INTERWORKING SIP AND H.323 FOR VOIP APPLICATIONS

Arshad Farhad
ABSTRACT

SIP and H.323 are currently the most prominent call processing and media control protocols for VoIP and multimedia applications. Since the introduction of the two standards in the mid 1990’s their review and extension to support new and improved services has been an ongoing activity. The interoperability of SIP and H.323 has attracted much attention of recent because it is envisaged that both protocols will continue to be deployed for VoIP and multimedia.

The main objective of this project therefore, is to develop an interworking solution that will enable hybrid SIP-H.323 communication. An in-depth study of both protocols carried out from the outset revealed the need to design a software protocol interworking function (PIWF) to translate from one protocol to the other. However, the complexity of the PIWF as well as its requirement to handle real-time concurrent communication necessitated the use of formal methodology, in particular UML and SDL, in its development. UML is used in the requirements specification, analysis and design stages, while the resulting design is modelled in SDL for verification. The SDL model is simulated and the outcome correlates with the expected behaviour thus validating the design. Indeed, the application of formal methodology has been found to be a very effective means of identifying and eliminating design flaws prior to implementation, and hence, will lead to high reliability and significant reduction in overall development cost and effort.

Keywords:- VoIP, SIP, H.323, Interworking, UML, SDL
TABLE OF CONTENTS

ABSTRACT ..................................................................................................................1

ACKNOWLEDGEMENT ...............................................................................................1

CHAPTER 1 INTRODUCTION .............................................................................1

CHAPTER 2 BACKGROUND THEORY .................................................................3

2.1 Historical evolution of VoIP signalling standards ..........................................3

2.2 Overview of H.323 .........................................................................................5

2.2.1 H.323 network architecture .................................................................5

2.2.2 H.323 protocol stack .........................................................................7

2.2.3 H.323 call signalling flows ................................................................8

2.2.4 FastConnect procedure ......................................................................9

2.3 SIP Overview ............................................................................................10

2.3.1 SIP functional entities .....................................................................12

2.3.2 SIP protocol stack ............................................................................13

2.3.3 SIP call signalling flows ..................................................................14

2.3.4 SIP message format .........................................................................15

2.4 Comparison between SIP and H.323 .........................................................16

2.5 SIP-H.323 interoperability issues ...............................................................17

2.5.1 Address translation .........................................................................17

2.5.2 Endpoint registration ......................................................................18

2.5.3 Call sequence mapping ..................................................................21

2.5.4 Session description ..........................................................................24

2.6 Overview of formal methods ....................................................................27

2.6.1 Unified Modelling Language (UML) ..................................................27

2.6.2 Specification and Description Language (SDL) .................................28

2.6.3 Relationship between UML and SDL ...............................................29

CHAPTER 3 PROJECT WORK .......................................................................31

3.1 SIP-H.323 interworking requirements .....................................................32
3.2 PIWF requirements

capture ................................................................. 33

3.3 Analysis phase ..................................................... 34

3.4 Design phase ....................................................... 36

3.4.1 PIWF architecture design ..................................... 36

3.4.2 PIWF functional design ....................................... 37

3.4.3 Object sequence diagrams ..................................... 40

3.5 State machine modelling ......................................... 44

3.6 PIWF modelling with SDL ....................................... 47

3.7 SDL model compilation and simulation ....................... 53

CHAPTER 4 RESULTS AND DESCRIPTION .......................... 56

4.1 Simulation results of SIP to H.323 call signalling ............... 56

4.2 Simulation results of H.323 to SIP call signalling ............... 58

4.3 PIWF verification .................................................. 60

CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS ............. 61

5.1 Conclusion .......................................................... 61

5.2 Recommendations ................................................ 62

REFERENCES ................................................................ 63

APPENDICES:

Appendix A. List of acronyms .............................................. 65

Appendix B. PIWF object state chart diagrams ...................... 66

Appendix C. SDL process diagrams ..................................... 68
Chapter 1

INTRODUCTION

Voice over IP (VoIP), which involves real-time voice communications over packet switched networks, is evolving into a powerful technology capable of changing the way voice applications are developed and delivered. Among real-time applications, VoIP is becoming a popular service in the Internet as well as in enterprise intranets [1]. It is generally accepted that IP will eventually become the dominant transport mechanism for all business information services—voice, data, video and multimedia. In fact, it is envisaged that in future, QoS-enabled all-IP core networks will provide both fixed and mobile services. VoIP technology eliminates the need for separate voice and data platforms thus leading to significant cost savings for small, medium and large enterprises. Also, service providers can deliver cheaper, integrated, value-added and flexible telephony services to customers, with VoIP.

Since the mid 1990s efforts have been intensified towards standardization of multimedia communications over packet switched networks as well as to achieve seamless integration with legacy telephone networks. This led to the emergence of signalling protocols for VoIP, the two most prominent of which are: SIP [2] developed by IETF and H.323 [3] defined by ITU-T. Both perform similar roles using different mechanisms and are hence incompatible.

H.323 is an umbrella standard that provides well-defined system architecture and implementation guidelines covering the entire call set-up, call control, audio/video codecs and the media transport protocols. SIP, on the other hand, is a text-based protocol designed to work hand-in-hand with other core Internet protocols like HTTP. Several functions in a SIP-based network rely on complementary protocols including IP. Currently, H.323 is the most widely used protocol for PC-based conferences while carriers providing IP telephony services are more inclined to SIP. Since both protocols are being deployed for VoIP and multimedia applications,
interworking between the two is essential to ensure full end-to-end connectivity. Apart from the benefit of enabling a larger interconnected customer base, interworking also eliminates the need for manufacturers to support both SIP and H.323 in their products thus reducing equipment complexity and cost.

Although previous work has been undertaken in SIP-H.323 interworking, the issue is still very much an ongoing effort since both protocols are very much under review with support for new features and advanced services being added in the drive to achieve the service quality and reliability of legacy telephone networks. This project ultimately aims to develop an interworking solution based on in-depth comparative study of SIP and H.323 and the various drafts and recommendations for interoperability issued by standard bodies.

The rest of the project report is organized as follows. Chapter 2 begins with a background on the historical evolution of H.323 and SIP, followed by a brief overview of their features and operation. A comparison of H.323 and SIP protocols is summarized next, followed by a discussion of the core issues involved in interworking them. Finally, an overview of the technical concepts, i.e. formal methods, underlying the project work is presented.

Chapter 3 covers the project work including the requirements capture, analysis, formal design, modelling and simulation of the interworking solution using Telelogic™ Tau UML and SDL suites. The results of the simulations are featured and analysed in chapter 4, while the concluding chapter, 5, presents the conclusion and suggestions for further work.
Chapter 2

BACKGROUND THEORY

2.1 Historical evolution of VoIP signalling standards

The definition of signalling protocols for VoIP began in the mid 1990’s as an ITU initiative. The development of multimedia standards was originally the responsibility of the ITU Study Group 15. The group was responsible for the ISDN H.320 [4] conferencing standard, but in May 1996, the development of H.323 was moved to Study Group 16 under the title “visual telephone systems and equipment for local area network which provide a non-guaranteed quality of service”. It was originally intended to be a standard for audiovisual conferencing over LANs. Following the approval of the H.323 version 1 in June 1996, interest in the protocol grew rapidly within the emerging Internet telephony industry thus making H.323 the first industry accepted standard for multimedia conferencing.

The founding of the VoIP forum which subsequently became part of the international multimedia teleconferencing consortium (IMTC), led to the wider acceptance of the H.323 version 1 with the standard being driven by the needs of the Internet telephony community. H.323 version 2 subsequently emerged under the title “packet based multimedia communications systems” (to reflect its use in VoIP applications) and was approved in January 1998. Subsequent versions 3 and 4 were approved in September 1999 and November 2000 respectively.

Version 1 identifies the main functional entities in an H.323 system: terminals, Gatekeepers, Multipoint Control Units (MCUs) and Gateways. The standard also references the use of H.323 on LANs employing IEEE 802.3 and IEEE 802.5, IEEE 802.10 and FDDI protocols. Version 2 addressed some of the deficiencies of version 1 and added new functionality to the H.245 [5] and H.225 [6] while introducing new protocols. The additions were aimed at increasing the speed
and the scalability of the operations, as well as to provide enhanced services like improved security, supplementary services such as call transfer, call diversion, call identifier, progress messages etc. H.323 version 1 only addressed basic call handling since it was developed in a relatively short time-scale. H.450.x framework was defined in version 2 to support supplementary services. H.450.1 [7] is the general functional protocol (i.e. the underlying framework), H.450.2 [8] defined call transfer, while H.450.3 [9] defined call diversion.

H.323 Version 3 was more of a maintenance release correcting the deficiencies of H.323 version 2. A few additions were made such as new supplementary services extensions to H.450.x as well as support for fax (T.38 fax) transmission. H.323 Version 4 principally addresses approaches to managing an increasingly complex protocol, support for interworking H.323 with heterogeneous circuit switched network environments and enabling a richer suite of applications.

The ratification of version 4 of H.323 in November 2000 coincided with the completion of a four-year period of study. Version 4 of H.323 can be used to provide a robust VoIP multimedia telecommunications solution. The next study period for H.323 study group SG16 that began in 2001 is due to for completion in 2004. The priority areas under study are multimedia related to mobility (especially when applied in mobile networks), the Internet, and the convergence between conversational multimedia and the interactive broadcasting. It includes the creation of a project- Mediacom 2004- for the development and harmonisation of multimedia standards across other ITU-T and ITU-R study groups and across other related forums [10].

SIP, on the other hand, has its origins in the late 1996 as a component of the Mbone [11] set of utilities and protocols. It was developed from existing Internet technologies like MIME, HTTP, etc., already standardized within the IETF. This resulted in a protocol that integrated easily with other IP applications such as email and the web. The development of SIP did not progress much until it became apparent that it was capable of supporting VoIP applications. This led to intensifying efforts towards rapid development of the protocol in late 1998. In 1999, SIP was
specified by the IETF Multimedia Session Control Working Group (MMUSIC WG) as a proposed standard in IETF RFC 2543. It was however updated by the SIP WG in 2002 and published in IETF RFC 3261. SIP provides advanced signalling and control functionality for a large range of multimedia communications.

In addition to baseline SIP according to RFC 3261, several RFCs and a large number of Internet-Drafts complete or enhance the architecture regarding SIP applications, supplementary services, feature programming, conference, routing, preference management, and interworking issues. In this way, the IETF SIP IP-telephony standardization is still a "work in progress" and has not yet reached a final state. The baseline RFC and other RFCs and the Internet-Drafts are under continuous refinement by several working groups (WGs) [12]. Recently, SIP has begun to gain tremendous market acceptance. Its scalability, extensibility, and most importantly- flexibility, appealed to service providers and vendors who had needs that a vertically integrated protocol such as H.323 could not address.

2.2 Overview of H.323

H.323, which is currently the most widely deployed call processing protocol in the VoIP industry, is an ITU-T recommendation that defines system aspects for multimedia communications over packet switched networks. It was originally designed to support multimedia services over a LAN with no QoS guarantees [13]. H.323 is an umbrella standard that provides well-defined system architecture and implementation guidelines covering the entire call set-up (defined in H.225.0), call control (defined in H.245), audio and video codecs (e.g. G.711 for audio and H.261 for video) and the media transport protocols (RTP and RTCP).

2.2.1 H.323 network architecture

A number of functional elements are required for multimedia conferencing using the H.323 protocol. These include: H.323 endpoints or terminals, Gatekeepers, Multipoint Control Units (MCUs) and Gateways. A H.323 endpoint may set up a direct two-way, point-to-point,
audio, video or data conferencing session with another H.323 endpoint. A Gatekeeper is an entity that provides services such as address translation, RAS (registration, admission and status) control, call redirection, resource management and other control functions to H.323 clients. The presence of a Gatekeeper defines a zone, and each client within the zone registers with the Gatekeeper through which it establishes sessions with other clients.

A multipoint control unit (MCU) is a functional element of H.323 networks that support multipoint conferencing between three or more terminals and gateways. It consists of a mandatory multipoint controller (MC) and an optional multipoint processor (MP). The MC supports the negotiation of capabilities with all terminals in order to ensure a common level of communications during the multipoint conferencing. MC also controls resources in a multicast operation. The MP, under the control of the MC, provides media switching and mixing functionality. It is essentially the central processor of the voice, video and data streams for a multipoint conference. H.323 can support centralized, decentralized and mixed multipoint conferences. In the centralized mode, the MCU establishes media channels with all participating endpoints. The MC manages the conference while the MP receives, processes and sends the media streams to the endpoints. In the decentralized mode the participating H.323 terminals communicate via their own MCs directly without an MCU, while their individual MPs process the multicast media streams. Figure 1 illustrates the architecture of an H.323-based network.

The Gateway is an optional entity, which provides interoperability with other non-H.323 clients attached to other networks. Gateways are used to link LAN-based H.323 endpoints to endpoints in PSTN, ISDN, and other external networks. The gateway performs protocol translation, conversion of media formats and transfer of information between the dissimilar clients. A gateway can also operate as an H.323 multipoint control unit.
2.2.2 H.323 protocol stack

H.323 is not a single protocol but a comprehensive framework that cites the use of several protocols. Figure 2 shows the current H.323 protocol stack. The standards specified for audio applications include G.711 with other G series (G.722, G.723, G.728, G.729) as options. H.261 and H.263 are the video standards while the T series support data applications. H.323 also references other lower level standards, some of which existed prior to the creation of the first version of H.323, such as IETF RTP specification, and some of which had to be developed in parallel with H.323, H.225.0 being an example of the latter [10]. H.225.0 is the recommendation for ‘call signalling protocols and media stream packetization for packet-based multimedia communication systems’ for H.323. Various control, maintenance and signalling operations are
provided by H.245 and Q.931. Audio and video packet streams are encapsulated into the Real Time Protocol (RTP) and then transported on a UDP socket pair between the two communicating endpoints. Real Time Control Protocol (RTCP) is used to assess the quality of connections and sessions as well as to provide feedback information between the endpoints. RTP [14] and RTCP are both IETF specifications. Either TCP or UDP could be used to transport data and control packets.

<table>
<thead>
<tr>
<th>Audio Applications</th>
<th>Video Applications</th>
<th>Data Applications</th>
<th>Terminal Control and Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.711</td>
<td></td>
<td>T.120</td>
<td>RAS Control H.225</td>
</tr>
<tr>
<td>G.722</td>
<td>H.261</td>
<td></td>
<td>H.245 Control</td>
</tr>
<tr>
<td>G.723</td>
<td>H.263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G.728</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G.729</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RTP/RTCP

UDP

UDP or TCP

Network layer (IP)

Link layer (IEEE 802.3, 802.5, etc)

Physical layer (IEEE 802.3, 802.5, etc)

Figure 2. The H.323 protocol stack.

2.2.3 H.323 call signalling flows

This section gives a brief description of the H.323 call set-up procedure between communicating parties for audio/video conference. Five main distinct phases can be identified for setting up and managing a call with H.323 [13]; they are:

- Phase 1: - call set-up (after Gatekeeper discovery and registration);
- Phase 2: - initial communication and capability exchange;
- Phase 3: - establishment of audiovisual communication;
- Phase 4: call services;
- Phase 5: - call termination.
Several messages will be exchanged during each phase, the number of which varies depending on the application, number of clients, the configuration and type of conference etc. Initially, endpoints undergo a registration operation with a Gatekeeper, by exchanging H.225 RAS messages. Endpoints can be optionally configured for automatic Gatekeeper discovery otherwise the registration process can be invoked manually. The call set-up phase is used to set up a connection for an end-to-end call between endpoints. The procedure begins with the originating endpoint issuing an admission request (ARQ) to the Gatekeeper in the zone. After the endpoint receives a confirmation message (ACF) from the Gatekeeper, the call set-up procedure continues with a SETUP and CONNECTION message exchange.

The capability exchange is executed next using H.245 terminalCapabilitySet messages, in order to ensure the correct exchange of media streams. Information regarding codec types and bit rates are exchanged during this procedure. Next, one or more H.245 logical channels is opened to carry the media traffic. At the end of the session, termination operation takes place closing all existing logical channels. H.245 is used by the endpoints to terminate a call, but if a Gatekeeper is involved, H.225 RAS is invoked to release the bandwidth. Figure 3 gives a summary of the entire call signalling operation showing the protocols involved in each phase.

### 2.2.4 FastConnect procedure

It is worth noting that in the standard H.323 procedure, a session is established before negotiating the features and function of that session (and finally opening a logical channel), and thus, call set-up can take a long time. An optional procedure, which reduces the signalling overhead by exchanging fewer messages, is the FastConnection Procedure defined in the version 2 of H.323. This is initiated by including a FastStart element in the SETUP message. The FastStart element incorporates the media capability of the originating endpoint in OpenLogicalChannel and thus allowing media stream transmission after one round-trip message exchange (instead of the three used in the standard procedure).
2.3 SIP overview

SIP is an application layer control protocol developed by IETF for initiating, negotiating, modifying and terminating sessions between communicating entities. A SIP session could be a multimedia session comprising voice, video clips, and HTML pages etc., or a simple point-to-point telephone call. SIP is not a vertically integrated communications system but rather a component that can be used with other IETF protocols to build complete multimedia architecture. Typically, these architectures will include protocols such as the Real-time Transport Protocol (RTP) for transporting real-time data and providing QoS feedback, the Real-Time streaming protocol (RTSP) for controlling delivery of streaming media, the Media Gateway Control Protocol (MEGACO) for controlling gateways to the Public Switched Telephone Network (PSTN), and the Session Description Protocol (SDP) for describing multimedia sessions [2]. Therefore, SIP should be used in conjunction with other protocols in order to provide complete services to the users. However, the basic functionality and operation of SIP does not depend on any of these protocols.

SIP offers an easily implemented, powerful, control environment capable of scaling to very large networks due to its simple message request/response format. This, combined with its relative immaturity compared to H.323, encouraged its adoption in the access segment of third generation networks, since this affords the opportunity to incorporate any mobile-
specific elements that were subsequently identified [15]. SIP is potentially the protocol of choice for supporting sessions in a variety of applications ranging from VoIP to 3G multimedia sessions because of its attractive features, which include:

- **Simplicity**: SIP is a simple, lightweight, text-based protocol capable of setting up sessions with the exchange of relatively few messages compared to H.323. It is thus suited to run on battery-powered portable devices like pagers.

- **Extensibility/modularity**: The core SIP protocol can be extended to support additional functionality. This implies the emergence of varied implementations or versions. For example, a specific SIP mobile extension is being defined by 3GPP for multimedia session control in UMTS [16].

- **Scalability**: SIP servers could be stateless (i.e. memoryless) thus enabling scalability and robustness. Scalability is also achieved through the use of multicasting.

- **Generic session description**: the signalling of sessions is separated from the description of the session. SIP could thus use other protocols other than SDP [17] to initiate and control completely new type of sessions having their own description protocols.

- **Compatibility with other TCP/IP protocols**: SIP easily interoperates with traditional IP protocols since it was derived from HTTP (header fields, authentication mechanism), SMTP (address server location), and border gateway protocol (BGP) (for stateless loop detection), which are all existing TCP/IP protocols. Together with real-time streaming protocol (RTSP), for example, it could be used to offer voice mail services or to invite a server to play a movie during a multiparty conference.

- **Seamless web integration**: integrates well into the client/server world of the World Wide Web thus allowing the pervasive web to be used as a ready-made platform for the SIP operations. Also, since URI’s are based on DNS, SIP takes advantage of the current Internet naming and addressing architecture. DNS is used to correlate DNS names to IP addresses.
2.3.1 SIP functional entities

The entities required for a SIP session comprise two major components: the user agents and the network server. The user agent (UA), which interfaces with the service user, is an application running on the client terminal. The UA contains both a user agent client (UAC) and a user agent server (UAS). UAC initiates the session or call, which is responded to by the UAS of the receiving endpoint. Thus SIP can operate as a peer-to-peer while using the client-server model since UAC and UAS are present in a UA.

The network server is an application that accepts requests, services them, and returns responses to the requests. SIP servers are Redirect, Proxy or Registrar. A Proxy acts both as a server and client for the purpose of making requests on behalf of other clients. A Proxy interprets or may re-write a request message before forwarding it to other servers. A Redirect server unlike a Proxy does not initiate its own SIP requests and it does not accept calls. When it receives a request, it maps the addresses to one or more new addresses and returns it to the client. A Registrar is a server that accepts REGISTER requests. It is typically co-located with a Proxy or a Redirect server and may offer location services. Call set-up scenarios with Proxy and Redirect servers are depicted in Figure 4.

(a) SIP configuration with Proxy Server
2.3.2 SIP protocol stack

<table>
<thead>
<tr>
<th>Conference set-up and discovery</th>
<th>Conference course control</th>
<th>Audio/video</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDP</td>
<td>RSVP</td>
<td>RTP/RTCP</td>
</tr>
<tr>
<td>SAP</td>
<td>Distribution control</td>
<td></td>
</tr>
<tr>
<td>SIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMTP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP or TCP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP and IP multicast</td>
<td>UDP</td>
<td></td>
</tr>
<tr>
<td>Integrated services forwarding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. SIP in the IETF multimedia conferencing architectures.

- SIP can operate over TCP or UDP. When sent over TCP or UDP, multiple SIP transactions can be carried in a single TCP connection or a UDP datagram.
- SAP (Session Announcement Protocols) and SDP support the establishment of multi-party conferencing sessions. SAP defines the procedures for advertising conferencing solutions by periodically multicasting information about active sessions. SDP supports the description of multimedia sessions including specification of preferred media types and scheduling information. SDP and SAP combine to provide a means of advertising sessions so intended parties can join.
2.3.3 SIP call signalling flows

Figure 6. Call flow diagram showing simple message flows for media session establishment and tear down.

A simple call flow scenario between two end points is illustrated in Figure 6. The flow is more complicated for calls involving Proxies and/or Redirect servers.

1. The calling party sends an INVITE message to the called party. The INVITE message contains a session description that informs the called party what types of media the caller can accept and where it wishes the media data to be sent.

2. The call is accepted by sending 200:OK response.

3. The calling party acknowledges with an ACK.

4. Media streams are then exchanged.
5. Calling party initiates teardown sequence by invoking a BYE method, which in turn is acknowledged by a 200:OK response from the other party.

2.3.4 SIP Message format

SIP uses plain text messages based on the HTTP and SMTP syntax. Thus a SIP message requires a relatively little processing. A SIP message, an example of which is shown in Figure 7, is either a request from a client to a server or a response from a server to a client. SIP uses URI’s of the form: SIP:user@host.domain to identify the participants in a session.

```
INVITE SIP: userb@brad.ac.uk SIP/2.0
Via: SIP/2.0/UDP home.com:5060
From: someone <SIP:userA@home.com>
To: recipient <SIP: userB@there.com>
Call-ID: 12345600@home.com
Cseq: 1 INVITE
Contact: someone <SIP:userA@home.com>
Content-Type: application/sdp
Content-Length: 147

v= 0
o= UserA 2896708526 2896708526 IN IP4 home.com
s= Session SDP
c= IN IPS 100.101.102.103
t=0 0
m= audio 49170 RTP/AVP 0
a= rtpmap: 0 PCMU/8000
```

Figure 7. Typical SIP INVITE message structure.

2.4 Comparison between SIP and H.323

SIP and H.323 share some similarities, for example, both use RTP for media transport and run over IP but there are differences between the two that must be taken into account in the design of an interworking solution. Some of these differences are summarized in Table 1.
### Table 1. A comparison of the features of SIP and H.323 protocols.

<table>
<thead>
<tr>
<th>FEATURE</th>
<th>SIP</th>
<th>H.323</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admission control</td>
<td>No admission control mechanism to manage session membership.</td>
<td>RAS procedures defined in H.225 for conference membership control.</td>
</tr>
<tr>
<td>Session Description</td>
<td>Uses separate protocol, typically but not necessarily SDP for session description. Session description bundled within call set-up (INVITE) messages.</td>
<td>Session description via H.245 messages <em>(capability descriptor).</em> Session descriptions separated from and invoked after call set-up procedure.</td>
</tr>
<tr>
<td>Modularity</td>
<td>All-in-one module: combine call signalling and media channels</td>
<td>Separate modules for different functions: independent call signalling and media channel</td>
</tr>
<tr>
<td>Advertising</td>
<td>Session announcement via SAP (session announcement protocol)</td>
<td>Obtain conference information via LDAP (optional)</td>
</tr>
<tr>
<td>Capability negotiation</td>
<td>Non-flexible</td>
<td>Flexible</td>
</tr>
<tr>
<td>Address Format</td>
<td>SIP URI of the form SIP:<a href="mailto:Sam@Bradford.ac.uk">Sam@Bradford.ac.uk</a></td>
<td>ASN.1 <em>AliasAddress</em> containing several fields e.g. email-ID, h323-ID, etc.</td>
</tr>
</tbody>
</table>

### 2.5 SIP-H.323 interoperability issues

This section examines the core issues involved in SIP and H.323 interoperability. These issues, which include address translation, endpoint registration, call sequence mapping, and
session description, all arise from the comparison made in the previous section and are crucial to the development of an H.323-SIP interworking solution.

2.5.1 Address translation

SIP and H.323 employ dissimilar addressing formats. Both address formats are aliases expressed in human-readable form to identify endpoints in a VoIP network. During registration, the alias addresses are mapped to the IP addresses which are used to locate endpoints for the establishment of a session. The resolution of alias addresses to IP addresses during a call is typically the function of the registration servers (i.e. SIP Registrar and H.323 Gatekeeper).

As mentioned earlier, a typical SIP address is a URL of the form sip:user@host e.g. sip:Vikky@bradford.ac.uk. SIP terminals also support other URL types like “tel:” URLs for telephone numbers or H.323 URLs. These addresses are contained within the SIP message header. A SIP URI identifies a communications resource [2]. H.323, on the other hand, uses ASN.1 AliasAddress, which could assume many forms, including unstructured identifiers (h323-ID), E.164 telephone numbers, URLs, host names or IP address and email address (email-ID). Thus, ASN.1 AliasAddress contains several fields but local user names and host names appear to be the most popular forms [18].

SIP-H.323 interworking, therefore, needs to tackle address translation from one format to the other. Particularly, a consistent way of mapping a SIP URI to a H.323 AliasAddress and vice-versa is required. This is very important because the addresses form part of the messages, which are exchanged during registration and call set-up. Furthermore, since interoperability will entail translating messages from one protocol format to the other, equivalent valid addresses need to be generated during the translation.

To translate from a SIP to a H.323 address, the SIP URI could be copied into the h323-ID field of the AliasAddress, since it supports unstructured identifiers. Alternatively, the user and the host parts of the SIP URI could be used to generate an email identifier, which is copied into
the email-ID field. Also, Transport-ID parameter could be copied from host part of the SIP URI if the latter is given numerically. The E.164 field is extracted from the user part of the SIP address if it is marked as a telephone number.

Translating a H.323 AliasAddress to SIP address is more difficult since multiple representations need to be merged into a single SIP address. If the H.323 AliasAddress contains a url-ID with a SIP URI it is simply copied into the SIP message. Otherwise, if the h323-ID contains a local user name and a host name that can be parsed as a valid SIP address, e.g. user<sip:user@host>, it is copied. If the H.323 alias has an email-ID it is used in the SIP URI prefixed with “sip:”. If a transport-ID is present, it could be used to form the host and port portions of the SIP URI.

2.5.2 Endpoint registration

In a stand-alone SIP network, endpoints register with the Registrar, while the Gatekeeper is the entity responsible for registration of H.323 endpoints in an H.323-based network. An endpoint is an entity that can call and be called and from which the media such as voice, video or fax originates or terminates [19]. An endpoint could be a H.323 terminal, H.323 gateway, H.323 MCU or a SIP UA. A hybrid network of SIP and H.323 endpoints could have one of several different registration architectures depending on the service deployment scenario. Figure 8 depicts some of the possibilities.
Interworking SIP and H323 for VoIP applications

a) Basic configuration.

b) Call involving SIP Proxy

c) Call involving a H.323 Gatekeeper

d) Call involving both SIP Proxy and H.323 Gatekeeper

Figure 8. Architectures for SIP-H.323 interworking.

In all the architectures shown, the SIP-H323 interworking function is a logical entity that translates the signalling procedures. It can be co-located with a Gatekeeper or a Registrar to function as a combined entity within the network.

In the basic configuration shown in (a), there is no registration involved hence the SIP UA is expected to know the H.323 destination address and vice-versa. Whereas, in the
architecture depicted in (b), the SIP-H323 interworking function incorporates a Gatekeeper. It is thus able to register H.323 endpoints via the usual H.225 RAS procedures, and then possibly export the registration (after address translation) to the Registrar in the SIP network. The interworking function exports the registration by registering the H.323 entity with the Registrar together with its own network address and a valid SIP URI created from the H.323 AliasAddress. The Registrar in the SIP network now has registration entries for H.323 entities, which are seen as native to the SIP network and are thus visible to the SIP UAs (as SIP URIs). When a SIP UA places a call, the Proxy will now forward the request to the interworking function which will in turn contact the H.323 entity in question. The drawback of this approach is that the Registrar is burdened with the additional entries for non-native H.323 elements [18].

The architecture of (c) is similar of that of (b). This time the SIP endpoints are registered with the interworking function/Registrar entity, which then exports the registration (together with its own IP address) to the Gatekeeper in the H.323 network. Hence, the H.323 side sees the SIP endpoints as though they were native and can then easily establish a call or session with them. Again, the drawback is the additional entries from the SIP side that the Gatekeeper has to handle.

In (d), the architecture has the SIP-H.323 interworking function operating independent of a Registrar/Proxy or a Gatekeeper. However, unlike the scenario in (a) calls will involve both Proxy and Gatekeeper. The interworking entity is registered with the Proxy, which is configured to forward unresolved requests from a SIP endpoint to the interworking function. The interworking function then broadcasts a LRQ (location request) to the H.323 network and the appropriate Gatekeeper responds with a LCF (locations confirm) if it has an entry for the H.323 client. In the opposite direction, if a H.323 Gatekeeper cannot resolve a request from a registered H.323 endpoint, it broadcasts a LRQ, which the interworking function is tuned to receive. The interworking function then uses the SIP OPTIONS message (which does not set-up a session) to
the Proxy in order to find out if the intended recipient exists within the SIP network. If the query succeeds the Proxy replies with the IP address of the SIP endpoint (in a 200 OK message) and then the interworking function forwards this to the Gatekeeper through a LCF (location confirm) message.

One major drawback of the approach in (d) lies in the fact that the address translation by the interworking function in either direction may not always lead to an equivalent resolvable address that can be found in the foreign network. For this reason, configurations (c) and (b) may be favoured over (d). But, (b) has some advantage over (c) for the following reason. In scenario (b), if the SIP network has more than one Proxy/Registrar, then each one has to be co-located with an interworking facility for address resolution to be possible for all the SIP entities. While in scenario (c), even with multiple Gatekeepers in the H.323 network, only one needs to be equipped with an interworking facility for address resolution to work. This is because the gatekeeper with the interworking facility could broadcast a LRQ message to locate H.323 endpoints not registered with it. When it receives a LCF message, the interworking facility can then send a response to the SIP side.

2.5.3 Call sequence mapping

The establishment of a media session between two endpoints involves the exchange of a sequence of messages in both SIP and H.323. These messages embody vital information required for the successful set-up of the session and subsequent transfer of media packets. Specifically, the destination address (D), media transport address (T), and the media capability descriptions (M) (i.e the supported codecs) must be known to the communicating parties.

In SIP, all three parameters of the sender are contained in a single request message (INVITE) and the corresponding response (200 OK) holds all three pieces of information for the recipient. While in H.323 the information is spread across several stages of the protocol
exchange. This means that is not possible to map the call sequence on a one-to-one basis during protocol translation.

A solution to this is to use the H.323 FastConnect (described earlier in section 2.2.4) supported in version 2, but since H.323 must support backward compatibility, FastConnect is optional. In the FastConnect procedure, all the three parameters, D, T, and M are present in the SETUP message. Thus fewer messages are exchanged during protocol translation. A possible message flow for SIP to H.323 call sequence translation using fast connect is illustrated in Figure 9.

![Figure 9. SIP to H.323 call sequence mapping with a FastConnect procedure.](image)

Since FastConnect is optional, protocol translation from H.323 client that does not support FastConnect should proceed with standard multi-stage call establishment. The protocol translation could involve FastConnect when a call originates from the SIP side but must switch to multistage call establishment procedure if the recipient H.323 party is unable to support FastConnect.
The multi-stage H.323 call establishment procedure which spreads these three key parameters (D, T, and M) has further implications for the SIP-H.323 call sequences mapping during protocol translation. Consider the mapping shown in Figure 10; the call is from the SIP side and the INVITE message has D, T and M. The interworking facility uses the parameters for the Q.931 (call set-up) and H.245 (capability exchange) phases. The responses from the H.323 side are collected and forwarded to the SIP side (in the 200 OK) message. Thus SIP to H.323 call translation is quite straightforward even without FastConnect.
On the other hand, H.323 to SIP translation as shown in Figure 11, will involve sending an INVITE to the SIP side without any session description (i.e. the body of the message will be empty with no SDP parameters) upon receipt of a Q.931 SETUP. The SIP side sends a 200 OK response with its capability description. The parameters obtained from the 200 OK are used by the interworking facility for capability exchange with the H.323 side. After the H.245 procedures are completed, the interworking facility sends the capability description of the H.323 side to the SIP side via the final ACK message.

2.5.4 Session description

As mentioned earlier, a media session between two parties can only proceed if the D, M, and T parameters of both are available to each other. Together, these parameters make up the session description in a call. In a H.323 session or call, M is equivalent to the capability set. SIP is a stand-alone protocol whose function is to establish, control and teardown session. It
therefore relies on other protocols, such as SDP, for the session description. In Internet telephony applications, SDP is used exclusively but other protocols could be used as well. SDP messages are carried in the body of SIP requests and responses as depicted in Figure 12.

```
INVITE SIP: userb@brad.ac.uk SIP/2.0
Via: SIP/2.0/UDP home.com:5060
From: someone <SIP:userA@home.com>
To: recipient <SIP: userB@there.com>
Call-ID: 12345600@home.com
Cseq: 1 INVITE
Contact: someone <SIP:userA@home.com>
Content-Type: application/sdp
Content-Length: 147

v= 0
o= UserA 2896708526 2896708526 IN IP4 home.com
s= Session SDP
c= IN IPS 100.101.102.103
t=0 0
m= audio 49170 RTP/AVP 0
a= rtpmap: 0 PCMU/8000
```

**Figure 12. SDP message in body of SIP INVITE.**

H.323 on the other hand, uses H.245 *TerminalCapabilitySet* message to exchange media capabilities, M, while T is conveyed via the *openLogicalChannel*. SDP is simpler than H.245 and, in conjunction with SIP, it attempts to convey the same information as H.245 in a single round trip. SDP, however, is not as flexible as H.245 in being able to describe the set of codecs or indicating how they are to be used in a session [20]. The interworking facility would need to translate between SDP messages and H.245 messages (during the Call sequence mapping described in the last section) to enable seamless capability exchange. However, problems may arise due to the differences in the way the media descriptions are interpreted by both protocols.

With SDP, when a receive media capability of G.711 and G.723.1 codec is specified, it means that the sender can switch between the two algorithms without informing the receiver.
With H.245, on the other hand, the sender chooses an algorithm from the capability set of the receiver and explicitly opens a logical channel for that algorithm. If the sender wants to change to another algorithm it must close the logical channel and reopen it with a new algorithm. Alternatively, the receiver could use H.245 modeRequest to request the sender to use a different algorithm.

Two approaches have been suggested in [18] for tackling the difference in the capability exchange mechanisms. One is to have RTP/RTCP packets from SIP to H.323 be intercepted by interworking facility and initiate the required H.245 procedures once a change in the coding algorithm is detected. This approach scales poorly though, and places much burden on the interworking facility.

The second approach is to limit the media description sent to the SIP side to one algorithm per media or per H.323 alternative capability set. The algorithm/codec that will be selected for the call is derived by the interworking facility, from the intersection of the two capability sets by selecting one algorithm per alternative capability set. For instance, let the SIP capability set = \{[G.711, G.732.1, PCMU] [H.261]\} and the H.323 capability set be \{[G.711, PCMU, G.729] [H.261]\} \{[G.723.1] [H.263]\}. I.e. H.323 can support either one of G.711, PCMU, G.729 with H.261 or G.729 with H.263, while the SIP user can support either one of G.711, PCMU, and G.723.1 audio with the H.261 video codec. Note that the capability set in H.323 is fixed and cross capability is not allowed, for example G.711 cannot be used with H.263 video for the above capability set. The maximum intersection derived is \{[G.711, PCMU][H.261]\}\{G.723.1\}. The interworking facility may then select \{[G.711, PCMU, G.729] [H.261]\} set for the H.323 side and conveys only PCMU and H.261 to the SIP side.
2.6 Overview of Formal methods

Having examined the design issues of H.323-SIP interworking in the previous section, a brief discussion of the technical concepts utilized in the project to develop an interworking solution is presented next. As it became clear that the interworking would involve the design of a complex software system required to operate within a real-time, event-driven communication environment, while executing several operations concurrently, it was decided that formal development techniques will be adopted rather than an informal approach.

*The informal/traditional approach relies on simulation, testing and code inspection for error checking which makes it more difficult and time consuming to detect errors. Also, the detection of a fault could lead to the review of the entire design process thus increasing cost and delaying product release. Formal methods on the other hand, enforce the system requirements on a mathematical model whose correctness can be mathematically proved. Errors can be detected and the correct functioning of the system can be verified in the early stages of the development process.*

Formal methods help in producing highly reliable software and are mainly applied in software development for system specification, validation and verification, functional testing, rapid prototyping and performance testing [21]. UML and SDL are the formal techniques that were used during the project.

2.6.1 Unified Modelling Language (UML)

*UML (Unified Modelling Language) is a graphical modelling technique used for software engineering [22]. It is the most important graphical notation for object-oriented software systems, which has been standardized by the Object Management Group (OMG) since November 1997. UML specifications are collections of diagrams, each of which supports a specific phase of the development process by capturing a particular abstraction of the system behaviour (analysis of the requirements, architectural and functional design, HW/SW*
partitioning, etc.). Its primary purpose is to facilitate communication between programmers, software architects and clients. It achieves this by providing facilities for modelling each phase of the software development life cycle from requirement analysis to implementation.

UML has many different diagrams that allow many different views of a system to be modelled. For instance, Use Cases provide a functional view; Class and Object diagrams, a structural view; State Chart, Sequence and Collaboration diagrams, a dynamic view; and Deployment and Component diagrams; an implementation view. The major disadvantage of UML, however, is that it has no formal semantics, thus making it difficult to simulate UML models in order to test them for correctness. There are dozens of commercial and freeware tools for UML, ranging from simple graphical editors to complete CASE packages like Telelogic Tau UML suite.

2.6.2 Specification and Description Language (SDL)

Specification and description language (SDL) is a formal description language evolved and standardized between 1976 and 1992 by ITU-T [23], which is used to specify and describe the functional behaviour of software systems. SDL is a high-level general-purpose description language for event-driven, real-time and communicating systems; telecommunication systems and protocols are one of its main application fields [24]. SDL is effective and has a friendly graphical notation, and is thus in widespread use in the academic and industrial sectors. It is the formal specification language most exploited by telecommunication manufacturing companies worldwide, and a great deal of software has been produced on SDL-based platform. Furthermore, the official specification documents of standardization bodies like ETSI and 3GPP feature SDL diagrams.
The main advantage offered by SDL is the ability to simulate or execute models for verification and validation. It does allow for the detection of errors before implementation thus saving time and resources in the long run. Moreover, SDL diagrams have a hierarchy of detail from high-level system diagrams to low-level process diagrams. It does not, however, provide for requirement specification and neither does it have as many views of a system as given by UML. SDL2000, the recent version, includes some UML features to enrich SDL software architecture description capabilities.

Several SDL based software packages are available including JADE and SITE, which are freeware tools. There is presently very little CASE support for SDL; the most successful commercial products are Telelogic Tau SDL Suite (SDT), by Swedish Telelogic, and ObjectGeode (recently acquired by Telelogic).

2.6.3 Relationship between UML and SDL

Both UML and SDL are useful formal techniques applicable in software development life cycle (SDLC). Whilst it is possible to employ UML alone throughout the entire SDLC, especially for simple or non real-time systems, in some cases there is a need to use UML in conjunction with SDL, especially for complex, real-time communication systems. The combined approach helps to overcome the limitations of both techniques while at the same time allowing their strengths to be exploited.

UML provides a rich array of diagrams that can support the earlier stages of development such as requirements capture, analysis and all levels of design; but because of lack of formal semantics the UML models cannot be validated through simulation. SDL, on the other hand, describes a system’s functional behaviour and hence supports simulation/execution of models to allow for verification/validation. The top-to-bottom hierarchy of detail captured by SDL gives a comprehensive structural view of the system. It is
however not suitable for the early SDLC stages because it does not provide for requirements specification and analysis. Unlike UML (sequence charts), SDL also lacks the notation to illustrate the dynamic behaviour of a system.

The differences notwithstanding, aspects of UML can be mapped directly to SDL thus facilitating a smooth transition from UML to SDL during the later stages of the SDLC. For example UML classes could correspond to processes in SDL, while the UML state charts correspond to the state machines in the processes. Thus UML and SDL can be said to complement each other and could be used together in the development of complex software systems that require verification.
Chapter 3

PROJECT WORK

This chapter covers the work carried out during the project from requirements analysis through design to verification. The development of the interworking solution was influenced at all stages by the issues discussed in section 2.5. The requirements capture, analysis, and design stages were carried out using Tellogic Tau UML suite, which provided project management tools, graphical design environments and error checking facility. The design was then modelled in SDL using Telelogic SDL suite. The SDL model was compiled after extensive debugging. The compiled model was used to simulate the SIP-H323 interworking scenarios using the SDL simulation UI tool in order to validate the correctness of the design. The simulation revealed design flaws, which were then corrected in the UML design, the equivalent SDL model updated and recompiled and then the simulation was run again. This cycle continued (as illustrated in Figure 13) until the model behaviour matched the expected operational requirements. Only then was the verification of the design deemed complete leading to high confidence in the correctness of the design.
This project will refer to the interworking facility designed for the translation of SIP-H.323 protocols as the protocol interworking function (PIWF). It follows from the comparison of SIP and H.323 and the discussion of interworking issues that the PIWF should, at a minimum, meet the following functional requirements [19].

- Resolving H.323 and SIP addresses;
- Registering the H.323 and SIP endpoints with SIP Registrars and/or H.323 Gatekeepers;
- Mapping of the call setup and teardown sequences;
- Negotiating terminal capabilities;
- Opening and closing media channels;
- Mapping media coding algorithms ( codecs) for H.323 and SIP networks.

Whilst performing the above functions the PIWF should not alter or modify the normal message sequences or formats of either protocols, but maintain them as given in the recommendations (i.e. RFC 3261 and H.323 recommendation) such that neither the SIP UA nor the H.323 terminal is aware of the PIWF presence. Furthermore, the parameters and messages, which do not have a direct mapping on the other side, are to be generated by the PIWF with default parameters.

SIP and H.323 typically use RTP for media transport. Therefore, with the assumption that a common transport protocol is used, the PIWF should not be required to process media. Media packets are exchanged directly between the endpoints. If, however, a particular
session requires the PIWF to handle media, it should forward the media packets from one network to the other without modification using a media switching fabric (MSF) [19].

The design of the PIWF should evolve in a modular fashion, starting from the basic protocol translation functionality to the configuration incorporating a H.323 gatekeeper (shown in Figure 8.b). The decision to design the PIWF with a H.323 Gatekeeper is based on the advantage over the configuration incorporating the SIP Registrar that was highlighted in section 2.5.2.

3.2 PIWF Requirements capture

The design of the PIWF began with requirements capture using UML use case diagrams (UCD). The use case diagrams were used to model the system behaviour i.e. what the system will do. The UCD in Figure 14 shows, (in the ovals), the tasks performed by the PIWF while interacting with the actors, in this case the H.323 and SIP endpoints (depicted as a stick Figures). The PIWF will register the H.323 endpoint, process its admission request, deal with its call set-up request, handle the description of the session and the opening of a logical channel for media exchange, while also initiating a session with the desired SIP endpoint. Thus the UCD captures in a nutshell, the PIWF operational requirements.
3.3 Analysis phase

The analysis phase followed after the requirements capture with UML use case diagrams. It involved a more detailed description of the system and how the components of the system interact to fulfil the requirements. UML sequence diagrams (SD) were the tools used in this stage to describe the most important scenarios. The decision to use sequence diagrams over collaboration diagrams, which could have served the same purpose at this stage of the project, was because sequence diagrams were found to be less confusing and more readable while collaboration diagrams tend to easily get cluttered as the system becomes more complex.

The sequence diagrams depict how the system will perform each of the tasks specified in the Use case diagrams. It was particularly useful in modelling the dynamic behaviour of the PIWF system. The sequence diagrams were used to develop the sequence of events that occur to fulfil each of the tasks portrayed in the UCD. For each of the tasks depicted in the UCD an
equivalent SD was modelled. This approach was taken to ensure traceability from the requirements capture to analysis stage. In fact, the Telelogic UML suite used provides a project management hierarchy that ensures traceability between all stages of a project. Traceability ensures that all stages of design conform to requirements and also that implementation proceeds according to design, thus making it easier to isolate errors and determine exactly at what stage(s) the errors crept in. For instance, if a particular aspect of the PIWF does not function accordingly during implementation, it would be easier to determine whether it is due to inadequate specification, incomplete analysis or incorrect design, if traceability were maintained throughout the project.

In the analysis stage the interaction between the PIWF as an entity with the SIP and H.323 endpoints in typical scenarios enabling end-to-end calls were the main concern. Figure 15 shows the sequence of events involved in a call originating from a H.323 endpoint and terminating at a SIP endpoint via the PIWF, while Figure 16 is the SIP-H.323 equivalent.
3.4 Design phase

The requirements capture and analysis phases enabled the gradual emergence of an architectural design for the PIWF. The design began with the determination of the main functional entities required to implement the PIWF behaviour and then continued in an iterative manner with more details being added at each stage.

3.4.1 PIWF architecture design

The PIWF system architecture shown in Figure 17 consists of a H.323 session manager (HSM) whose function is to maintain the H.323 state machine. It coordinates the entities that work together to implement H.323 functionality, i.e. Q.931, RAS, and H.245 control etc. The SIP session manager (SSM), on the other hand controls the communication with the SIP side. There is also a central control unit, the Interworking Control Unit which monitors and controls
both SIP and H.323 call sequences (through the SSM and the HSM), performs protocol translation (call sequence mapping), and semantics adaptation (e.g. address translation) necessary to establish and maintain a hybrid SIP-H.323 call session. It also handles registration of H.323 endpoints with the Gatekeeper unit (GKU).

Figure 17. PIWF system architecture.

3.4.2 PIWF functional design
This stage focused on the design of the PIWF components using UML class diagrams and sequence diagrams. The resulting class model captures the characteristics and the static relationship that exists amongst the various objects that make up the PIWF system. The development of the class model was aided from elaboration of the analysis sequence diagrams. As objects that will fulfil the required tasks were conceived, they were included in the sequence diagrams to break up the sequence of events into specific well-defined sub-tasks that could be implemented.

In the process of adding detail, additional objects were also identified and retrofitted into the class model. Although the objects used to develop the sequence diagrams were derived from the class diagrams, the elaboration of the interaction between the objects enabled further development of the class diagrams. Furthermore, the attributes and methods of the classes in the class diagrams were identified through assigning responsibilities to the objects according to the events in the sequence diagrams. For example the `SendSignalSetup()` method assigns the responsibility of sending a SETUP signal towards the H.323 side, to the H.323Connection object. Thus a fully developed class model design was attained only after iterating back and forth between the sequence diagrams and the class diagrams.
Figure 18. Class model for PIWF design showing the main classes with their characteristics and static relationships.

The class model in Figure 18 is a detailed design showing the main objects that make up the PIWF and their relationship within the system. A description of the class model follows.

- The H.323 Session Manager is ‘stereotyped’ as a ‘block’ because it contains other instances of other sub classes, i.e. H.SM_main and the H.323listener, both of which are stereotyped as ‘process’. The filled diamond at the end of the association lines denotes this relationship. Only one instance of this class may run at any given time.
- The HSM_main object manages the H.323 sessions through instructions received from the ICU_Main object. It also creates as many instances of the H.323Connection class as required...
when it is instructed to initiate H.323 sessions with the H.323 side. It then hands over control to the H.323Connection object.

- The H.323listener object is instantiated on start up. Its purpose is to listen for incoming H.323 signalling message at a given port. Whenever it receives a SETUP PDU it creates an H323Connection object and hands over the session to it, after notifying the HSM_main object of the arrival of the H.323 signal.

- The H323Connection object sends and receives signalling PDUs to and from the H.323 side. It also communicates with the HSM_main from which it receives commands that trigger its change from one state to another during a session. If a call set-up is successful, the H.323Connection object creates an H245Control object, which handles the H.245 procedures.

- The Interworking Control Unit embodies the ICU_Main class. The ICU_Main class is the central coordinator of the events that occur to provide the overall PIWF functionality. It maintains the states of both SIP and H.323 sides through communication with the HSM_main and the TransactionController. It is responsible for mapping the sequence of messages of both sides and the adaptation of protocol semantics into the appropriate format. All these functions are executed through the methods depicted in the class diagram. For instance the SetupRequest() method is invoked to pass a request to the HSM_main for a SETUP message to be sent to the H.323 side. This happens when the TransactionController notifies the ICU_Main (through its onInviteIndicate() method) of the arrival of a SIP INVITE.

- The SIPsessionManager is also a ‘block’ that contains the TransactionController, the ClientTransaction and the ServerTransaction classes. The TransactionController creates instances of either the ClientTransaction when a SIP session needs to be initiated, or a ServerTransaction when an invitation to a session is received from a SIP UA.
3.4.3 Object Sequence diagrams

In addition to aiding the class model design, the UML sequence diagrams also illustrated the dynamic relationship between the various objects in the class model. For instance, the sequence diagram in Figure 19 depicts a SIP UA sending an INVITE message, which is received by the TransactionController object of the PIWF. The TransactionController method CreateServerTransaction() is used to create a ServerTransaction process to handle the request. Also, an INVITEindicate signal is sent to the ICU_Main via the onReceivedInvite() method as an event notification. This triggers the ICU_Main process to send a SETUPrequest signal to the HSM_main process via its own SetupRequest() method. The INVITEindicate signal could be implemented as an object that will convey the parameters of the SIP UA to the ICU_Main. Likewise, the SETUPrequest signal should embody the adapted or translated parameters in H.323 format. The translation or adaptation which should occur as described in section 2.5, could be invoked within the onReceivedInvite() method by the ICU_Main.

The HSM_main upon receiving the SETUPrequest signal creates an H323Connection object and then instructs it to send a SETUP PDU to the recipient H.323 endpoint. The H323Connection then proceeds to establish a session using the normal H.225 (Q.931) call set-up procedure. It notifies the HSM_main of the arrival of a CONNECT response while also instantiating a new H245Control object to carry on with the H.245 procedures. The H.245 procedure begins after the receipt of an H245Start signal from the H323Connection. The H245Control object is destroyed at the end of the H.245 message exchange, and if it was successful, an H245Success notification is sent to the HSM_main, otherwise an H245fail is returned. The resulting notification message eventually reaches the ICU_Main, and, if it was an H245Success, then a 200_OKrequest signal is sent to the TransactionController (via InviteRequest() method) which in turn gives instruction to ServerTransaction to send a 200 OK
response to the SIP UA. The ServerTransaction is also terminated as soon as it sends the 200 OK.

Again, the H245Success notification would be used to convey the session description parameters from the H.323 side. The ICU_Main handles the translation of the parameters into SIP compliant format and then forwards them in the 200_OK request. These session parameters are used to construct the 200 OK PDU that is sent to the SIP side. Hence, with both the H.323 side and the SIP UA having received each other’s session description (indicating the codecs, bit rates and media receive ports), direct media exchange can now commence using RTP as transport.
Figure 19. Sequence diagram illustrating the events and messages involved in a SIP initiated call.

Figure 20, is an equivalent sequence diagram for a H.323 initiated session. The H323 listener object which listens for incoming H.323 calls, creates an H323Connection object to which it passes the incoming SETUP message. H323Connection then sends a SETUPindicate notification to HSM_main object which in turn notifies the ICU_Main. ICU_Main sends an InviteRequest signal to the TransactionController, prompting it to
instantiate a new ClientTransaction object. The ClientTransaction receives an instruction to send an INVITE message without a (SDP) session description. This is because the session parameters from the H.323 side are not yet known at this point.

The 200 OK from the SIP side will have a session description, which is conveyed to the ICU_Main (through the TransactionController) via a 200_OKresponse notification signal. Now that the SIP side session parameters are available, the ICU_Main adapts them to create a H.323 compliant set of parameters. The parameters are carried in the ConnectRequest sent to the HSM_main. These parameters are used during the H.245 procedures handled by the H245Control object which is instantiated by the H323Connect after the latter has sent a CONNECT response to the H.323 endpoint.

As with the scenario in Figure 19, when the H245Control completes the H245 process, it returns either an H245Success or H245fail prior to being terminated. The H245Success will carry the session parameters of the H.323 endpoint and this is adapted yet again to a SIP format by the ICU_Main. The parameters are embodied within the ACKrequest that is sent to the TransactionController, which is in turn prompted to send an ACK message (containing session description body) to the SIP UA.

At this point, all session parameters are available to both sides and thus media channels are open to exchange voice, video or data packets using RTP.
Figure 20. Sequence diagram illustrating the events and messages involved in a H.323 initiated call.

3.5 State machine modelling

The PIWF design will be incomplete without including Object state machines. Object state machines detail the behaviour of each class/object in the class model, which cannot be depicted by class diagrams and sequence diagrams alone. The UML state chart diagram (or state transition diagram) was used to model the object state machines. The state chart diagram
shows all the states of an object, the messages it can receive or send, and all possible events and transitions that are triggered during the lifetime of an object.

Figure 21. State transition diagram for Interworking Control Unit (ICU)

The ICU_Main State machine is shown in Figure 21. It goes into IDLE state on start up and when it receives a SETUPindicate (from the HSM_main), it sends an INVITErequest (to the TransactionController) and goes into the AWAIT_200_OK state. Thus SETUPindicate is a ‘triggering, message because it leads to a change in the state of the ICU_Main object. While it is in the AWAIT_200_OK state, if it receives an 180RINGINGindicate (from TransactionController), it will send an ALERTINGrequest (to HSM_main) and remain in the same state. Thus 180RINGINGindicate is a ‘non Triggering or informational’ message. The same receive/send convention is used throughout the rest of the diagram. From the class model, we see that ICU_Main interacts directly with only three classes. The message naming
convention adopted (e.g. \textit{SETUPindicate, INVITErequest}) indicates which class is the \textit{sender/recipient}.

![H323 Session Manager state machine](image)

Figure 22. H323 Session Manager state machine.

The HSM\textsubscript{main} state machine is illustrated in the state chart of Figure 22. Again, from the class model, it is clear that the \textit{sender/recipient} of the messages is either the ICU\textsubscript{Main} or the H.323Connection object. Unlike, the ICU\textsubscript{Main} which deals with both SIP and H.323 messages, the HSM\textsubscript{main}, as the state diagram shows, only handles H.323 messages.
Figure 23. TransactionController state machine.

Figure 23 is the state machine of the TransactionController. Its role is to listen for incoming SIP calls and then create a ServerTransaction object upon receipt of an INVITE. When an outgoing call request is made (from the ICU_Main) it creates a ClientTransaction object. The state machine diagrams of the ServerTransaction, ClientTransaction, and other objects in the class model are given in Appendix B.

3.6 PIWF modelling with SDL

After the PIWF design, the next step was the modelling in SDL for verification. For this purpose, the Telelogic Tau SDL suite was used. SDL, being a graphical language enables the visual design of models instead of using only a textual notation. SDL provides graphical structuring features (blocks, etc.), state machines and communication through signals that are not available in programming languages such as C++ or Java [25].
The SDL models were developed from the UML class diagrams, the sequence diagrams and the state transition diagrams. The hierarchy of the SDL model was derived directly from the associations in the class model. Figure 24 depicts the highest level of the model. It shows the PIWF modelled as a block within the interworking system, with incoming and outgoing channels. The signals to the left of the PIWF are to/from the H.323 side, while those on the right are from/to the SIP side. Next in the hierarchy are the ICU, HSM, and SSM, which were modelled as blocks in the PIWF, as shown in Figure 25.

Figure 24. PIWF in the SIP-H.323 interworking system showing incoming and outgoing channels.
The blocks consist of processes, which are equivalent to the classes with the ‘process’ stereotype in the UML class model. Each process contains an extended finite state machine derived from the corresponding UML state chart. Communication between processes is via signal routes or channels. The signals represent the function calls and messages passed between the objects as illustrated in the UML sequence diagrams. For example, in Figure 25, ssm2icu declared as a SIGNALLIST represents the list of possible signals that could be passed from the SIPsessionManager to the InterworkingControlUnit via the channel SSMtoICU. Similarly, icu2ssm represents a set of signals that can travel in the reverse direction through the same path. The same convention applies throughout the entire model.
The H.323SessionManager block is uncovered in Figure 26. It contains four processes (equivalent to the UML classes) interconnected with channels through which signals are exchanged. C1 is the route to the ICU, while C6 and C8 are the channels to the Environment (i.e. external systems) through which signals are exchanged with H.323 endpoints. Again, the signals are denoted as SIGNALLISTs, which are declared in the curled boxes. The numbers in brackets, shown on the processes indicate the number of instances that can run at start up as well as the maximum instances allowed. For instance, the (1,1) on the HSM_main process indicates that only one instance can run on start up and the maximum allowable instances is also one. Meanwhile, the (0) on the H245Control indicates that zero instances are run on start up while no maximum limit specified.
Figure 27. ICU block in SDL

Figure 27 is the InterworkingControlUnit block, which houses the ICU_Main process. The process communicates via C1 and C2 to HSM and SSM respectively. The SIPsessionManager on the other hand, has three processes, the Transactioncontroller, ClientTransaction and ServerTransaction, shown in Figure 28.
After modelling the system, blocks, and processes together with their intercommunication signals and routes, the SDL state machines were developed next. These were contained within the processes shown in the previous Figures and defined the behaviour of each of the processes. For example, the ICU_Main process uncovered in Figure 29 shows the SDL state machine. The state machines for all other processes are given in Appendix C.
3.7 SDL compilation and Simulation

- **SDL compilation**
  
The next step taken after the conversion of the UML design to an SDL model was to compile the SDL model into an executable file ready for simulation. The compilation was successful after several attempts and cycles of debugging.

- **SDL simulator UI**
  
The SDL simulator UI tool of the Telelogic™ Tau SDL suite was used for the interactive simulation of the scenarios described in the analysis and design stages with UML sequence diagrams. The aim was to simulate the sequence of events as given in the sequence diagrams in order to verify the correctness of the design in meeting the specified requirements.
The simulation UI tool allowed for the sending of signals to the processes and then the transitions were stepped through in succession. While the transitions were being made, the messages and states were concurrently monitored on an MSC trace. Figures 30 and 31 are snapshots of the simulation process in the Simulator UI of the Telelogic SDL suite.

Figure 30 the simulator ready for SDL animation and MSC recording
Figure 31 the simulator after sending SETUP pdu to H323 listener.
Chapter 4

RESULTS AND DESCRIPTION

The results of the simulation of the PIWF functionality using the SDL model developed from the UML design, is presented in this chapter. The outputs of the simulation were captured with MSC (message sequence chart) traces generated using the SDL simulator, which is part of the Telelogic SDL 4.5 suite used in the project. Efforts were concentrated on simulating aspects of the PIWF functionality deemed to be most crucial in attaining the minimum requirements of interworking SIP and H.323. In particular, call set-up scenarios from SIP endpoint to H.323 and vice versa were investigated. The aim, ultimately, was to ensure that the SDL PIWF model was working correctly in accordance to the specification and design that it mirrored.

4.1 Simulation results of SIP to H.323 call signalling

The results of a simulated call set-up scenario involving a call originating from a SIP endpoint is shown in the MSC trace of Figure 30. All incoming signals originate from the
Environment entity, represented by the \textit{env} \_0 object; and so are all outgoing signals (from the PIWF). Thus the \textit{env} \_0 emulates both the originating SIP endpoint as well as the recipient H.323 endpoint. When an INVITE signal is sent to the PIWF, it triggers a sequence of events, which begins with the creation of a ServerTransaction object by the TransactionController and ends with the transmission of a 100\_TRYING by the ServerTransaction. The latter happens after TimerA, which is programmed to timeout after 200ms, expires.
Figure 30. Simulation output MSC trace for a SIP originated call.
Another sequence is triggered by the receipt of a CONNECTpdu (from the H.323 side). This is handled by the H323Connection process, which in turn creates an instance of H245Control to carry out H.245 procedures. On completion of the H.245 exchange, the H245Control is destroyed after sending an \textit{H245SUCCESSind} notification that eventually reaches the ICU_Main. The control of the session returns to the SIP side, when the ICU_Main sends a \textit{200_OKreq} to the TransactionController, which in turn issues the same request to the ServerTransaction. The ServerTransaction is destroyed after it sends a 200 OK to the SIP endpoint. The incoming ACK response from the SIP endpoint is intercepted by TransactionController, which then sends the notification to the ICU_Main. At this point, the session parameters would have been exchanged between the two endpoints and the media session can now commence.

\textbf{4.2 Simulation results of H.323 to SIP call signalling}

An equivalent simulation output trace for a call originating from an H.323 endpoint is shown in Figure 31. One simplifying assumption was made in this simulation; i.e. the H.323 endpoint was pre-registered with the PIWF gatekeeper and had a pre-granted admission status. Hence, the registration and admission (RAS) procedures are not shown in the simulation trace.

The call set-up begins with the arrival of the SETUPpdu at the H323listener port, which then creates an H323Connection and passes the former to it. The H323Connection notifies the HSM_main, which also communicates this to the ICU_Main. The ICU_Main requests the SIP section to set up a connection to the desired SIP endpoint. This task is handled by the ClientTransaction, which is created as soon as the request reaches the TransactionController. Two timers, Timers A and B are started. TimerA controls retransmission of INVITE requests (as per RFC 3621) after every 500ms. While TimerB is set to fire and terminate the session after 32 seconds if no response is received from the SIP endpoint. Timers are required because SIP uses UDP (or TCP) to transport signalling messages.
Interworking SIP and H323 for VoIP applications

Figure 31. Simulation output MSC trace for an H.323 originated call.
With the arrival of a 200 OK response from the SIP endpoint, the ClientTransaction notifies the TransactionController and then terminates immediately, having completed its task. The notification is sent to the ICU_Main, which then returns control to the H.323 session manager. H323Connection is then prompted to send a CONNECTpdu and at the same time it creates an H245Control to handle H.245 procedures. As in the previous scenario, H245control sends a success notification to HSM_main and terminates when it has completed its task. Again, HSM_main sends this success indication to the ICU_Main, which then instructs the TransactionController to send an ACK. After the ACK is sent to the SIP endpoint, session parameters have now been exchanged and media communication between the H.323 and SIP endpoints can now begin.

4.3 PIWF verification.

To verify the operation of the PIWF against the expected behaviour, the SDL simulation trace outputs (Figures 30 and 31) are compared with the UML sequence diagrams developed in the design stage (Figures 19 and 20). A close comparison reveals a near perfect match between the two (except for the depiction of SIP timers in the trace outputs). Thus, this verifies the correct operation of the PIWF model, giving credence to and validating the design against the main scenarios.
Chapter 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

H.323 and SIP are currently the two most prominent and widely deployed VoIP and multimedia signalling protocols. H.323 is a suite of vertically integrated protocols that provide a tightly coupled multimedia conferencing solution. SIP, on the other hand, is modular, flexible and extensible protocol for initiating and controlling variety of sessions, VoIP and multimedia inclusive. The issue of interoperability in general and interworking between the two protocols in particular is an important one within the VoIP industry and standardization bodies. Interworking SIP and H.323 means that manufacturers don’t have to support both protocols in their products. This in turn offers operational cost benefits to service providers while making consumer end products cheaper as well.

The main aim of this project was to develop an interworking solution that would enable SIP-H.323 interoperability. An indepth investigation of the interworking issues was undertaken, and this resulted in the decision to incorporate formal methods in the software development life cycle. In particular, the requirement of the interworking solution to operate within a real-time, event-driven environment whilst handling concurrent communications was the main motivation for employing a formal methodology to support verification of the solution before implementation.

Thus, the first phase involved a development cycle from requirement specification, analysis, to the design of a Protocol Interworking Function (PIWF) using UML tools provided by the Telelogic Tau UML suite. UML diagrams, such as Use cases, sequence charts, class diagrams, and state charts, provided a comprehensive description of the architectural, functional and operational aspects of the PIWF. These were then used in the second phase to develop a model in SDL notation, using the Telelogic SDL suite. The SDL model was compiled and used
to simulate in an interactive manner, the operation of the PIWF in the envisaged deployment scenarios. The outputs of the simulations were captured on MSC traces. Where anomalies were detected, they were traced back to the UML design and corrected until the simulation output correlated with the expected behaviour of the system thus verifying the correctness of the design.

This project has therefore achieved a validated PIWF design, which if implemented or coded correctly in a high level language would produce a functional and reliable SIP-H.323 interoperability solution. The validation will also accelerate time-to-market and make maintenance easier since there is now high confidence that design-induced bugs have been eliminated. Furthermore, new requirements could be easily be added if the need arises. Since the models were developed with traceability, and in an object oriented approach, this can be accommodated without the need to redesign or revalidate the entire system from scratch.

5.2 Recommendations for further work

- A code implementation of the PIWF in an object oriented high level language such as C++ should be developed directly from the UML class models.
- After coding, the PIWF should be deployed and analysed for performance.
- The PIWF SDL model could be further developed to include object-oriented features of SDL so as to allow the components built to be reused as building blocks for modelling and investigating other possible deployment scenarios not covered in this project.
- The models could be expanded to simulate and investigate the interworking of advanced supplementary services.
REFERENCES


69
### Appendix A. List of acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>3G Partnership Project</td>
</tr>
<tr>
<td>ACF</td>
<td>Admission Confirm</td>
</tr>
<tr>
<td>ARQ</td>
<td>Admission Request</td>
</tr>
<tr>
<td>BGP</td>
<td>Border Gateway Protocol</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Software Engineering</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunication Standard Institute</td>
</tr>
<tr>
<td>FDDI</td>
<td>Fibre Distributed Data Interchange</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IMTC</td>
<td>International Multimedia Teleconferencing Consortium</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISDN</td>
<td>Integrated Services Digital Network</td>
</tr>
<tr>
<td>ITU-T</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>Mbone</td>
<td>Multicast Backbone</td>
</tr>
<tr>
<td>MC</td>
<td>Multipoint Controller</td>
</tr>
<tr>
<td>MCU</td>
<td>Multipoint Control Unit</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail</td>
</tr>
<tr>
<td>MP</td>
<td>Multipoint Processor</td>
</tr>
<tr>
<td>MSC</td>
<td>Message Control Sequence</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Network</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAS</td>
<td>Registration, Admission and Status</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource Reservation Protocol</td>
</tr>
<tr>
<td>RTCP</td>
<td>Real-time Transport Control Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real-time Streaming Protocol</td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>TIPHON</td>
<td>Telecommunications and Internet protocol Harmonisation Over Networks</td>
</tr>
<tr>
<td>UA</td>
<td>User Agent</td>
</tr>
<tr>
<td>UAC</td>
<td>User Agent Client</td>
</tr>
<tr>
<td>UAS</td>
<td>User Agent Server</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications Service</td>
</tr>
<tr>
<td>URI</td>
<td>Universal Resource Indicator</td>
</tr>
<tr>
<td>URL</td>
<td>Universal Resource Locator</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over Internet Protocol</td>
</tr>
</tbody>
</table>
Appendix B. PIWF object state chart diagrams

[Diagram of state transitions]

- Ready
  - SETUP/SETUP
  - AWAIT_CONNECT_PDU
    - CONNECT/CONNECT response, Create H245Control process
      - AWAIT_H245
        - H245Success/H245Start/H245End
        - CONNECT/CONNECT response, Create H245Control process
          - CONNECT/CONNECT response, Create H245Control process
    - CONNECTED
      - Close logical channel/H245End

- INVITE/INVITE
  - Setup Timer A, Timer B, Timer D, Timeout:=T1 Sec
    - Reset Timer A, Timeout:=Timeout x 2
      - 180RINGING/180RINGINGindicate
        - Timer B expires[-64 x T1]
      - 200_OK/200_OK response
        - Timer D expires[T=0 or T=64 x T1]

- Proceeding
  - 4XXERROR/ACK, ACKindicate
  - 4XXERROR/ACK

- Completed
1) State chart diagram for H323Connection object
ClientTransaction object

2) State chart diagram for

3) State chart diagram for ServerTransaction object
chart diagram.

4) H323listener State
Appendix C. SDL process diagrams
I
nterworking SIP and H323 for VoIP applications

1) HSM_main process state machine in SDL.
2) H323Connection

process state machine in SDL.
3) TransactionController process state machine in SDL.

4) ClientTransaction process state machine in SDL(page 1).
5) ServerTransaction process state machine in SDL.

6) H323listener process state machine in SDL.

ix