matter whether the end user device is connected to the network or not, if the feature does not receive an answer signal within the time-out period (30 seconds), then the call will be forwarded to the address specified in the operational data for the feature.

Voice Mail is a feature that we would like to deploy both in the end user device and inside the network. With this deployment scheme, when the end user device is connected to the network, the voice mail in the end user device will record the messages and store the messages. Storing messages in the end user device saves storage space inside the network and is safer than storing the messages inside the network, since the end user device is not normally a target for hackers. When the end user device is not connected to the network, the voice mail inside the network will spring into action as a backup. In the FDDP version of the DFC, the call will be directed to the callee's mailbox in the Voice Mail Center, where the message will be recorded and stored.

7.3 Performance and Scalability

The modularized feature structure and pipeline fashion in signal processing in the DFC architecture introduce quite an amount of performance overhead. However, the modularized feature structure and pipeline fashion in signal processing bring many advantages, such as structured feature composition and ease of feature interaction analysis, to the DFC.

A performance study of the original DFC shows that the response time of each call setup process increases dramatically with the increase in the total number of subscribed features residing inside the network, once the number of features has passed a certain critical number [15]. As each router supports a large number of users and each user subscribes to some features, the total number of features running on each router will be very large. Therefore a reduction in the number of features running on the routers inside the network is beneficial for the performance of the system. In the FDDP version of the DFC, a significant number of features can be deployed in the end user devices, leaving fewer features running on the routers inside the network, so the performance and scalability of the IP telephony system will be improved.

7.4 Security

Security is becoming more and more important in Internet applications. The most widely used method of protecting a local area network (LAN), a server or personal computer (PC) connected to the Internet is to set up a firewall at the boundary. The firewall is responsible for restricting what information is communicated between the Internet and the LAN, the server or the PC.

In IP telephony, security is also an important issue. Both the IP telephony system and its users have to be protected by firewalls from malicious attacks. As components in the IP telephony system, such as the provisioning server and routers, have static IP addresses, once hackers find the address, they can attack again and

again. Currently most Internet users do not have a static IP address. Each user is assigned a new dynamic IP address each time they connect to the Internet; therefore it is unlikely that users would be attacked again by the same hacker. For broadband Internet users who do often have a static IP address, hackers can attack the same computer again. However the motivation to attack an individual personal computer is much lower than to attack the servers of big corporations and financial institutions. The possibility that IP telephony systems will be attacked is much higher than for users.

To maintain a user's privacy and security with regard to messages, in the FDDP version of the DFC, a VM feature is deployed in the end user device. A VM feature is also deployed inside the IP telephony network as a backup to prevent losing important messages, in case the end user device is not connected to the Internet.

7.5 Evolvability

IP telephony is a new technology, evolving very rapidly. More and more new services are being added to it. Providing advanced services to customers is critical for IP telephony to replace traditional telephony eventually. To be a successful service architecture, it must evolve well with the addition of those services.

In H.323, there is no official service architecture. It is left to IP telephony software companies or researchers to develop their own service architectures.

Therefore the evolvability of the service architectures will depend on the individual company or research group.

In SIP, even though the SIP protocol and Call Processing Language (CPL) can be viewed as a service architecture, this service architecture is still under development. It does not provide detailed guidelines on how to design, modularize or assemble features, or how to solve the feature interaction problem. When new features are added to the SIP system, bad feature interactions can occur. If bad feature interactions do take place, both the new and old features may have to be reprogrammed.

The DFC was designed for feature modularity, structured feature composition, analysis of feature interactions, and separation of service and transmission layers. Each feature in the DFC is independent of other features. Features in each usage are dynamically assembled according to their precedence. When a new feature needs to be added to the system, the system operator just needs to find which zone the feature belongs to and its priority, and then simply add the feature to a proper file directory without even having to recompile the system. Therefore the DFC has much better evolvability than H.323 and SIP. The FDDP version of the DFC inherits all these advantages.

Chapter 8 Conclusions and Future Work

With the rapid advance of computer hardware, there is a strong trend towards deploying IP telephony features in the end user devices. This thesis investigated the deployment of features in the DFC. With the addition of a local feature manager and Blind Call Transfer (BCT) feature, a modified DFC architecture with features deployed in dual places (FDDP version DFC) is proposed. A voice mail system is designed and implemented based on the new architecture. Some important issues concerning the service architectures of IP telephony are discussed. This chapter contains conclusions concerning the research carried out in this thesis, as well as possible future work.

8.1 Conclusions

DFC is a service architecture with modularized features, structured feature composition, ease of feature interaction analysis, and separation of service and transmission layers. However, in the original DFC, all the features are deployed on routers inside the network. This deployment scheme unnecessarily overloads the network and ignores the intelligence of the end user devices. A modified version of the DFC (FDDP version DFC) is proposed in this thesis.

In the FDDP version of the DFC, a local feature manager, an end user LI box proxy and a Blind Call Transfer feature are added. In the FDDP version DFC, each IP telephony user can subscribe to those features that will be deployed inside the IP telephony system network through the proxy of the user's LI box, which runs on the router inside the network, and subscribe to those features that will be deployed at the end user device through the user's own LI box, which runs in the end user device.

Due to the fact that at present most Internet users only have a dynamic IP address when they are connected to the Internet service provider, the deployment of the features has to be based on the nature of the features. For service reliability, some features, such as Call Forward No Answer, should be deployed inside the network; some features, such as Incoming Call Screen, are better deployed in the end user devices; and some features, such as Voice Mail, have to be deployed both in the end user device and inside the network.

The FDDP version of the DFC allows the end user to load telephony features from the central server and run features originally residing inside the network in the end user device, without any change. In the original DFC, all the features run inside the network. When the number of the features is larger than the critical value, the performance decreases dramatically. In the FDDP version of the DFC, a significant number of the features can be deployed in the end user device, and therefore the number of the features residing inside the network is reduced significantly. As a result, the FDDP version of the DFC is expected to have better overall performance.

The deployment of features such as VM in the end user device also improves security.

Both the modularized features and pipeline fashion signal transport in the original DFC introduce overhead to the performance of the system. The FDDP version of the DFC also inherits these defects.

8.2 Future Work

The current research shows that none of H.323, SIP and DFC are perfect. Among them, the DFC is the best service architecture in terms of modularity of features, structured feature composition and ease of feature interaction analysis. However, the performance overhead with DFC is obvious. The IP telephony architecture is still wide open. The development of new architectures with good performance besides modularized features, structured feature composition and ease of feature interaction will be very interesting.

Modification of the present DFC implementation to improve the performance of the DFC is possible and practical. Time-out is not implemented in the current version of the DFC. Adding Time-out in the future version of DFC will make the system more robust. More research work on the implementation of the DFC is needed.

Developing more advanced features that involve multimedia and instant messaging will also be interesting and challenging.

As more and more features are invented and added to IP telephony, the feature interaction problem will become more serious. In the DFC, the precedence rule is the only constraint to controlling feature interaction. If the features are DFC features, they will be assembled according to the precedence role. Therefore the precedence of each feature, as it relates to that of the others, has to be identified. A great deal of investigation is needed to determine the desired precedence of feature boxes.

Applying DFC's ideas on feature construction, feature composition to H.323 and SIP and other software architectures may reduce the feature interaction problems in these architectures. How to add DFC ideas to other architectures smoothly needs more investigation.

Appendix A

FDDP version DFC Architecture

In this part, we mainly test the proxy of user's LI box and Blind Call Transfer feature. In this test, there are two routers associated to one Mbox. There are two LI boxes running on the router inside the network (router i). One is Eric's LI box and the other is the proxy of Andrew's LI box. There is one LI box running on the router at the end user device (router e).

Figure A.1. Snap shot showing starting to connection.

Figure 1. shows that Eric calls Andrew, the proxy, however, BCT transfer the call to andrew0 at the end user device. The status label on the andrew0 shows “erci is calling”.

Figure A.2. Snap shot showing the caller and callee connected.

Figure A.2 shows Eric's LI box and Andrew's real LI box are connected. It means that the signal and media channels between the two LI boxes are established.

Figure A.3. Snap shot showing media communication.

Figure A.3 shows the media communication performs well. All these three pictures show that the proxy and Blind Call Transfer fulfill the design.

Appendix B

Trace File for Leaving Message on Busy and Retrieving Message

Adding classpath element file:/C:/eclipsejun14/boxcode.jar

Part 1

Adding classpath element file:/C:/eclipsejun14/boxcode.jar

SDPDescriptor.parse: Key: v, Value: 0

SDPDescriptor.parse: Key: o, Value: unknown 0 0 IN IP4 127.0.0.1

SDPDescriptor.parse: Key: s, Value: unknown

SDPDescriptor.parse: Key: c, Value: IN IP4 127.0.0.1

SDPDescriptor.parse: Key: t, Value: 1028671300010 1028671300010

SDPDescriptor.parse: Key: m, Value: PhoneGUI 1204 RTP/AVP 0

SDPDescriptor.parse: Begin media parsing

SDPDescriptor.parse: Add one media

SDPDescriptor.parse: Key: v, Value: 0

SDPDescriptor.parse: Key: o, Value: unknown 0 0 IN IP4 127.0.0.1

SDPDescriptor.parse: Key: s, Value: unknown

SDPDescriptor.parse: Key: c, Value: IN IP4 127.0.0.1

SDPDescriptor.parse: Key: t, Value: 1028671304900 1028671304900

SDPDescriptor.parse: Key: m, Value: PhoneGUI 1214 RTP/AVP 0

Lazy cleanup of precedence change

This part is generated by the MBox and the router, and is not useful.

Adding classpath element file:/C:/eclipsejun14/boxcode.jar

Part 2

VoiceMailBoxFSM boxPort receives Setup

VoiceMailBoxFSM callee Port sends Upack

VoiceMailBoxFSM caller Port sends Setup

VoiceMailBoxFSM caller Port receives Upack

AnswerFSM answerPort sends Avail

TransparentActFSM port1

receivescom.att.eclipse.core.messages.status.AvailMessage

TransparentActFSM port2spt

sendscom.att.eclipse.core.messages.status.AvailMessage

Transparent2LinksActFSM port2 receives OpenMessage

Open2LinksFSM opener sends OpenChan

Open2LinksFSM opener receives OpenChanAck

Open2LinksFSM opener sends Open2Links

Open2LinksFSM opener receives Open2LinksAck

Open2LinksFSM openee sends Open

AnswerFSM answerPort receives Open

MediaAnswerFSM answerSPT sends OpenChan

MediaAnswerFSM answerSPT receives OpenChanAck

MediaAnswerFSM answerSPT sends OAck

Open2LinksFSM openee receives OAck

Open2LinksFSM opener sends OAck

TransparentActFSM port2

receivescom.att.eclipse.core.messages.status.ReadyMessage

TransparentActFSM port1spt

sendscom.att.eclipse.core.messages.status.ReadyMessage

TransparentActFSM port1

receivescom.att.eclipse.core.messages.status.WaitMessage

TransparentActFSM port2spt

sendscom.att.eclipse.core.messages.status.WaitMessage

MediaAnswerFSM answerSPT sends Accept

TransparentActFSM port1

receivescom.att.eclipse.core.messages.status.AcceptMessage

TransparentActFSM port2spt

sendscom.att.eclipse.core.messages.status.AcceptMessage

This part show user c successfully setup a call with user b.

VoiceMailBoxFSM boxPort receives Setup a1

VoiceMailBoxFSM callee Port sends Upack a2

VoiceMailBoxFSM caller Port sends Setup a3

VoiceMailBoxFSM caller Port receives Upnack a4

VoiceMailBoxFSM callee Port sends VoiceMailCenter Address message a5

Part 3

CalleeTeardownFSM port sends out Teardown a6
 CalleeTeardownFSM port receives Teardown a7
 CalleeTeardownFSM port sends out Downack a7'
 CalleeTeardownFSM port receives Downack a6'
 Lazy cleanup of precedence change
 VocieMailCenterBoxFSM box port receives Setup b1
 VocieMailCenterBoxFSM dual port sends Upack b2
 AnswerFSM answerPort sends Avail b3
 AnswerFSM answerPort receives Open b4
 MediaAnswerFSM answerSPT sends OpenChan b5
 MediaAnswerFSM answerSPT receives OpenChanAck b6
 MediaAnswerFSM answerSPT sends OAck b7
 VoiceMailCenterLineFSM internTerm receives Ready b8
 VoiceMailCenterLineFSM internTerm sends Wait b9
 MediaAnswerFSM answerSPT sends Accept b10
 LeaveRetrieveMsgFSM sends greetings b11
 LeaveRetrieveMsgFSM dual port receives LeaveMsg b12
 LeaveRetrieveMsgFSM sends Beep b13
 LeaveRetrieveMsgFSM receives MediaMsgs b14
 VoiceMailCenterLineFSM dual port receives Teardown b15

This part matches the sequence diagram in figure 6.3. Teardown (a6) and
 Downack (a6') , Teardown (a7) andDownack(a7') are pairs.

Part 4

VocieMailCenterBoxFSM box port receives Setup e1

VocieMailCenterBoxFSM dual port sends Upack e2

AnswerFSM answerPort sends Avail e3

AnswerFSM answerPort receives Open e4

MediaAnswerFSM answerSPT sends OpenChan e5

MediaAnswerFSM answerSPT receives OpenChanAck e6

MediaAnswerFSM answerSPT sends OAck e7

VoiceMailCenterLineFSM internTerm receives Ready e8

VoiceMailCenterLineFSM internTerm sends Wait e9

MediaAnswerFSM answerSPT sends Accept e10

LeaveRetrieveMsgFSM sends greetings e11

LeaveRetrieveMsgFSM dual port receives RetrieveMsg e12

LeaveRetrieveMsgFSM sends VMInfo&query e13

LeaveRetrieveMsgFSM dual port receives RetrieveMsg e14

LeaveRetrieveMsgFSM sends MediaMsgs e15

LeaveRetrieveMsgFSM dual port sends teardown e16

This part of the file is for retrieving message. It matches the sequence diagram in figure 6.7 except untested 2 closing channel messages

Trace File for Leaving Message on No Answer

```
Adding classpath element file:/C:/eclipsejun14/boxcode.jar
VoiceMailBoxFSM boxPort receives Setup  c1
VoiceMailBoxFSM callee Port sends Upack  c2
VoiceMailBoxFSM caller Port sends Setup  c3
VoiceMailBoxFSM caller Port receives Upack  c4
/*AnswerFSM answerPort sends Avail // AswerFSM is a component of LIbox*/
TransparentActFSM port1 receives com.att.eclipse.core.messages.
status.AvailMessage  c5
TransparentActFSM port2spt sends com.att.eclipse.core.messages.
status.AvailMessage  c6
Transparent2LinksActFSM port2 receives OpenMessage  c7
Open2LinksFSM opener sends OpenChan  c8
Open2LinksFSM opener receives OpenChanAck  c9
Open2LinksFSM opener sends Open2Links  c10
Open2LinksFSM opener receives Open2LinksAck  c11
Open2LinksFSM openee sends Open c12
/* AnswerFSM answerPort receives Open
MediaAnswerFSM answerSPT sends OpenChan
MediaAnswerFSM answerSPT receives OpenChanAck
MediaAnswerFSM answerSPT sends OAck
```


MediaAnswerFSM is from LIBox/*

Open2LinksFSM openee receives OAck c13

Open2LinksFSM opener sends OAck c14

TransparentActFSM port2

receivescom.att.eclipse.core.messages.status.ReadyMessage c15

TransparentActFSM port1spt

sendscom.att.eclipse.core.messages.status.ReadyMessage c16

TransparentActFSM port1

receivescom.att.eclipse.core.messages.status.WaitMessage c17

TransparentActFSM port2spt

sendscom.att.eclipse.core.messages.status.WaitMessage c18

TransparentActFSM port2 spt sends VoiceMailCenter Address message c19

VoiceMailBoxFSM callee Port receives Teardown message c20

CallerTeardownFSM port sends out Teardown c21

CalleeTeardownFSM port sends out Downack c22

This file is for the first part of leaving a message on no answer case in figure 6.4. It matches the sequence diagram except the untested closing channel part in figure 6.4.

Trace File for Leaving Message on Disconnected

```
Adding classpath element file:/C:/eclipsejun14/boxcode.jar
VoiceMailBoxFSM boxPort receives Setup    d1
VoiceMailBoxFSM callee Port sends Upack    d2
VoiceMailBoxFSM caller Port sends Setup    d3
Adding classpath element file:/C:/eclipsejun14/boxcode.jar
VoiceMailBoxFSM caller Port receives Upack    d4
/*CalleeTeardownFSM port sends out Teardown //from BCT */
VoiceMailBoxFSM caller Port receives Teardown message    d5
VoiceMailBoxFSM callee Port sends VoiceMailCenter Address message    d6
CallerTeardownFSM port sends out Downack    d7
CalleeTeardownFSM port sends out Teardown    d8
CalleeTeardownFSM port receives Teardown    d9
CalleeTeardownFSM port sends out Downack d8'
CalleeTeardownFSM port receives Downack d9'
```

This trace file matches the sequence diagram in figure 6.5.

References

1. Mahbub Hassan and Alfandika Nayandoro, "Internet Telephony: Services, Technical Challenges, and Products", IEE Communications Magazine, pp96, April 2000.
2. Michael Jackson and Pamela Zave, "Distributed feature composition: A virtual architecture for telecommunication services", IEEE Transactions on Software Engineering, XXIV(10), pp831, October 1998.
3. Ismail Dalgic and Hanlin Fang, "Comparison of H.323 and SIP for IP Telephony Signalling", Proc. of Photonics East, Boston, Massachusetts, Sept. 1999.
4. Dirk O. Keck and Paul J. Kuehn, "The Feature and Service Interaction Problem in Telecommunications Systems: A Survey", IEEE Transactions on Software Engineering, vol.24, no.10, 1998.
5. Jane Cameron and F. Joe Lin, "Feature Interactions in the New World", in "Feature Interactions in Telecommunications and Software Systems V", eds K. Kimbler and L.G. Bouma, IOS Press, pp3, 1998.
6. Pamela Zave and Michael Jackson, "New Feature Interactions in Mobile and Multimedia Telecommunication Services", in "Feature Interactions in Telecommunications and Software Systems VI", eds M. Calder and E. Magil, IOS Press, pp51, 2000.
7. Jonathan Lennox and Henning Schulzrinne, "Feature Interaction in Internet Telephony", in "Feature Interactions in Telecommunications and Software Systems VI", eds M. Calder and E. Magil, IOS Press, 2000.
8. Pamela Zave and Michael Jackson, "DFC Modifications I: Routing Extensions",

- AT&T Technical Memorandum HA1640000-000128-4TM.
9. Pamela Zave and Michael Jackson, "DFC Modifications II: Protocol Extension",
AT&T Technical Memorandum HA1640000-991119-16TM.
 10. Pamela Zave, Michael Jackson and Eric Cheung, "Media implementation in
ECLIPSE I: Principles of signaling/media separation", AT&T Technical
Memorandum HA1640000-000808-9TM.
 11. Christos A. Polyzois, K. Hal Purdy, Ping-Fai Yang, David Shrader, Henry
Sinreich, François Menard and Henning Schulzrinne, "From POTS to PANS:
A Commentary on the Evolution to Internet Telephony", IEEE Network, pp58,
May/Jun, 1999.
 12. Jonathan Rosenberg, Jonathan Lennox and Henning Schulzrinne, "Programming
Internet Telephony service", IEEE Network, pp42–49, May/June, 1999.
 13. D. Isenberg, "The Dawn of the Stupid Network," ACM Network, v.2, no. 1,
pp24, 1998.
 14. Daniele Rizzetto and Claudio Catania, "A Voice over IP Service Architecture for
Integrated Communications", pp42–49, May/June, 1999.
 15. Jun Feng, M.S. thesis to be finished, Department of Systems and Computer
Engineering, Carleton University.
 16. "IP Telephony: Packet-based multimedia communications systems", eds., Olivier
Hersent, David Gurle and Jean-Pierre Petit, Addison-Wesley, 2000.
 17. Bernard Pagurek and Tony White, "A Quick Evaluation of H.323/H.450, SCE
Technical Report, SCE 99-02, Department of Systems and Computer
Engineering, Carleton University.

18. M. Handley, H. Schulzrinne, E. Schooler and J. Rosenberg, "SIP: session initiation protocol,' IETF, RFC 2543, March, 1999.
19. Henning Schulzrinne, "The IETF Internet Telephony Architecture and Protocols", IEEE Network, pp18, May/Jun, 1999.
20. Henning Schulzrinne and Jonathan Rosenberg, "Signalling for Internet Telephony", in International Conference on Network Protocols (ICNP), Oct., 1998.
21. Henning Schulzrinne and Jonathan Rosenberg, "A Comparison of SIP and H.323 for Internet Telephony", Proc. NOSSDAV, July, 1998.
22. Mary Shaw and David Garlan, "Software Architectue: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
23. Greg Bond, Eric Cheung, Michael Jackson, Hal Prudy, Chris Ramming and Pamela Zave, "Investigations of A New Service Architecture for The Next Generation of Telecommunications", <http://www.research.att.com/~pamela/dfc.html>.
24. Gregory Bond, Andrew Forrest, Eric Cheung, Michael Jackson, Hal Purdy, Chris Ramming, and Pamela Zave, "DFC as the basis for ECLIPSE, an IP Communications software platform", IPTS 2000, pp19, Sept. 2000.
25. Michael Jackson and Pamela Zave, "The DFC Manual", May 2000, <http://www.research.att.com/~pamela/dfc.html>
26. Pamela Zave, "An Architecture for Three Challenging Features", [http://www .research.att.com/~pamela/dfc.html](http://www.research.att.com/~pamela/dfc.html).
27. Christian Huitema, Jane Cameron, Peteros Mouchtaris and Darek Smyk, "An

- Architecture for Residential Internet Telephony Service”, IEEE Internet Computing, pp73, May/June, 1999.
28. Pawan Goyal, Albert Greenberg, Charles R. Kalmanek, William T. Marshall, Partho Mishra, Doug Nortz and K. K. Ramakrishnan, “Integration of Call Signaling and Resource Managenet for IP telephony”, IEEE Internet Computing, pp44, May/June, 1999.
29. Roch H. Glitho, “Advanced Services Architectures for Internet Telephony: A critical Overview”, IEEE Network, vol.14, no.4, pp38, July/August 2000.
30. Jingrong Tang, Tony White and Bernard Pagurek, “Advanced Service Architecture for H.323 Internet Protocol Telephony”, MATA’2000.
31. Jonathan Lennox and Henning Schulzrinne, "Call Processing Language Framework and Requirement", RFC 2824, IETF, May 2000.
32. Xiaotao Wu and Henning Schulzrinne, "Where Should Services Reside in Internet Telephony Systems?", in Proceedings of the IP Telecom Services Workshop 2000, pp39-44, Atlanta, U.S.A, Sept. 2000.
33. Private Communication with Andrew Forrest of AT&T Research Lab.