Providing Media Control to SIP-based IVR Applications: The IVRObject Approach

Renato Simoes

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Under the auspices of the Ottawa-Carleton Institute for Computer Science

uOttawa

University of Ottawa
Ottawa, Ontario, Canada
January 2009

Abstract

A popular application that runs on a Voice-over-IP network is the Interactive Voice Response (IVR), which provides a way for interacting with an end user's phone via a telephony interface by presenting a set of audible menu options, and collecting the user's Dual Tone Multi-Frequency responses as the user presses the telephone numeric keypad, and recording the user's voice.

This thesis takes a closer look into IVR systems on an IP-based network, specifically in networks that support the Session Initiation Protocol (SIP) for controlling the telephony signalling, and more specifically on IVR systems that have their SIP signalling controlled by a SIP Application Server (SIPAS) implementing the SipServlet 1.0 or 1.1 specifications and where a SIP-based Media Server is used to stream IVR media to the end-user.

We describe existing ways for supporting an IVR development in a SIP network, and then we propose an alternative way of accomplishing the same task: the *IVRObject*. With the help of prototypes, the IVRObject is compared with the existing state of the art against three criteria: a) how easy it is to develop, b) how portable the development solution is, c) and how scalable the solution is in order to sustain a high call volume.

As a conclusion, it will be highlighted that the IVRObject provides an easy mechanism for development of IVR-based applications running on a SIPAS, that it is portable to different media server vendors, and that it supports a test strategy that can be leveraged to improve software development quality and faster development. This makes the IVRObject a good alternative especially for enterprise-based IVR applications where scalability is less of an issue than in carrier-space applications.

Acknowledgment

First and foremost, I would like to thank my wife, Adriana Simões, for her support and patience during the 4 years I was involved in this part-time Masters. It wasn't always easy to deal with pressures from a full-time work and still have to attend university classes at night, spend long weekends getting assignments done and studying for exams, and getting this thesis ready with research, prototypes and reviews. I would have never been able to balance the time between my studies, family and social life without her.

Gostaria também de agradecer ao apoio que os meus pais me deram, o Sr. Arlindo Simões e D. Lucy. A distância que nos separou não foi sempre fácil, e desculpem-me se faltei-lhes com a devida atenção durante estes últimos 4 anos, mas obrigado pelo incentivo que vocês sempre me deram. Agora com o fim desses estudos, quem sabe, eu tenha um bom motivo para retornar a terra natal e tentar recuperar um pouco deste tempo e atenção que lhes devo.

I would like to express my gratitude to five old friends: Rossana Andrade, Riverson Rios, Jerffeson Souza, Igor Sales and Jauvane de Oliveira for having given me the incentive for getting me aboard this long part-time study.

Last, but not least, I would like to thank my supervisor and friend, Professor Daniel Amyot. It has been a real pleasure working with him. His views, leadership, and knowledge are things that I truly admire. His input, the discussions, and the countless drafts reviewed we had, made this thesis work possible and also enjoyable.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
3PCC	Third-Party Call Controller
API	Application Programming Interface
B2BUA	Back-To-Back User Agent
CCXML	Call Control eXtensible Markup Language
DTMF	Dual Tone Multi-Frequency
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IVR	Interactive Voice Response
IVVR	Interactive Video and Voice Response
MCD	Minimum Common Denominator
MOML	Media Objects Markup Language
MS	Media Server
MSCML	Media Server Control Markup Language
MSML	Media Server Markup Language
M*ML	MSCML and MSML
PSTN	Public Switched Telephone Network
RFC	Request for Comments
RMI	Remote Method Invocation
RTP	Real-time Transport Protocol
SBC	Session Border Controller
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SIPAS	SIP Application Server
SIPUA	SIP User Agent
TA	Terminal Adaptor
TCP	Transmission Control Protocol
TTS	Text-to-Speech
UA	User Agent
UDP	User Datagram Protocol
VoiceXML	Voice eXtensible Markup Language
VoIP	Voice over IP
VXML	VoiceXML
WAR	Web Archive
WS	Web Server
XML	eXtensible Markup Language

Chapter 1. Introduction

1.1. Motivation

Voice over IP (VoIP) is becoming more and more common these days as the telephony industry is driven by both cost reduction and greater flexibility in handling calls. Even if an end user does not have an IP phone, after a call is made and before the final signalling reaches the other end, there is a good chance that, via signalling gateways, the call has traversed an IP-based network.

A popular application that runs on a VoIP network is the Interactive Voice Response (IVR), which provides a way for interacting with an end user's phone via a telephony interface by presenting a set of audible menu options, and collecting the user's Dual Tone Multi-Frequency (DTMF) responses as the user presses the telephone numeric keypad, and recording the user's voice.

The development of Interactive Voice Response applications on Session Initiation Protocol (SIP [20]) networks leads to several challenges and difficulties including:

a) complex signalling that has to be handled by the IVR developer;
b) race conditions that occur due to the asynchronous nature of SIP and where a SIP end point can behave at any time as a server or as a client;
c) different and incompatible protocols for invoking a Media Server (MS) that makes the IVR application dependent on a certain MS vendor and not easily portable;
d) having a development testing strategy implemented that targets development quality.

Existing approaches to IVR development are often insufficiently flexible when the above concerns need to be addressed. This thesis proposes an innovative alternative for addressing such difficulties and points out its strengths and weaknesses.

1.2. Research Objective

This thesis aims to propose a flexible way of providing IVR capabilities to end users via a telephony interface where the IVR application is run on a SIP network. This proposal is named *IVRObject*.

More specifically, IVRObject will target IVR applications that are deployed and run on a SIP Application Server (SIPAS) that either implements the SipServlet 1.0 specification (JSR-116 [15]) or the SipServlet 1.1 specification (JSR-289 [8]), and where a SIP-based MS is used to stream IVR media to the end-user.

IVRObject will balance the existing and sometimes conflicting forces coming into play such as flexibility and scalability. Flexibility aspects, such as the ease of development of IVR applications and their ability to be run using different media servers, will have a higher priority than scalability aspects, which include the ability to have the application scale in order to handle a great number of simultaneous calls.

By achieving this goal, we believe the IVRObject will result in a development alternative that requires a simple signalling and provides a mechanism for developers to automate their development via call simulation, and where the developed application is more likely portable to different media servers.

The conflicting force, scalability, will have a lower priority. This trade-off is believed to make the IVRObject a good alternative for applications that need to be developed quickly with high quality and that are run on networks that will have a limited numbers of users (thousands instead of millions). This will make IVRObject a good match for enterprise-based applications, rather than carrier-space and service-provider ones.

1.3. Thesis Contributions

The IVRObject approach addresses several of the challenges around the development of IVR applications that are run on a SIP-based network and that are deployed and run on a SIPAS. This approach:

- Provides a mechanism that shields the application developer from having to know how to communicate with the MS in order to have media streamed to the end user.

- Provides a simple Application Programming Interface (API) that hides the complexity of handling the SIP signalling and the parsing of its messages.
- Provides a mechanism for the developer to apply an automated testing strategy to improve the development quality.
- Hides the inner mechanism for achieving a call-back communication from the MS to the SIPAS, which allows the call-back mechanism and protocol to be changed easily in the future without affecting the application.
- Provides a mechanism for abstracting the application from dealing with different MS that might support different versions and flavours of VoiceXML (VXML), further enhancing its portability.
- Provides a separation of the application logic from the signalling handling that increases the reusability of components and leads to faster application development.

The thesis includes the implementation of the IVRObject approach and the results of a comparative experiment based on an auto-attendant case study.

1.4. Thesis Outline

The rest of the thesis is divided into seven chapters:

- Chapter 2 presents concepts and definitions related to SIP and IVR that will initiate the reader to the basic concepts relevant to the thesis.
- Chapter 3 presents the evaluation criteria and a use case that will be used for evaluating the state of the art as well as the thesis' proposition.
- Chapter 4 presents the state of the art in IVR development, and the outcome of the evaluation of two prototypes against the criteria defined in Chapter 3.
- Chapter 5 introduces the new IVRObject approach, together with its concepts, architecture, application programming interface, and testing strategy.
- Chapter 6 presents the IVRObject prototype and its evaluation against the criteria defined in Chapter 3.

- Chapter 7 summarizes and further analyses the results collected during prototyping, for both of the state-of-the-art approaches and also for the proposed thesis' alternative, but this time from a comparison perspective.
- Chapter 8 presents the overall conclusions, contributions and suggested future work.

These chapters are meant to be read in sequence. In order to help with the numerous acronyms used in this thesis, a glossary is provided on page ix.

Chapter 2. Background

This chapter reviews the concepts related to the thesis' proposition, in order to ground the reader on the relevant background around this work.

2.1. Interactive Voice Response (IVR)

An IVR application (or simply IVR) can serve multiple purposes such as accessing a banking system, accessing voice mail, or reaching an auto-attendant extension.

The end user interacts with the IVR via a telephony interface, that is: using the telephone device (either mobile, softphone or landline) to hear the instructions of the IVR and the phone's numeric keypad to respond to the IVR. For example:

- system: "*please enter you account number*"
- user: <enters the required numeric info via the phone key pad>
- system: "*now, please enter your access code*"
- user: <enters the required numeric info via the phone key pad>
- system: accesses the user records
- system: "*your total balance is $100.56, please press 1 for….*"

The popularity of IVR is mostly due to the fact that anybody can use it from any type of phone, providing a universal access that does make any distinction on the type of device the user has.

2.2. SIP Related Background

2.2.1 Session Initiation Protocol (SIP)

SIP is now the de-facto standard for Voice over IP (VoIP) networks. It is a layer-7 (application layer) protocol.

During this work we will mainly use the basic SIP as defined in its original RFC 3261 [20] that is responsible for the establishment of connections between two end points, but also the SIP INFO extension (detailed in RFC-2976 [9]) as a mechanism for communication between a SIP application server (SIPAS) and a media server (MS), which is needed by one of the state-of-the-art alternatives for supporting IVR control.

In a nutshell, SIP is used for establishing/initiating a session (the talking path) between two end points. It is important to note that the actual streaming of the audio between the SIP end points is outside of the SIP messaging scope and is actually done via other protocols such as the Real-time Transport Protocol (RTP [24]), according to the terms agreed by both end points on the media options exchanged via SIP.

The following call flow illustrates the basic SIP usage for connecting two generic SIP end points, also known as SIP User Agents (SIPUA), as per RFC-3261 [20].

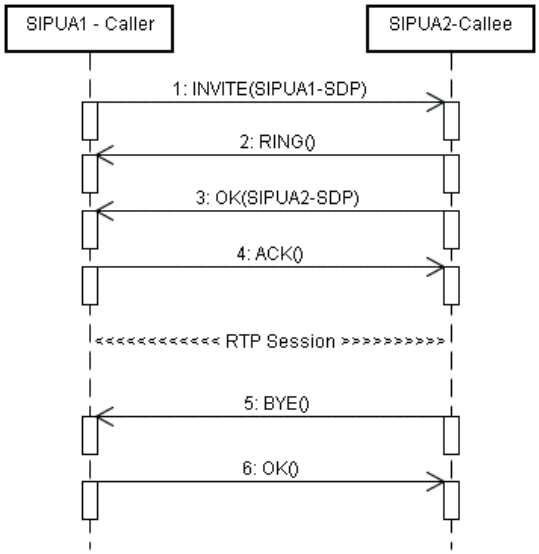

Figure 1 Basic SIP Call

Table 1 clarifies the meaning of the signals exchanged between the above two SIPUA.

Table 1 Basic SIP Call – Message Details

SIP Message	Details
1-INVITE	The Caller UA (SIPUA1) takes the initiative to connect to the Callee UA (SIPUA2). It sends an INVITE SIP message for this, and along the INVITE it carries a payload of its media options (SIPUA1-SDP) described via the Session Description Protocol (SDP [13]). In other words, SIPUA1 contacts SIPUA2 and passes SIPUA2 the means/options describing how the talking path can be established.
2-RING	The Callee UA starts ringing, and lets the Caller UA know about it. Upon receiving the RING, the Caller UA would typically play back a ring tone to let the caller (the person) know the Callee UA is ringing.
3-OK	This means the callee has answered the call (picked up the phone). Along with this SIP message, the media options for the Callee UA (SIPUA2-SDP) are passed to the Caller UA.
4-ACK	Caller UA acknowledges the media options exchange and the talking path is established.
5-BYE	The message sent by the endpoint that decides to terminate the call (the Callee UA in this case).
6-OK	The acceptance of the BYE. At this point the RTP (the talking path) is torn down.

2.2.2 SIP Message Structure

A SIP message is composed of 5 parts:

- The "Request URI" identifies where the message should be sent to, the SIP message type, and the protocol version used.
- The "System Headers", automatically added and maintained by the SIPAS runtime environment. The application has no control over them – their purpose is to maintain the call state and call route.
- The "Mandatory Custom Headers" that are application-specific values, but that need to be specified.
- The "Optional Custom Headers", which are application-specific values that the application is free to define for its own needs.
- The "Body", which is the payload of the message. It is optional but when specified it typically carries the SDP with the media options. During this

study, we will also see INFO messages carrying XML payload to instruct the media server to execute specific IVR commands.

This structure is illustrated with the following INVITE message bellow:

Table 2 SIP Message Structure

Part	**SIP Message**
Request URI	INVITE sip:AA-IVR@sipas.com SIP/2.0
System Headers	Via: SIP/2.0/UDP 1.1.1.1;branch=z9hG4bKac1888903573 Call-ID: 18888915512620081956 37@1.1.1.1 CSeq: 1 INVITE Contact: <sip:caller@1.1.1.1> Content-Length: 125
Mandatory Custom Headers	From: <sip:caller@test.com>;tag=1c1888892482 To: <sip:AA-IVR@sipas.com> Content-Type: application/sdp
Optional Custom Headers	MyHeader: MyValue
Body (carrying the SDP media options in this case)	v=0 o=caller 1888866924 1888866604 IN IP4 1.1.1.1 s=Phone-Call c=IN IP4 1.1.1.1 t=0 0 m=audio 6010 RTP 18 8 0 4 2 96 a=rtpmap:18 G729/8000 a=fmtp:18 annexb=no a=rtpmap:8 PCMA/8000 a=sendrecv a=rtcp:6011 IN IP4 1.1.1.1

2.2.3 SIP User Agent (SIPUA)

A SIP User Agent is an endpoint that can make use of SIP to establish a talking path (RTP session).

A SIPUA is hence an entity that is not limited to a SIP-enabled phone, as it can also be the SIP element of a SIP gateway, of a SIP firewall, of a SIP-enabled MS, or of a SIPAS. That is, the call flow illustrated in Figure 1 could well be an interaction between a SIPAS and a MS; it would look exactly the same.

2.2.4 SIP Enabled Media Server (MS)

As explained before in section 2.2.1, the actual audio streaming flows via the RTP session. Therefore, the MS is the entity that interacts directly with the end-user SIPUA in order to play prompts, collect digits and record the audio stream.

Although the RTP flows directly from MS to an end-user SIPUA, the SIP signalling does not, and the SIPAS will act as a middle-man to insulate one from the other from a SIP perspective.

The MS types we will be looking into are the SIP-aware ones, that is, the ones that expect SIP to be used as a protocol to establish the RTP session as well as to carry the triggers for telling the MS what needs to be streamed to the SIPUA on the other end of the RTP session.

2.2.5 SIP Application Server (SIPAS)

A SIPAS is to VoIP applications what an HTTP [11] application server is to Web applications. It is a container where different applications can be deployed at the same time, and that container takes care of handling the multiple threads needed by the different applications, of isolating the memory space so applications do not interfere with each other, of creating the sockets to send the signals out, and of handling transport-related aspects on the application behalf to ease the development.

The SIPAS is an implementation of the JSR-116 specification [15], and more recently its updated JSR-289 version [8], that defines the container role and the SIP Servlet API.

SIPAS works as an orchestrator for the SIP signalling needed by a specific application. SIPAS does not terminate the media, meaning no RTP will flow from/to the SIPAS. In our case, SIPAS will make use of common signalling patterns in the SIP world such as the B2BUA (section 2.2.7), and the 3PCC [21] (section 2.2.8) in order to exchange the media options from the end-user SIPUA and the MS so they can “talk” to each other while insulating these end points from direct SIP signalling. This signalling insulation is important as the MS often has very specific needs for the SIP signalling (as will be further detailed during our analysis of the state of art) and the user SIPUA is often just a standard SIPUA that would not know how to talk via SIP to the MS otherwise.

2.2.6 SIP Servlet

The SIP Servlet is defined in the JSR-116 specification [15] and more recently its updated JSR-289 version [8].

A SIP Servlet is to a SIPAS what a HTTP Servlet is to a Web application server. It provides an API for an application developer to create SIP-aware applications that will run on a SIPAS. It has methods to create a request, create a response, manipulate headers and payload, and get notified when a new request or response arrives in order to embed specific business logic to achieve a specific signalling.

The fundamental difference between SIP and HTTP is that SIP is asynchronous and also bi-directional. That is, while in HTTP there is the concept of a client that issues the request and a Web server that sends the response, in the SIP world any of the SIPUA can be a client or a server. For example, in Figure 1, the Caller is acting as a client when sending the INVITE request, but it is the Callee that is acting as a client when sending the BYE (it is then bi-directional). Also, when the Caller issues the INVITE request, the OK response does not come in the same thread, it will come later at some point when the callee answers (it is then asynchronous).

These are fundamental differences that in fact make the SIP Servlet development quite difficult to manage as race conditions often occur.

2.2.7 Back-To-Back User Agent (B2BUA)

B2BUA is a signalling pattern in the SIP world. It is implemented in a SIPAS via the SipServlet API.

An application, coded as a B2BUA is positioned in between two SIPUA with the intent of preventing them from having direct signalling exchange. This might be done for different reasons, such as having the application staying on the path for billing purposes (as it will know when a call has started and ended), for applying specific business logic during the call, but more specific to our study as a way for the SIPAS to establish the "talking path" (RTP session) between the end-user SIPUA and the MS at the same time as insulating the SIPUA from the special signalling that the MS requires to achieve a certain IVR functionality such as play a prompt, collect a user input that was entered via the end-user telephone keypad, and record the end-user's voice.

The following diagram shows a basic B2BUA call flow.

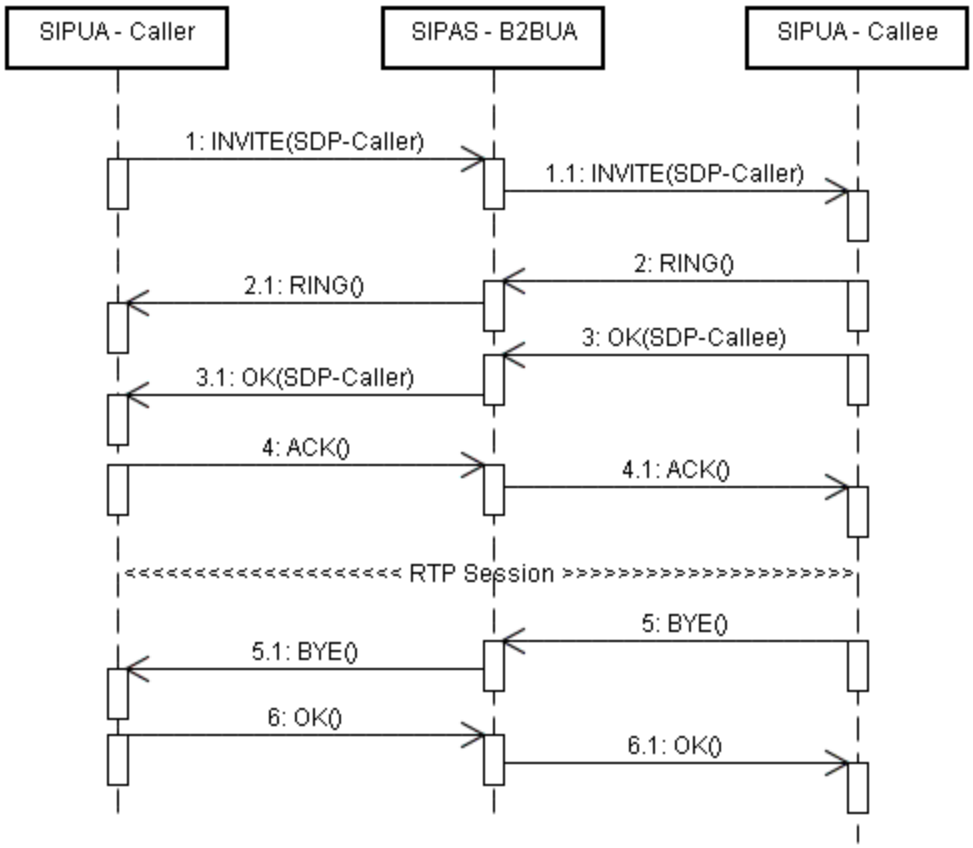

Figure 2 B2BUA Basic Call Flow

As illustrated in Figure 2, a B2BUA involves a Caller SIPUA triggering the signalling and a callee SIPUA being dialled out. From this flow, we see that the messages sent from one SIPUA do not propagate directly to the other side; they are first intercepted by the SipServlet running as a B2BUA.

Note that the media options (SDP) sent by the caller are propagated to the callee when the callee is dialled out (message “1.1” - INVITE in Figure 2), and the callee sends its SDP when it answers the call (message “3” - OK in Figure 2) which is propagated to the caller (message “3.1” - OK in Figure 2).

Although in this basic example the SIPAS is simply mirroring the signalling to the other side, this is not necessarily always the case, and we will actually see a non-mirroring (non-symmetric) B2BUA signalling when we soon analyse the state of the art in section 4.1, where the SIPAS will be positioned between the caller and the MS, but

specific SIP INFO signalling will be exchanged between the SIPAS and the MS only in order to provide the caller with an IVR.

2.2.8 Third-Party Call Controller (3PCC)

3PCC is also a SIPAS signalling pattern, similar to B2BUA. However, instead of having one of the SIPUA being a caller, a 3PCC exchanges the media options of two SIPUA acting as a callee. In our specific study, this is important if we want to dial out a number and place it in an IVR.

Figure 3 shows a basic 3PCC call flow. Note the different way of exchanging the media options on a 3PCC scenario, where the CalleeA is initially invited with no media options, but as soon as it answers, its media options are directed to CalleeB via the INVITE message, and once the CalleeB answers its media options are sent to the CalleeA along with the ACK message (message "5" - ACK in Figure 3).

From an IVR perspective, the automata SIPUA (the UA that answers automatically) would be the second one to be invited. That is, CalleeA would be an end-user phone, and CalleeB would be a media server (the automata one).

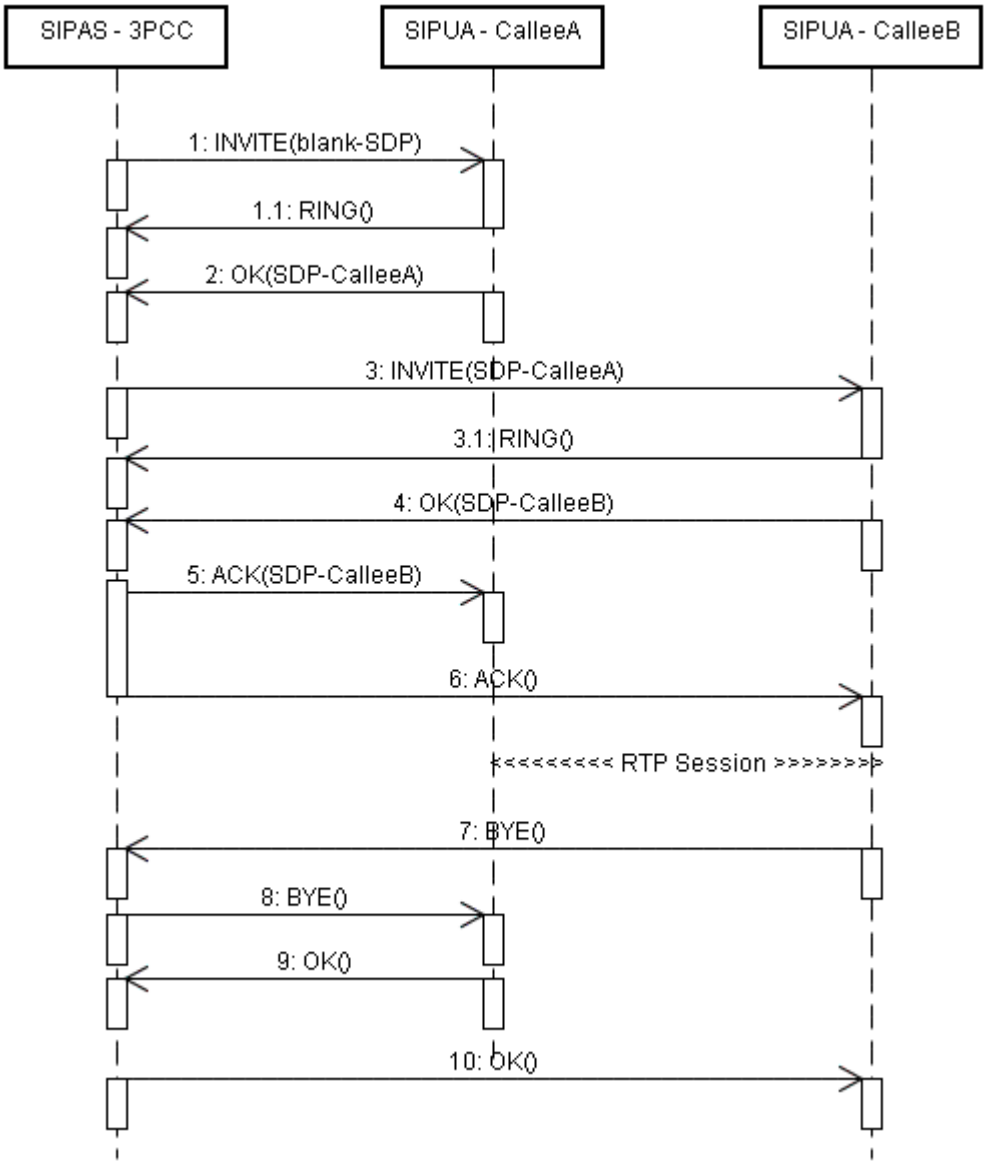

Figure 3 3PCC Basic Call Flow

2.3. Minimum Common Denominator (MCD) SIPUA

A SIPUA acting either as a caller or callee and interacting with the SIPAS has the requirement of supporting only the basic SIP RFC-3261 [20] and of being able to send Dual Tone Multi-Frequency (DTMF) via the established RTP session.

In other words, in order for a phone to be able to interact with the IVR, it is only expected from a phone to support what we will be calling here as the *minimum common*

denominator (MCD) of the possible features a phone can have. Therefore, a phone only needs to be able to: dial (via SIP), ring (via SIP), connect (via SIP), talk (via RTP), send DTMF (via RTP), and disconnect (via SIP). So, if a phone has extra features such as conferencing, call transfer, 3-way calling, call display, voice-mail, or has a super color display that can make use of SIP UPDATE [19] messages to retrieve emails and check user presence information, then this is all irrelevant for the purpose of this study.

The idea behind the MCD approach is to be able to support the greatest number of phones. Imagine an IVR from a bank; can it require that only phones that have special features A and B are able to access its system? Obviously, the bank IVR has to support and expect nothing more than the "MCD" phones, even if A and B are defined by standard bodies.

It is not the intent of this study to limit or support any specific SIPUA device or vendor (on the contrary: portability is a key objective for this study). So the caller should be able to reach the SIPAS from a phone attached to a terminal adaptor (TA [25]) of any vendor, from a landline of any Public Switched Telephone Network (PSTN) provider, or from a cell phone of any provider, regardless of whether the signalling crosses a SIP-PSTN gateway or a Session Border Controller (SBC), again or any vendor.

The next section provides an architectural view of the possible supported IVR users.

2.4. Big Picture – Supported Users

Figure 4 provides an overview of the possible paths and devices a user can utilize to interact with the IVR. There are some points to note in that figure:

- The arrows show the signalling, not the RTP session; the RTP will flow directly between the SIPUA and the MS.
- SIPAS is the orchestrator; all the SIP signalling goes to it, and it decides where to route signals according to its application business logic.
- Phones can be of different types (including non-SIP ones), but at some point along the network the signalling will be converted to SIP (via some sort of gateway) so SIPAS can handle it.

- The MCD is all that is expected from the end-user's phone.

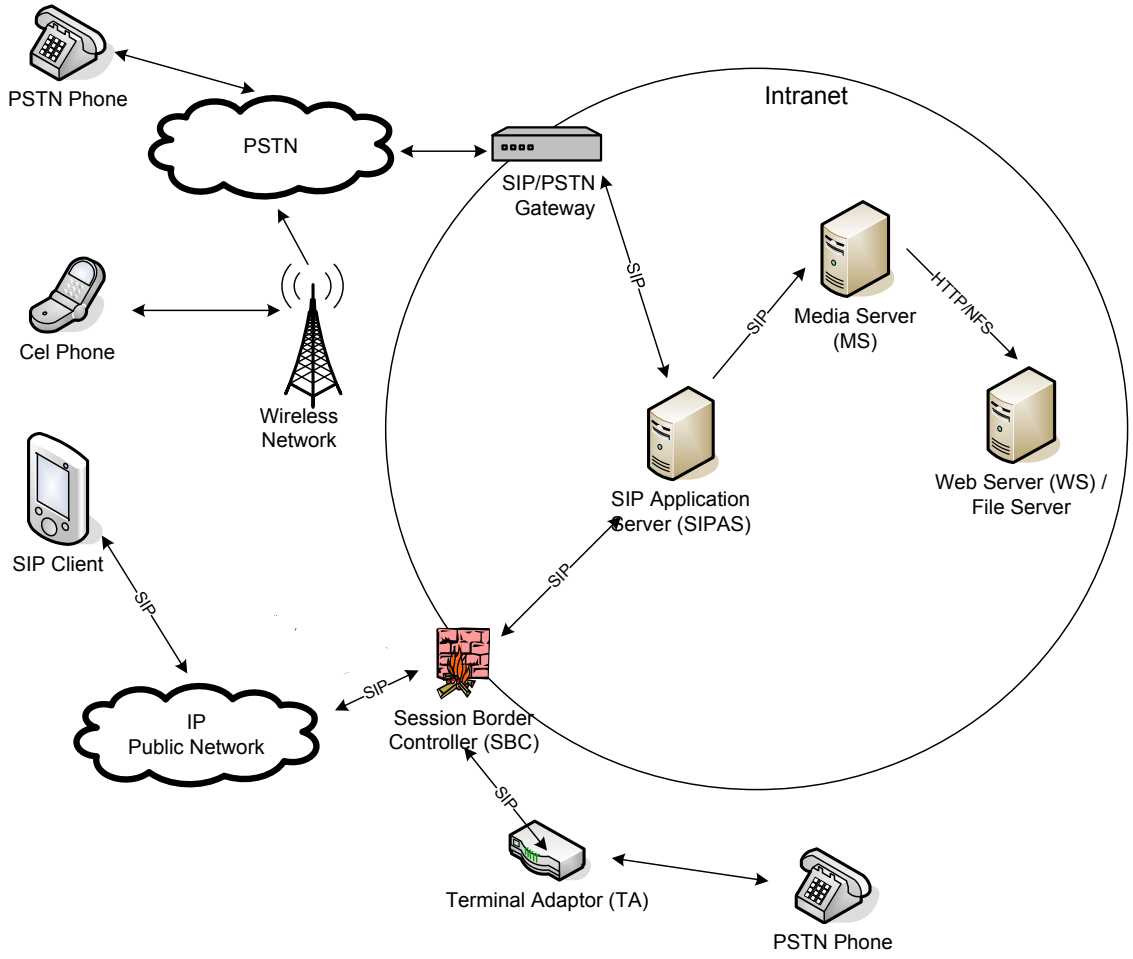

Figure 4 Big Picture

2.5. IVR with SIP Signalling Controlled by a SIPAS

As illustrated in Figure 4, there are typically three components required for an application hosted in a SIPAS to provide IVR capabilities to an end-user SIPUA:

- The SIPAS itself, where the application logic (composed of business logic and signalling handling) is deployed.
- The MS, where the IVR commands are interpreted, executed and the appropriate prompts are played, DTMF is collected, and audio is recorded.

- The Web Server (WS) that is used as a repository of prompts (this usage will be further explored when we take a closer look at one of the state-of-the-art alternatives in section 4.1) and/or as a repository of the Web pages that will generate the VoiceXML interactions (this usage will be further explored in section 4.2 once we further detail a second state-of-the-art alternative).

SIPAS will then be the orchestrator of the SIP signalling, so the required IVR sequence is directed to the end-user SIPUA. The application running on the SIPAS will make use of the B2BUA (section 2.2.7) and/or 3PCC (section 2.2.8) signalling patterns to facilitate the establishment of the RTP session between the MS and the end-user SIPUA, and hide the signalling from each other.

2.6. Chapter Summary

In this chapter, we had the opportunity to review several concepts, protocols, and network elements with the intent to ground the reader on the important areas around this study and to facilitate the understanding of the sections to come.

We reviewed the main protocol we will be using, namely SIP. We identified the role of a SIPAS and how an IVR application is developed using the SipServlet API. We also discussed two signalling patterns (B2BUA and 3PCC) that are commonly used in SIPAS applications to connect an end-user SIPUA with a SIP-enabled MS. We also noted that he MS is the network element that ultimately ends the media path established directly to the end-user phone, as the talking path (the RTP session) flows directly between the SIP endpoints and does not traverse the SIPAS.

The next section will define a set of evaluation criteria and a use case that will be referenced throughout the thesis. The use case will later be prototyped for two popular approaches, as well as for this thesis' proposed alternative (IVRObject). The evaluation criteria will then help us analyse these three approaches based on a common ground of comparison, which in turn will ease the drawing of relevant conclusions.

Chapter 3. Evaluation Criteria and Use Case

This chapter defines a set of evaluation criteria (section 3.1) and a use case (section 3.2) that will be used first against the current state of the art in Chapter 4 and then against our proposed alternative in Chapter 5.

This will provide a common ground for comparing the different approaches that target the provisioning of IVR capabilities to an end-user SIPUA (caller or callee) that has its SIP signalling controlled by an application running in a SIPAS, and have the media streamed by a SIP-enabled MS.

"IVR capabilities" means to provide a way for: a) playing a prompt to the end-user telephone, b) collecting DTMF input from the end-user via the telephone keypad, and c) recording the end-user audio/voice spoken via the telephone to a file, or a combination of the above.

3.1. Evaluation Criteria

The different approaches will be evaluated against 3 criteria: a) how easy it is to develop an IVR application, b) how easy it is to port it to run in different MS vendors, and c) how much network bandwidth impact the solution has for its signalling in comparison to the total load during the IVR call.

In order to help the comparison of the different alternatives, the following scoring scheme is defined:

- 1: one point means a low mark during the evaluation as it is considered poor, insufficient, complicated, or slow.
- 2: two points means a neutral mark during the evaluation as it is consider acceptable.
- 3: three points means a high mark during the evaluation for its positive nature.

Next, each of the three criteria is detailed and the point system applicability defined.

Criterion-1) Ease of Development

The different IVR implementation alternatives will be analysed from five different perspectives in order to understand how *easy* it is to develop them:

Simplicity of IVR Command Request Generation and Response Parsing

This will help evaluate the effort, from a developer's perspective, required to program an IVR, taking into consideration the parsing and generation of the commands embedded in a SIP message or by other means.

In section 2.2.2, we saw the structure of a SIP message. The SipServlet API provides a mechanism for building and parsing the SIP message, except for the body, where it is up to the application to interpret the payload.

Some of the alternatives we will analyse rely on XML payloads to be generated and/or parsed by the application (running on SIPAS or in the WS), and the following point scheme will be applied during the evaluation:

- 1 point: if parsing and generation of SIP and of its XML payloads are required for IVR commands.
- 2 points: if only the generation of XML is needed, but no SIP handling is needed.
- 3 points: if handling parsing and generation of SIP and XML are not required at all from the application perspective.

Signalling Simplicity

This will evaluate the quantity of signalling needed from the SIPAS to the MS in order to provide the IVR.

Some of the biggest challenges in stabilizing an application (i.e. getting rid of bugs) hosted in SIPAS are the so-called *corner cases*. Corner cases are common in SIP-based applications due to the bidirectional and asynchronous nature of the SIP messages highlighted in section 2.2.6, which causes messages to come from any SIP endpoint at any time. Messages can also cross each other, which makes

the number of possible combinations of states and transitions to be handled quite challenging for a developer.

The alternative that requires the fewest messages is then the preferred choice. The following points scheme will be applied during the evaluation:

- 1 point: if INFO messages are required for instructing the MS what IVR commands to execute, causing a high and granular SIP traffic.
- 2 points: if SIP is only used for call setup, but non-SIP means are required to provide partial IVR update results back to the SIPAS.
- 3 points: if SIP is only used for call setup, and non-SIP messages are only used once at the end of the IVR interaction to report the result as a whole.

Ease of SIP Unit Testing

This will evaluate the effort needed by a developer to run call simulations for functional tests.

SIP unit testing is a common practice for exercising funtional tests and validating the health of a SIP application during its development. SIP-based applications can benefit from tools like SipUnit [16] to facilitate this testing-oriented development approach, where the end-user SIPUAs that directly interact with SIPAS are substituted for mock ones. In other words, instead of having a real caller SIPUA, a real callee SIPUA, and a real media server, and then conduct manual tests, the idea behind SIP unit testing for SIPAS-based applications is to substitute these elements for a mock-Caller-UA, a mock-Callee-UA and a mock-MediaServer. These mock elements are fully functional SIP stacks that are controlled via an API and orchestrated via the different test cases to accomplish the different test scenarios and indirectly test the application deployed in the SIPAS (in a black-box testing fashion).

Being able to conduct SIP unit testing during the development is a key strategy that improves development quality and efficiency, allowing developers to handle the interesting call scenarios, to constantly run regression tests to check the application, and to quickly address side-effect bugs that could otherwise be left unnoticed. Such approach also simplifies the overall application maintenance.

The alternative that requires the least effort to make SIP unit testing possible would be the preferred choice. The following points scheme will be applied during the evaluation:

- 1 point: if the effort needed to be able to conduct unit testing is so high and requires so much development investment in making it possible that it is usually avoided all together.
- 2 points: if unit testing is possible, but not easily achievable and requires a combination of SIP and HTTP simulation to mock the user behaviour.
- 3 points: if unit testing is possible and simple.

Central Development

This will evaluate whether development is needed only in SIPAS, i.e. in a single (central) place, or in two different entities (in the WS and in SIPAS). Central development is preferable as this requires expertise from developers in just one area this requires fewer integration points, which often increase development time.

The alternative that allows central development would be the preferred choice. The following points scheme will be applied during the evaluation:

- 1 point: if the application logic is split, and central development of signalling and business logic is not possible.
- 2 points: *not applicable*, as either an alternative will allow central development or not.
- 3 points: if central development is possible.

Call-back Mechanism

This will evaluate the existence of a communication mechanism between the MS and the SIPAS for reporting IVR results (example: prompt played, DTMF collected, and prompt recorded).

Once an IVR asks the user to enter an extension number, for example, this criterion will help measure how easy this information can get to SIPAS so it can act on it according to the application business logic hosted in SIPAS.

The alternative that has a built-in call-back mechanism would be the preferred choice as this would prevent the developers from having to create their own mechanisms. The following points scheme will be applied during the evaluation:

- 1 point: if the alternative does not have a call-back mechanism and the application developer has to create/define one.
- 2 points: *not applicable*, as either an alternative will have a built-in call-back mechanism or not.
- 3 points: if a call-back mechanism is part of the alternative and the application developer can simply use it.

Criterion-2) Portability

This criterion will evaluate how easy it is to port an application to a different media server.

It is desirable that once an application is coded, it can be run on a different media server vendor without requiring the application to be rewritten.

During this study, we will see some alternatives that are more portable than others. Some might use proprietary means (or rely on a standard adopted only by few) to achieve the IVR capabilities while others rely on mechanisms defined by standards bodies and more widely accepted by different MS vendors.

The following point scheme will be applied during the evaluation:

- 1 point: if the alternative is not easily portable as it relies on a mechanism that is MS vendor dependant.
- 2 points: if this alternative relies on a mechanism defined by a standards body, and is well accepted, but has possible portability issues due to versions / flavours of the defined standard.
- 3 points: if this alternative relies on a mechanism defined by a standards body, is well accepted, and also abstracts the version / flavour issues from the application developer.

Criterion-3) Signalling Load

This will evaluate the network bandwidth impact for the signalling needed while using each of the approaches in comparison to the total load during the IVR call, including the network bandwidth required for the streaming of the RTP packets.

It is desirable that an application requires a low network load for its signalling in order to have its different parts (SIPAS, MS, and WS) communicate. A heavyweight application will have difficulties to run under load and be able to adequately cope with a high call volume environment.

This evaluation is different from the others as the goal here is to quantify the load capacity of the various alternatives, to enable comparisons once they are all known.

A different scoring scheme will be defined here. For each prototype to be developed from each approach being analysed in this study, a call will be run according to the use case detailed in section 3.2. For each alternative, we will capture, using the Wireshark network traffic analyser [7], the number of bytes needed to run a single call from a signalling perspective and from an RTP perspective.

The RTP load is dependent on the codec that gets exchanged between the parties, in our case, on the codec exchanged between the caller and the media server for the streaming of the IVR. Two popular codecs are the G.711 and the G.723.1, and they require 87.2 kilobits per second (Kbps) and 21.9 Kbps, respectively (see [6] for details).

For our evaluation we will consider the average bandwidth for these two codecs, and consider an IVR call duration of 25 secs, i.e. ((87.2 Kbps + 21.9 Kbps) /2) * 25 s. This amounts to 1,363.7 kilobits (Kb), which is equivalent to 174,560 bytes (1024 * 1363.7 b / (8 b/byte)) for a single call of 25 secs. Let IVRRTPLoad be a constant for this value (174,560 bytes).

For our evaluation we will capture the number of bytes required for the IVR signalling control (the IVRSignallingLoad) and apply the following formula:

SignallingLoad = 100% * (IVRSignallingLoad / (IVRRTPLoad + IVRSignallingLoad))

The score will hence be the SignallingLoad value. Note that this score works in the opposite direction from the others: the lower the percentage number, the better it is.

3.2. Use Case

The following Auto-Attendant use case will be prototyped for the current state-of-the-art approaches (sections 4.1 and 4.2) and the thesis proposed approach (Chapter 5). These prototypes will be compared under the evaluation criteria defined in section 3.1.

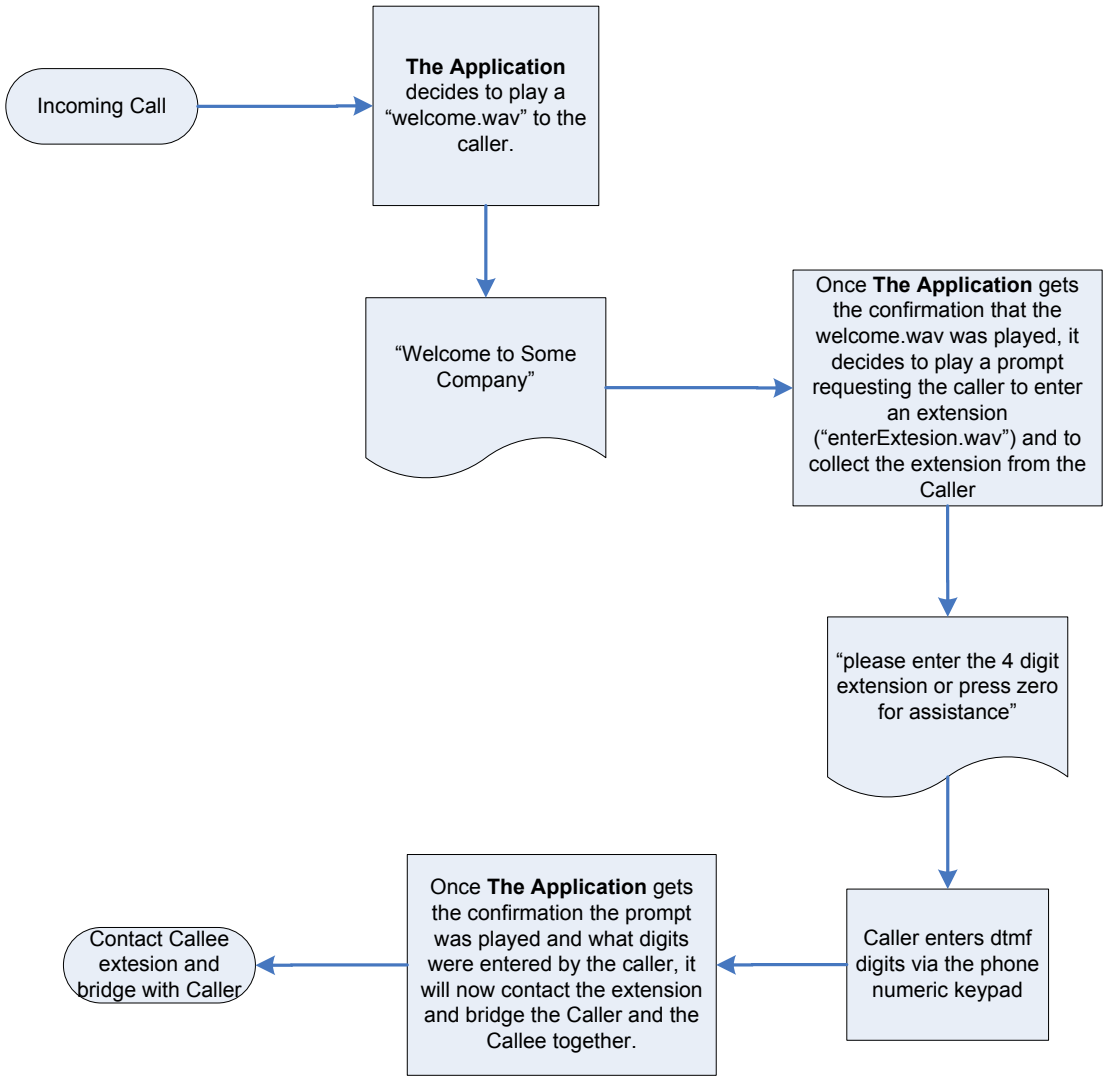

Figure 5 Auto-Attendant Flow Chart

This scenario represents a popular auto-attendant use case where the following steps are involved:

- An end-user dials in and reaches SIPAS.
- SIPAS is running a previously installed auto-attendant application.

- The auto-attendant application orchestrates the SIP signalling in order to connect the calling end-user SIPUA to the MS by making usage of a B2BUA (as described on section 2.2.7). This signalling pattern will allow a talking path (RTP session) to be established directly between the end-user and the MS, while keeping SIPAS in between them from a SIP signalling perspective, which prevents the end-user and the MS from having to know how to directly communicate via SIP.
- The auto-attendant plays a welcome prompt – the actual streaming of the audio is done by the MS directly to the end-user.
- The auto-attendant requests the end-user to enter the extension number – again, the actual streaming of the audio is done by the MS.
- The auto-attendant captures the DTMF input entered by the end-user via its telephone keypad – the actual capturing of the digits is done by the MS.
- MS lets the SIPAS know of the collected DTMF, representing the callee number to be dialled out.
- The auto-attendant disconnects the MS, as it is no longer needed – the IVR part of the call is over.
- The auto-attendant application contacts the extension requested by the end-user and bridges (via SIP signalling manipulation) the caller to this new callee end-user SIPUA so a new talking path can be established between the caller and the callee.

This scenario is believed to be ideal for the analysis of the different alternatives detailed along this study for several reasons:

- It is simple, and the call flows to be soon detailed are easy to follow, which facilitates the understanding of the ideas and concepts explained in this study.
- It is a well-known scenario, which also facilitates the overall understanding
- It provides the need for call-backs so the SIPAS application can act upon it, in this case the call-back carrying the extension entered, so SIPAS can bridge the caller to the callee and a new talking path can be established.

- It provides a chance to showcase how related IVR commands can be bundled for sequenced operations that do not require business logic between them, such as play prompt and collect DTMF, which will illustrate a key difference among the alternatives where some of them are capable of taking advantage of this and have the signalling optimized.
- It provides a chance to exercise all of the evaluation criteria defined in section 3.1, enabling us to compare the different alternatives, measure them accordingly, and draw the conclusions required for this study.

3.3. Chapter Summary

This chapter has defined a set of evaluation criteria (section 3.1) that will be used to compare different approaches for developing IVR applications. This set is composed of 3 categories:

- Criterion-1) Ease of Development
- Criterion-2) Portability
- Criterion-3) Signalling Load

Criterion-1 is itself analysed from 5 different perspectives:

- Simplicity of IVR Command Request Generation and Response Parsing
- Signalling Simplicity
- Ease of SIP Unit Testing
- Central Development
- Call-back Mechanism

All the different alternatives will be prototyped by implementing the Auto-Attendant use case defined in section 3.2. This will allow us to provide a common measurement for the different alternatives, to compare them, and to draw conclusions.

The next chapter will analyse two state-of-the-art alternative approaches, one using M*ML, and the other VoiceXML.

Chapter 4. State of the Art in IVR Development

This chapter describes and evaluates two popular approaches for providing IVR capabilities to an end-user SIPUA (caller or callee) that has its SIP signalling controlled by an application running in a SIP application server, using a SIP-enabled media server.

Section 4.1 details the M*ML approach whereas section 4.2 details the VoiceXML one.

4.1. Using M*ML-Enabled MS

This section describes the usage of M*ML as a way for SIPAS to instruct the MS about what IVR commands should be directed to the end-user SIPUA (caller or callee). By M*ML, we mean to cover the following approaches:

- The *Dialogic* media server (and compatible servers) with the Media Server Control Markup Language (MSCML) [10].
- The *Radisys* media server (and compatible servers) with the Media Server Markup Language (MSML) [23] combined to the Media Objects Markup Language (MOML) [22].

The term *M*ML* is being used to generalize the above aproaches, as from an evaluation perspective both will be treated as a single type due to their almost identical SIP signalling, and the way they carry some sort of MS control XML (Extensible Markup Language) payload.

Burger [4] comments on the similarities of the protocols behind M*ML: "Interestingly, MSML and MSCML exchange the same number of messages to do the same task". Burger also clarifies the difference between them, which relies on the particular way each protocol preserves the SIP semantics [4]:

- "MSCML uses the SIP Requires and Content-Type headers to ensure interoperability and preservation of SIP semantics. MSCML correlates the commands received on the dialog with the dialog's media streams."

- "MSML relies on a private (non-Internet) agreement between the Application Server and Media Server to know the context of the INFO messages. MSML tunnels SDP-layer information over the established dialog; in the case of media processing, it uses a secondary markup, MOML. MOML is a device control protocol".

The difference mentioned is related to SIP semantics and considered minor from this study's perspective. M*ML will then be considered simply as some XML scripting language that is embeded in the body of SIP messages exchanged between the SIPAS and the MS. These XML messages provide a way of defining a protocol within the SIP protocol to achieve the required IVR capabilites. SIP INFO messages and the 200-OK (for the INFO message) are used to carry this XML payload.

For this alternative, the Web Server (WS) is used only as a prompt repository (and for some installations it can also be substituted for a file server).

4.1.1 Deployment View

Figure 6 shows the deployment view for the network elements and for the modules that make part of a custom application (the Application) in a typcial solution involving the usage of a M*ML-enabled MS and SIPAS.

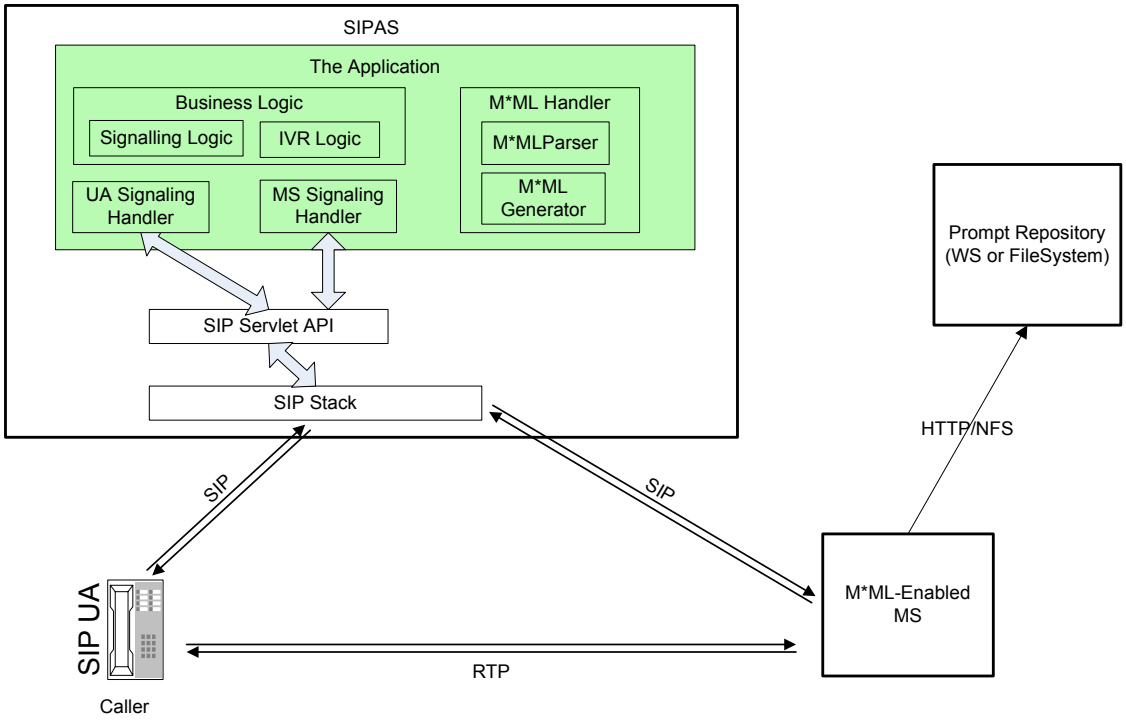

Figure 6 M*ML Alternative - Deployment View

Note in Figure 6 that the application logic is centralized in SIPAS.

4.1.2 Use Case Call Flow

Figure 7 details the SIP signalling required in order to implement the use case defined in section 3.2 using M*ML.

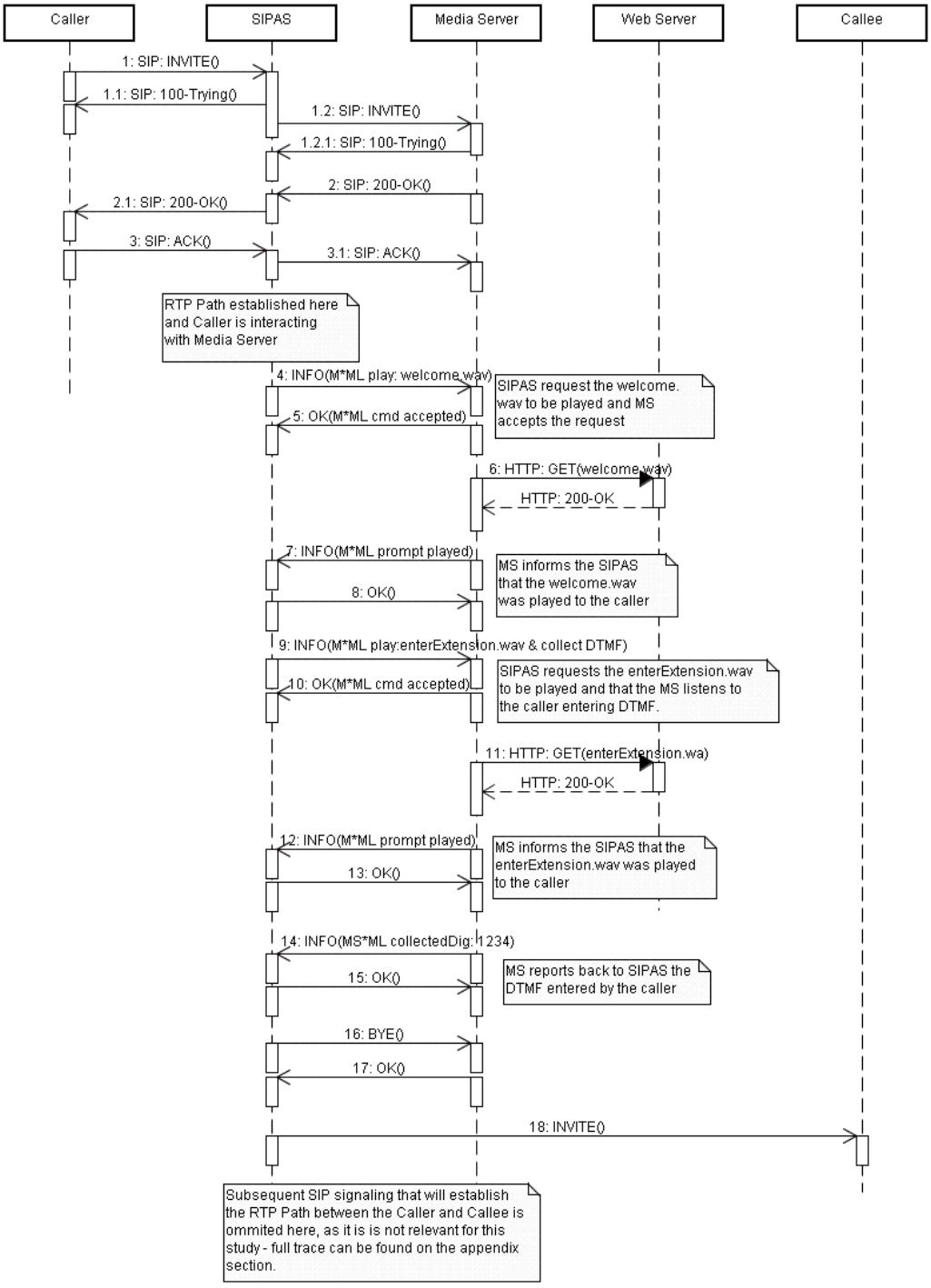

Figure 7 IVR to Caller using M*ML with SIP INFO Call-back

One important aspect to note is that the minimum common denominator (MCD concept, see section 2.3) is in place here for the Caller: it is a simple SIPUA, and it is totally un-aware of the complex SIP signalling being used to reach the MS via the SIPAS. This can

easily be seen by looking at the messages that are exchanged between Caller and SIPAS in Figure 7.

4.1.3 Runtime Data

In order to help understand the peculiarities of this aproach, and to assist with its evaluation, runtime data were collected for a prototype developed for the use case specified in section 3.2, using M*ML, and sumarized next. For the full SIP signalling trace captured during this test, please refer to Appendix A. For an overview of the high-level signalling, please refer to Figure 7.

SIP and HTTP Network Traffic Analysis

The following diagram shows the network traffic data captured for both SIP and HTTP protocols from the MS perspective while running the prototype for this M*ML approach.

Legend:

- X-axis: time, in seconds
- Y-axis: traffic, in bytes
- Blue bar:
  SIP signalling
- Red bar:
  HTTP signalling

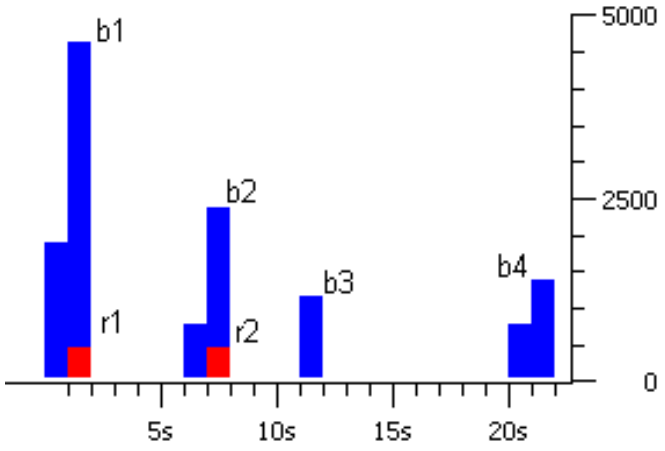

Figure 8 SIP and HTTP Network Traffic (from MS Perspective)

In order to facilitate the correlation of the messages exchanged between the SIPAS and the MS, detailed in Figure 7, and the network analysis diagram in Figure 8, the following table is provided.

Table 3 SIP and HTTP Traffic for M*ML using SIP INFO Call-back

Point in Time	Details
b1	This is the SIP call set up between SIPAS and the MS, and the INFO request for playing the welcome.wav prompt. This corresponds to the following steps in Figure 7: 1.2, 1.2.1, 2, 3.1, 4 and 5
b2	This is the INFO call-back notification that the welcome.wav prompt was played, and the new INFO requests to both play the enterExtension.wav and collect the DTMF from the Caller. This corresponds to the following steps in Figure 7: 7, 8, 9, and 10.
b3	This is the INFO call-back notification that the enterExtension.wav prompt was played, and the new INFO call-back notification for the DTMF keys collected from the Caller. This corresponds to the following steps in Figure 7: 12, 13, 14, and 15.
b4	Termination of MS. This corresponds to steps 16 and 17 in Figure 7.
r1	This is the HTTP request to load the welcome.wav. This corresponds to step 6 in Figure 7.
r2	This is the HTTP request to load the enterExtension.wav. This corresponds to step 11 in Figure 7.

The following table details the bandwidth usage for SIP and HTTP messages.

Table 4 SIP and HTTP Bandwidth Usage

	Total Bytes	Number of Messages	Avg Bytes/Message
SIP (i)	10,949 (92.7%)	16 (80%)	684.3
HTTP (ii)	864 (7.3%)	4 (20%)	216.0
Total	11,813	20	590.6

(i) Between SIPAS and MS

(ii) Between MS and WS

4.1.4 Evaluation

Table 5 summarizes the evaluation of the M*ML approach for the 3 criteria:

Table 5 Evaluation summary for M*ML

Criteria	Score
Criterion-1) Ease of Development	1.8 (avg)
Criterion-2) Portability	1
Criterion-3) Signalling Load	6.3%

Figure 9 shows once again the deployment view, but now highlighting the positive and negative aspects around the different elements. The checkmark (✓) indicates a high score and the × means a low score for a given aspect being analysed. The numbering indexes will be referenced along the remaining part of this section where a detailed evaluation is provided.

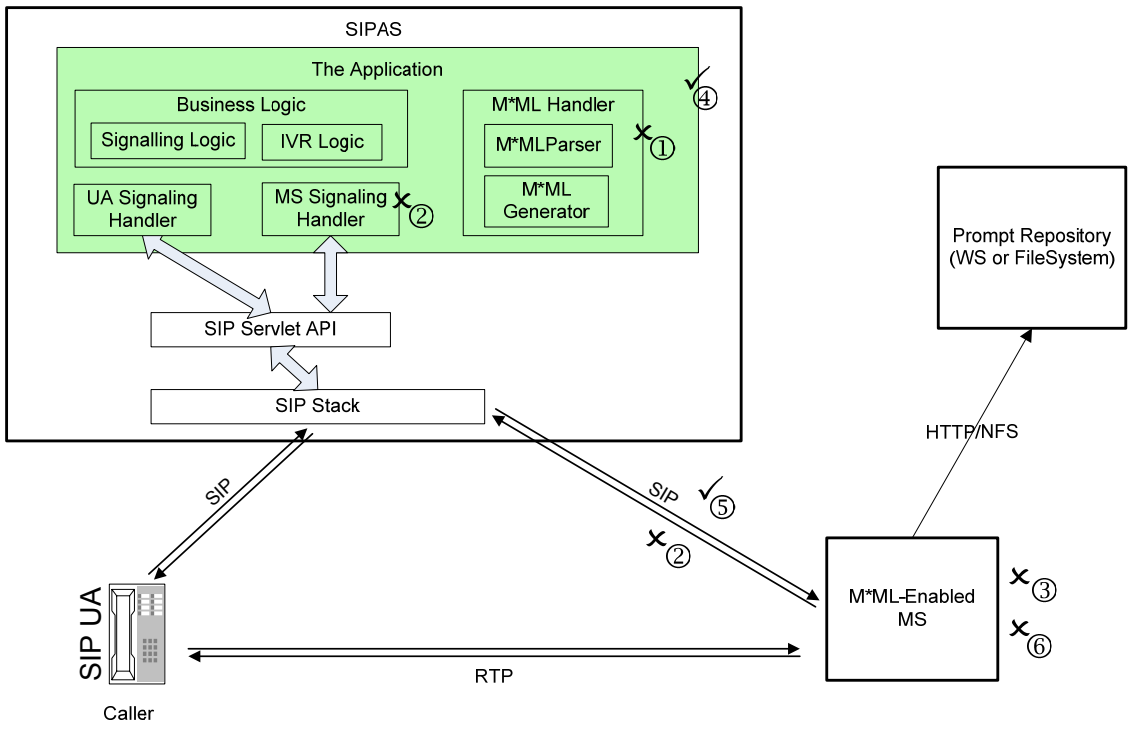

Figure 9 M*ML Alternative - Deployment View – Evaluation Summary

Next, each of the criteria is analyzed in detail, and given a proper score:

Criterion-1) Ease of Development

Simplicity of IVR Command Request Generation and Response Parsing

Evaluation mark: 1 point (Low).

The M*ML approach is complex. It is quite powerful (as it supports also non-IVR functionalities such as conference control), but overly complex if the only target functionality is to provide IVR capabilities. In addition, it makes the application hosted in the SIPAS difficult to code as it requires developers to parse/generate the M*ML code embeded within the SIP messages.

For detailed SIP signalling, and more specifically on the M*ML code used during this prototype, please refer to Appendix A, and more specifically to the SIP INFO messages and to the SIP OK messages that follow each of the SIP INFO messages.

This evaluation can be visualised in Figure 9 under index number ①.

Signalling Simplicity

Evaluation mark: 1 point (Low).

The less SIP signalling used, the easier it is to make an application stable (solid / with few or no bugs). Yet, this is not easily achievable using M*ML, which relies heavily on INFO messages to drive the IVR commands. The M*ML is then a protocol (IVR command protocol) within the SIP protocol. As an example: a simple request for playing a prompt produces 4 messages (illustrated here in plain English instead of M*ML for easier reading):

a) "play this prompt" (from SIP application server to media server in a INFO message), see message 4 in Figure 7 ;
b) "yes I will play" (in the 200-OK message, from the media server back to SIP application server), see message 5 in Figure 7;
c) "prompt has been played" (in a INFO message from the media server to SIP application server), see message 7 in Figure 7;
d) "thank you" (in the 200-OK message from SIP application server to the media server), see message 8 in Figure 7.

This evaluation can be visualised in Figure 9 under index number ②. This index appears twice in the diagram; this is simply to highlight that this impacts both the SIP traffic between the MS and the SIPAS, as well as the module inside the Application that has to handle the signalling itself.

Ease of SIP Unit Testing

Evaluation mark: 1 point (Low).

The fact that the MS expects M*ML makes the development of unit tests difficult, as a mock-MS would have to implement an M*ML parser to validate the commands sent by the SIPAS, and this parser would be specific to each of the M*ML flavours. As a

consequence, applications that use M*ML do not often use this testing approach in order to exercise functional tests via SIP endpoint simulation, as a high development investiment would have to be done just to make it possible.

This evaluation can be visualised in Figure 9 under index number ③, where it is challenging to mock an M*ML-enabled MS.

Central Development

Evaluation mark: 3 points (High).

The whole application logic is developed and run in the SIPAS only. There is no specific development needed in the MS or in the WS.

This requires less skill sets from developers (as they do not need to develop Web pages, for example). This approach also facilitates development as it does not introduce integration points between different teams that use different techonologies.

This evaluation can be visualised in Figure 9 under index number ④.

Call-back Mechanism

Evaluation mark: 3 points (High).

The M*ML provides direct communication between the SIPAS and MS. Any value entered by the IVR user or any other media related event (such as prompt played or prompt failed to be played) are detected by the MS and immediately reported back to the SIPAS via an INFO message. Please refer to messages 7, 12, 14 in Figure 7 for the exact moment where the SIP INFO messages are used by the MS to call back SIPAS in order to notify it of the prompts played and the DTMF digits collected.

This evaluation can be visualised in Figure 9 under index number ⑤; the call-back is built in the SIP signalling and part of the M*ML protocol from the start.

Criterion-2) Portability

Evaluation mark: 1 point (Low).

As mentioned before, M*ML represents different XML formats, and this difference causes portability issues. An application that uses MSCML is written with the intent of running on the Dialogic media server (and compatible ones) only, and cannot run on a Radisys (and compatible) one, which would require MSML.

This evaluation can be visualised in Figure 9 under index number ⑥.

Criterion-3) Signalling Load

Evaluation mark: 6.3%.

From Table 4, we can see that the total number of bytes needed to fulfil the use case using M*ML was 11,813 bytes, this value represents the IVRSignallingLoad.. Applying the formula detailed in section 3.1, this alternative gives us: SignallingLoad = 100% * (11,813 bytes / (174,560 bytes + 11,813 bytes)) = 6.3%.

4.2. Using VoiceXML-Enabled MS

This section describes the usage of VoiceXML [17], or simply VXML, as a way for SIPAS to instruct the SIP-enabled MS about what IVR commands should be directed to the end-user SIPUA (caller or callee).

VXML is a standard defined by W3C [26]. It is an XML-based language that describes what IVR commands to execute. Unlike M*ML, VXML is generated by the WS and not by SIPAS, but the VXML is also interpreted by the MS. Also, unlike M*ML, VXML is a rich language that contains logical operators, decision blocks and variables to store temporary values.

In order to request IVR capabilities, the SIPAS simply sends an INVITE to the MS passing in a header (the Request URI) a HTTP URL as a parameter, which points to the WS that will generate the VXML page. Once the MS gets the INVITE, it extracts this URL and issues the HTTP request to the Web server which then generates the VXML code and returns it back to the MS to be interpreted and run.

VXML pages are generated then by HTTP Servlets much the same way that HTML [18] pages are generated by HTTP Servlets. The difference relies on the fact that while HTML pages are parsed by a Web browser, the VXML ones are parsed by a voice browser that is embedded in the VXML-enabled MS.

4.2.1 Deployment View

Figure 10 shows the deployment view for the network elements and for the modules that make part of a custom application (the Application) in a typical solution involving the usage of a VXML-enabled MS and SIPAS, where the SIPAS is the SIP signalling orchestrator.

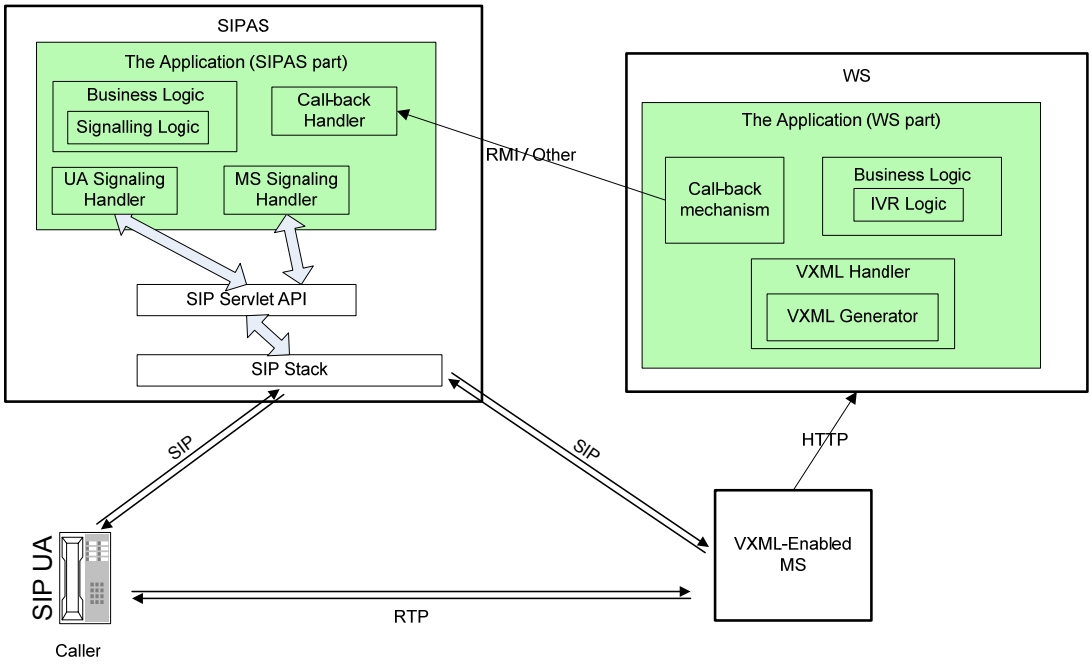

Figure 10 VXML Alternative - Deployment View

Note in Figure 10 that the application logic is split and has to be deployed and run simultaneously from the SIPAS and from the WS.

Due to the lack of a call-back mechanism, an out of band mechanism has to be defined by the developer in order to provide IVR call-back. In this experiment, RMI is used, but any other technology that can be mutually agreed by the WS and SIPAS could be used instead.

The VXML-enabled MS issues HTTP requests to the WS in order to retrieve the VXML pages that are generated by the custom application via HttpServlets.

4.2.2 Use Case Call Flow

The following sequence diagram details the SIP and HTTP signalling required to implement the auto-attendant use case (defined in section 3.2), using VXML generation from a WS, and the use of RMI [14] for call-backs to SIPAS from the WS.

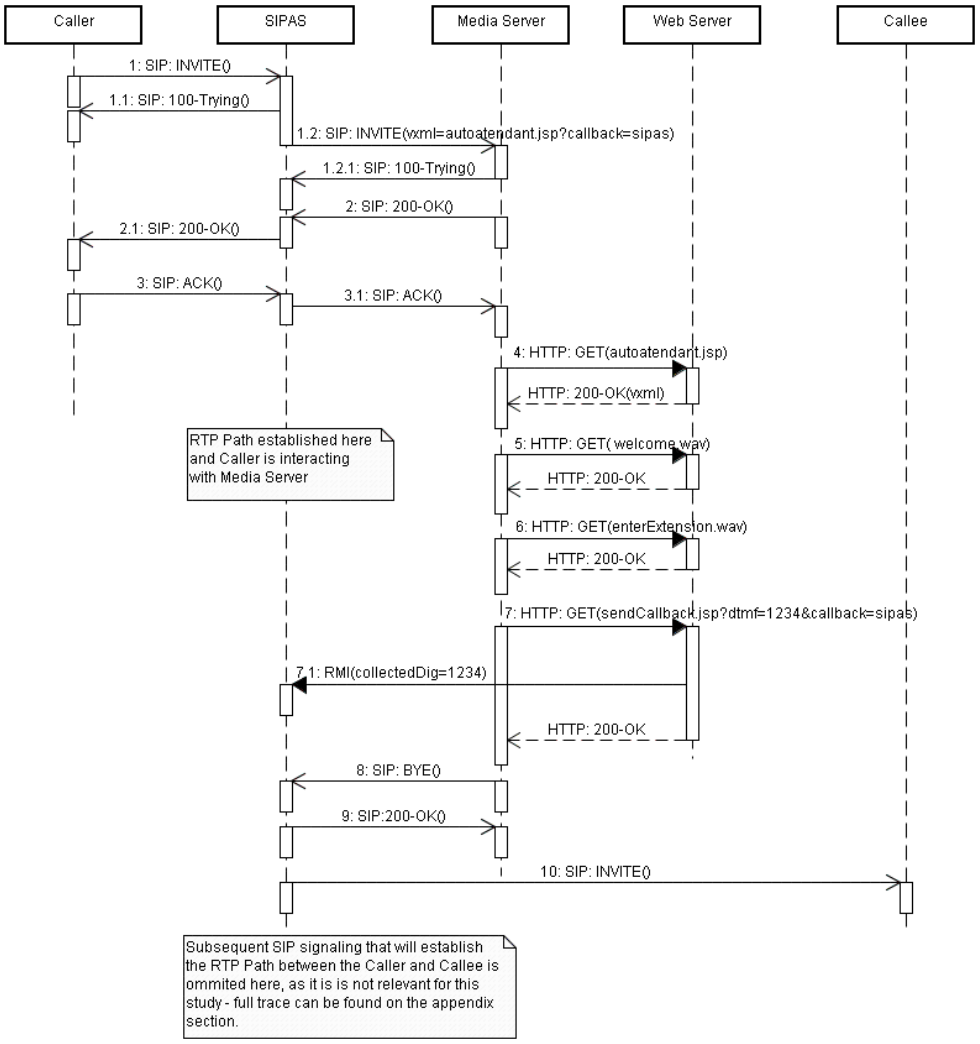

Figure 11 IVR to Caller using VXML with RMI Call-back

Once again, note here the MCD concept (see section 2.3) is in place here for the Caller; who is totally unaware of the signalling being used to reach the MS.

4.2.3 Runtime Data

In order to help understand the peculiarities of this aproach, and to assist with its evaluation, runtime data were collected for a prototype using VoiceXML and sumarized next. For the full SIP signalling trace captured during this test, please refer to Appendix B.

SIP, HTTP and RMI Network Traffic Analysis

The following diagram shows the network traffic data that was captured for both SIP and HTTP protocols from the MS perspective while running the prototype for this VoiceXML approach. Only messages in and out of the MS are captured here.

Legend:

- X-axis: time, in seconds
- Y-axis: traffic, in bytes
- Blue bar:
  SIP signalling
- Red bar:
  HTTP signalling
- Green bar:
  RMI signalling

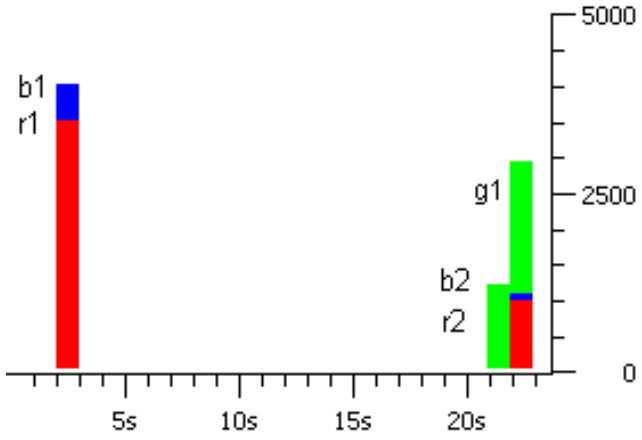

Figure 12 SIP, HTTP and RMI Network Traffic

In order to facilitate the correlation of the SIP messages exchanged between the SIPAS and the MS, for the HTTP messages between the MS and WS, and for the RMI messages between the WS and the SIPAS, detailed in Figure 11, and the network analysis diagram in Figure 12, the following table is provided.

Table 6 SIP and HTTP Traffic for VoiceXML

Point in Time	Details
b1	This is the SIP call set up from SIPAS to the MS. This corresponds to the following steps in Figure 11: 1 through 3.1
b2	Termination of MS. This corresponds to the following steps in Figure 11: 8 and 9.
r1	MS loading of the VXML from the WS that will greet the caller (loading of the autoattendand.jsp) and the prompts needed (welcome.wav, and enterExtension.wav). This corresponds to steps 4, 5 and 6 in Figure 11.
r2	MS invokes the WS Servlet to issue the call-back to SIPAS reporting on the data collected from the caller. This corresponds to step 7 in Figure 11.
g1	WS call-back to SIPAS via RMI. This corresponds to the step 7.1 in Figure 11.

Note that in Step 7.1 of the diagram in Figure 11 shows the RMI usage to report the collected digits by the MS back to SIPAS. The RMI call-back is issued by the WS via a HTTP Servlet.

The following table details the bandwidth usage for SIP, HTTP and RMI messages.

Table 7 SIP, HTTP and RMI Bandwidth Usage

	Total Bytes	Number of Messages	Avg Bytes/Message
SIP (i)	5,073 (37.3%)	7 (43.7%)	724.7
HTTP (ii)	4,456 (32.7%)	8 (50%)	557.0
RMI (iii)	4,094 (30.0%)	1 (6.3%)	4,094.0
Total	13,623	16	851.4

(i) Between SIPAS and MS

(ii) Between MS and WS

(iii) Between the WS and SIPAS

4.2.4 Evaluation

The following table summarizes the evaluation of the VXML approach for the 3 criteria:

Table 8 Evaluation summary for VXML

Criteria	Score
Criterion-1) Ease of Development	1.8 (avg)
Criterion-2) Portability	2
Criterion-3) Signalling Load	7.2%

Figure 13 shows once again the deployment view, but now highlighting the positive, neutral and negative aspects around the different elements. The checkmark (✓) indicates a high score, the crossed-circle (⦸) suggests a neutral score, and the × means a low score for a given aspect being analysed. The numbering indexes will be referenced along the remaining part of this section where a detailed evaluation is provided.

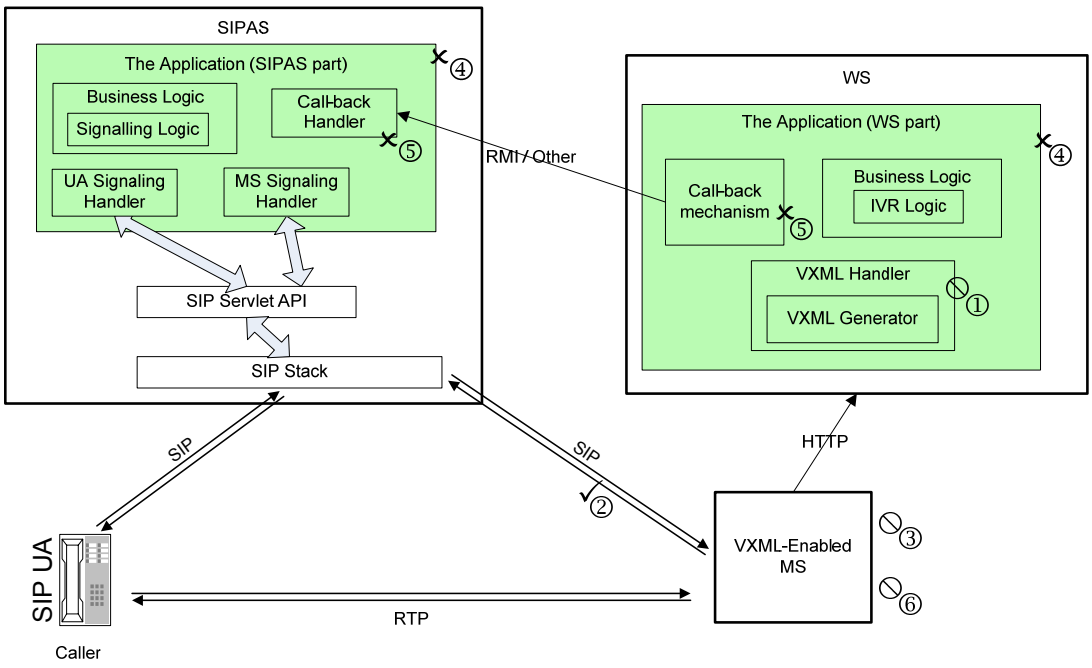

Figure 13 VXML Alternative - Deployment View – Evaluation Summary

Next, each of the criteria is analyzed in detail, and given a proper score:

Criterion-1) Ease of Development

Simplicity of IVR Command Request Generation and Response Parsing

Evaluation mark: 2 points (Neutral).

The business logic development is split in this approach. The SIPAS is still responsible for orchestrating the SIP signalling and trigering the IVR, but it is the WS that has the logic for driving the IVR. This is possible because VXML is a complex language with decision blocks and variables. The MS can also issue a subsequent HTTP request in order to generate an updated VXML page and trigger a server side business logic during this process.

Although there is minimum SIP traffic needed between the SIPAS and the MS, and although all the IVR commands interpreted by the MS are actually generated in the WS, the coding of a VoiceXML application is not an easy task even for experienced developers:

a) it is not intuitive to combine server side (HTTP Servlet) commands with VXML tags;
b) there is limited syntax checking as it happens when using an API like Java (meaning, if the developer makes a typo, she will not detect it until she tries to run the code, which is time consuming);
c) if the user tries to run the code and there is a problem, then it becomes difficult to trace this problem. Often, messages on the MS logs are difficult to trace and this makes it hard to pinpoint the exact problem back to the WS side, where the HTTP Servlet was coded;
d) samples found on the Internet that could potentially help solve the problem do not always work on the specific VXML flavour a developer is targeting as per a specific MS requirement.

This evaluation can be visualised in Figure 13 under index number ①.

Signalling Simplicity

Evaluation mark: 3 points (High).

Due to the fact that the application logic is split between SIPAS (handling the SIP) and the WS (handling the IVR), the application hosted in the SIPAS is not actually in control over what specific IVR commands are run, the Web pages hosted in the WS are the ones that generate the VXML and are the ones in control of the IVR. This leads to a very simple SIP signalling between the SIPAS and the MS (which can be seen from

Figure 11), meaning less SIP corner cases and only one call-back needed at the end of the IVR to report the SIPAS of any collected information.

This evaluation can be visualised in Figure 13 under index number ②.

Ease of SIP Unit Testing

Evaluation mark: 2 points (Neutral).

Two open source unit testing tools, SipUnit [16] and HttpUnit [12], are often leveraged here in order to provide the required SIP simulation for exercising the SIPAS functionalities, and the HTTP simulation for exercising the WS functionalities and therefore simulate end-user IVR interactions.

Such testing is often not a trivial task as it requires the Web application to be coded in specific ways, and both SipUnit and HttpUnit to be coded in a single test case for a meaningful end-to-end call simulation.

This evaluation can be visualised in Figure 13 under index number ③.

Central Development

Evaluation mark: 1 point (Low).

The application hosted in the SIPAS is responsible only for the SIP signalling and for instructing the media server where to get the initial VoiceXML page from. The actual logic of the IVR handling (what to play and collect and how to handle the IVR results) is coded in the WS. This makes application integration harder as developers are split into two expertise domains, it is harder to trace issues due to this split as well, and there is also a need for developers to have specialized knowledge in both SIP containers (using SipServlets) and Web containers (using HttpServlet or others).

This evaluation can be visualised in Figure 13 under index number ④. Note that the index ④ is shown twice, in the WS and in the SIPAS parts of the developed custom application.

Call-back Mechanism

Evaluation mark: 1 point (Low).

There is no call-back mechanism defined in VXML. Although the MS has an open SIP dialog with the SIPAS, runs the VXML code, and collects the end-user DTMF input, there is no formal way of passing that input back to SIPAS. In other words, the SIP

dialog is maintained between the MS and SIPAS, but the IVR logic is in the WS and the WS does not have a way of instructing the MS to send in-dialog SIP messages back to SIPAS.

This forces developers to be responsible for coding their own mechanisms to accomplish such task. In this protoype, RMI was used as Java is a popular implementation language for both Web and SIP application servers.

This evaluation can be visualised in Figure 13 under index number ⑤. Note that this index also appears twice, once in the WS to generate the call-back, and once in the SIPAS to receive the call-back and trigger the appropriate business logic.

Criterion-2) Portability

Evaluation mark: 2 points (Neutral).

Although there are more media servers that are VXML-enabled than M*ML-enabled ones, and although VXML is a standard language defined by W3C [26], different MS might support different versions of the standard. Also, XML-based languages inherit the "X" capability from XML that makes them "eXtensible". This can be seen as good thing for data structures, but its value for a language (such as VoiceXML) is questionable. By being extensible, the different media servers are allowed to create custom tags, extend the existing ones, or even partially implement a given VXML specification. This leads to a proliferation of different (and often proprietary) flavours of VXML, which means that a VXML script that is generated by a WS and run on given MS cannot be guaranteed to run on a different MS. In a previous paper [1], we already observed that the use of XML-based scripting languages "at times leads to the proliferation of additions that break code portability and interoperability (recall what happened to HTML)".

This evaluation can be visualised in Figure 13 under number index number ⑥.

Criterion-3) Signalling Load

Evaluation mark: 7.2%.

From Table 7 we can see that the total number of bytes needed to fulfil the use case using VXML was 13,623 bytes, this value represents the IVRSignallingLoad.

Applying the formula detailed in section 3.1, this alternative gives us: SignallingLoad = 100% * (13,623 bytes / (174,560 bytes + 13,623 bytes)) = 7.2%.

4.3. Chapter Summary

This chapter described and analysed two state-of-the-art development approaches for IVR applications that are to run on a SIPAS and that have the media provided by a SIP-enabled MS.

The first approach used MSML or MSCML and was generically named M*ML (section 4.1). The SIPAS controls the MS via an embedded protocol (the M*ML) within the SIP protocol, via the use of SIP INFO messages.

The second approach analysed used VXML (section 4.2). The application development is split and the call control part is driven by the application deployed in the SIPAS, and the IVR part is driven by the WS that generates VXML pages to be interpreted by the MS.

Both state-of-the-art approaches were prototyped and run using the Auto-Attendant use case (defined in section 3.2). Their output results and the experience acquired during the development were used to assess the approaches against the evaluation criteria defined in section 3.1.

The next chapter will detail the novel IVRObject approach, this thesis proposed alternative to simplify IVR application development.

Chapter 5. IVRObject – Concept and Definition

This chapter proposes *IVRObject*, a novel approach for providing IVR capabilities to an end-user SIPUA that has its SIP signalling controlled by an application running in a SIP application server and the IVR media streamed by a SIP-enabled media server. A general overview of the selected strategy is first presented in section 5.1 followed by implementation details in sections 5.2 and 5.3. An automated testing approach for IVRObject-based applications is then explained in section 5.4.

5.1. The IVRObject Strategy

In Chapter 4, we analyzed two popular approaches for providing IVR capabilities, one using M*ML-enabled media servers and the other using VXML-enabled media servers. We also evaluated each approach against the criteria defined in section 3.1.

We can observe so far that VXML shows several benefits over M*ML for the following aspects:

- Requires a simpler SIP signalling;
- VXML-enabled MS are more popular, which facilitates portability;
- As per our evaluation, it is easier to deal with the generation of VXML than the generation and parsing of M*ML payloads and the SIP messages used to carry them.

We can also observe the following advantages of M*ML over VXML:

- Has a built-in call-back mechanism;
- Requires a centralized application development.

The strategy for IVRObject is to take the best features present in each of the two state-of-the-art approaches and target a solution where:

- A simple SIP signalling is required;

- Portability is facilitated through a dependency on a VXML-based MS;
- A built-in call-back mechanism is provided to the developer;
- A centralized application development is required.

Moreover, IVRObject also aims:

- To increase the portability among different VXML-based MS, by abstracting the different versions and flavours from the application development.
- To define a testing strategy in order to allow the application developer to rely on SipUnit [16] only, and not on HttpUnit [12], to make the different call simulations.
- The abstraction of the call-back implementation details from the application developed. Therefore, if this call-back mechanism or its protocol is changed, this will not affect the IVR application developed.

5.1.1 Deployment and Implementation Strategy Overview

Figure 14 shows the deployment view for the network elements and for the modules that make part of a custom application (the Application) in a typcial solution involving the usage of the IVRObject.

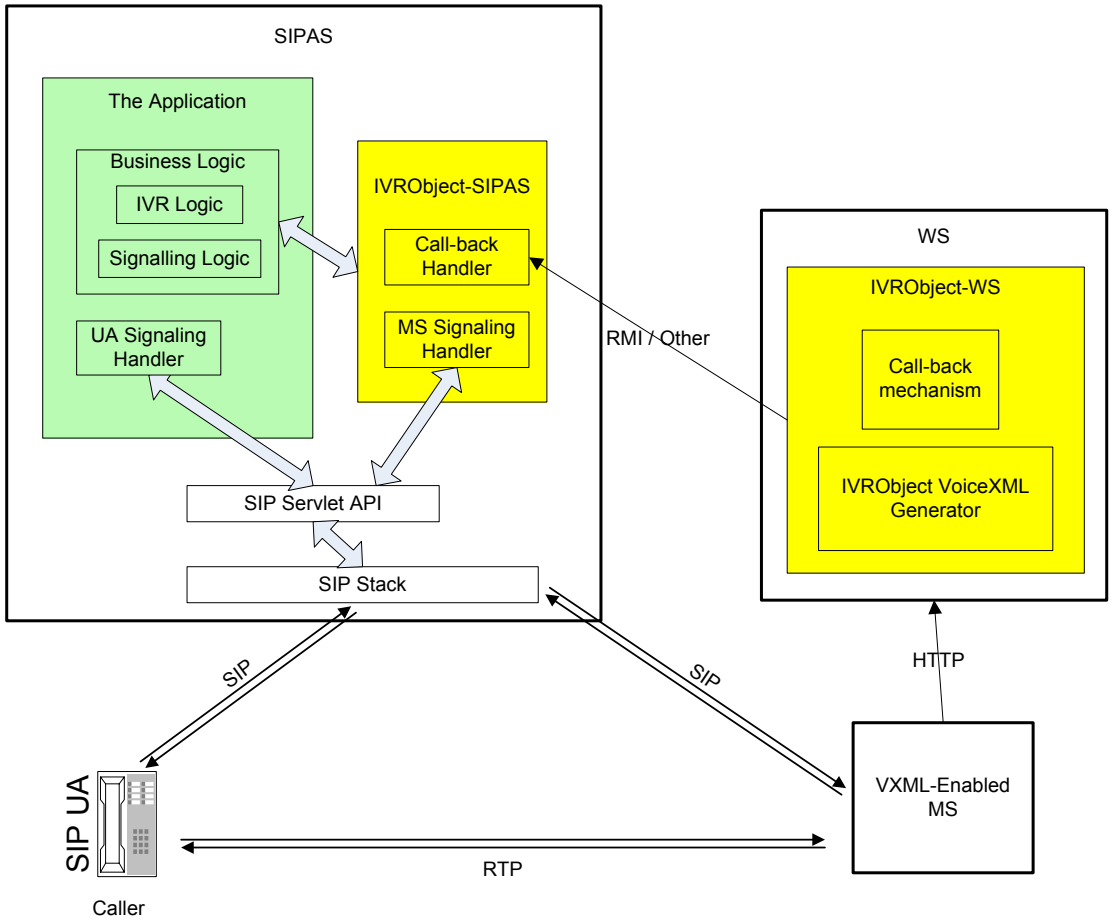

Figure 14 IVRObject - Deployment Strategy Overview

Note in Figure 14 that, as described in section 2.5 and as shown in Figure 6 (for the M*ML approach) and Figure 10 (for the VXML approach), the IVRObject also requires the same three components in order to have the IVR capabilities provided: the SIPAS, the WS and the SIP-enabled MS.

The following key characteristics for the IVRObject implementation strategy can be observed from the Figure 14:

- The green boxes represent the custom application that requires specific development; note here that the IVRObject requires a centralized development as all the green boxes are on the SIPAS side.
- The yellow boxes are part of the IVRObject framework itself. Think of them as libraries or pre-installed components that are available for the developer to use. They do not need to be modified and they were previously tested.

- The yellow boxes on the WS side communicate directly to the yellow boxes on the SIPAS side, suggesting here that there is a built-in call-back mechanism that the developer does not need to be aware of.
- The IVRObject approach still relies on a VXML-based MS, which improves the likelihood for portability to different MS vendors.
- Also, by relying on a VXML-based MS we will be able to reduce the number of SIP messages used to interact with the MS, hence increasing the overall application robustness.
- The VXML code to be interpreted by the MS is generated by the IVRObject VoiceXML Generator. This VXML generation mechanism is application independent. This mechanism will also allow us to cope with the different VXML flavours via the use of plug-ins, which the application developer will not need to be aware of, hence simplifying the application development.

5.2. IVRObject Components and Implementation Details

The following diagram further details the overview given for the IVRObject approach in Figure 14 by presenting the underlying components needed to achieve the proposed strategy and the key interfaces.

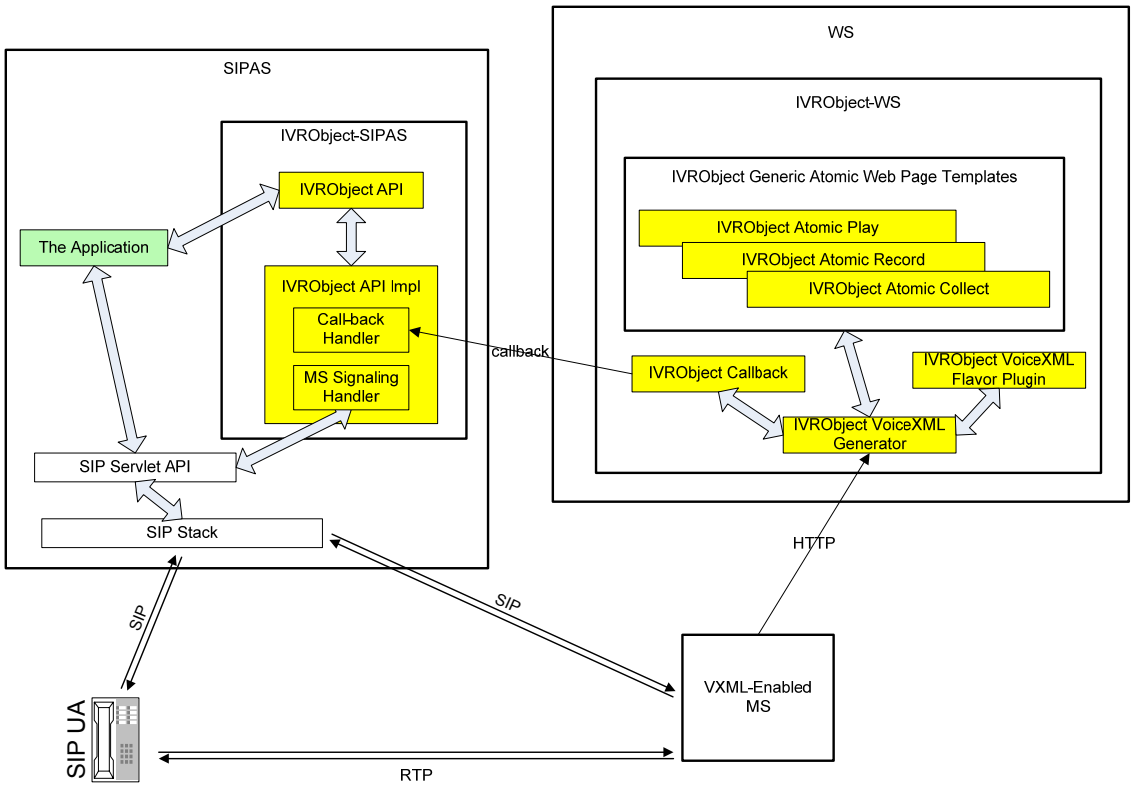

Figure 15 IVRObject - Deployment Strategy Details

IVRObject is a reusable piece of code that is application independent and that targets the abstraction of the media handling from the IVR applications. The IVRObject is divided into two parts: the IVRObject-SIPAS that resides in the SIP application server, and the IVRObject-WS that is deployed in the Web server.

The IVRObject-SIPAS can be seen as a generic library that is available to the SIPAS application developer, and the IVRObject-WS can be seen as a Web Archive (WAR) that is also generic but gets to be deployed on a Web application server as a standalone Web service.

It is important to note that the SIPAS developer will not have to make direct use of IVRObject-WS, as it is deployed in the WS simply to assist and interact with the IV-RObject-SIPAS.

The IVRObject-SIPAS is composed of two components: the IVRObject API and the IVRObject API Impl. The IVRObject-WS is composed of four components: the IV-

RObject VoiceXML Generator, the IVRObject Call-back, the IVRObject VoiceXML Flavour Plug-in, and the IVRObject Generic Atomic Web Pages Templates. Table 9 gives an overview of these components.

Once IVRObject-SIPAS and the IVRObject-WS components are in place and properly configured to interact with each other, the application running in SIPAS can make use of the IVRObject via the IVRObject API (detailed in Appendix D), whose usage is detailed in the next sections. In Figure 15, the application is the customer-specific code that makes use of the IVRObject API in order to orchestrate the IVR interaction.

Table 9 IVRObject Components Details.

Component	Details
IVRObject API	Provides the Application with an easy-to-use API for requesting IVR functionality (see Appendix D).
IVRObject API Impl	The implementation of the IVRObject API that is able to handle call-backs from the IVRObject-WS and to establish the connection to the MS on the developer's behalf.
IVRObject VoiceXML Generator	The orchestrator of the VXML page to be generated for the MS. It can request multiple atomic operations to be part of the generated VXML, it includes the call-back mechanism, and it adapts to a specific VXML flavour for compatibility to different media servers (more details on this generator in the example to follow).
IVRObject Call-back	Injects hooks into the generated VXML so the MS issues a subsequent HTTP request back to this component, which will provide a way of notifying the IVRObject API Implementation of the atomic operation's result.
IVRObject VoiceXML Flavour Plug-in	Massages the generated VXML to comply to a vendor-specific VXML flavour.
IVRObject Generic Atomic Web Page Templates	Provides template VXML tags for the IVR atomic operations: play, collect and record.

5.2.1 Use Case Call Flow

This section details the SIP signalling required in order to implement the auto-attendant use case defined in section 3.2 using the IVRObject. To better explain the inner working mechanisms and component interactions around the IVRObject, its signalling was divided into 2 parts (A and B).

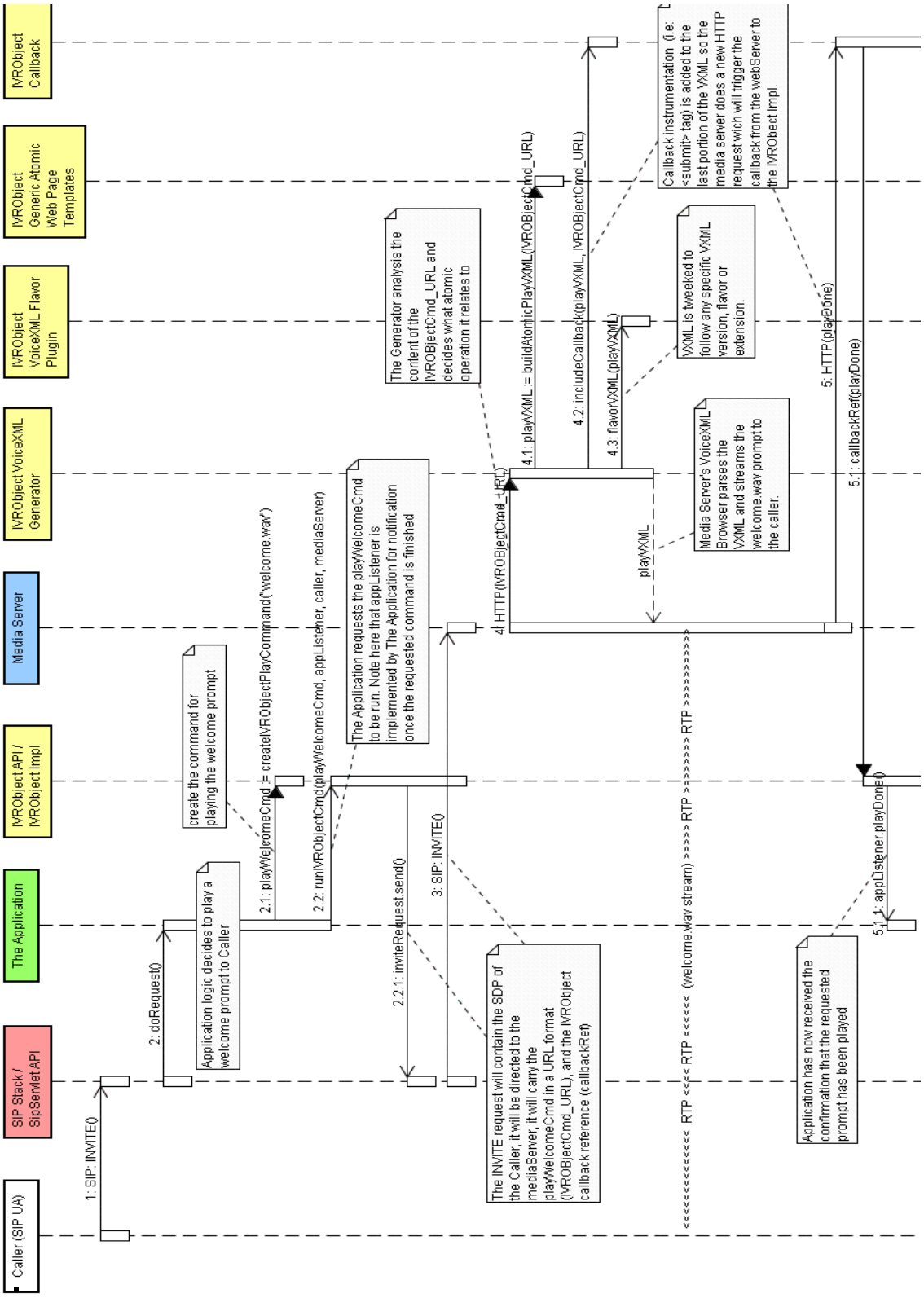

Figure 16 Auto-Attendant Signalling Part-A

Signalling Part-A

Figure 16 details the signalling for the first part for the IVRObject implementation for the auto-attendant use case, where the application (running in SIPAS) requests the playing of a welcome prompt (in messages “2.1” and “2.2”) and is notified of its completion (message 5.1.1).

Note that although lots happen behind the scene, the interface that the IVRObject API provides to the Application is very simple. From the application perspective, the IVRObject API provides a way for requesting some IVR service at a high level and a way for letting the application know once this service has terminated.

Also note that some SIP messages (RING, OK for INVITE, ACK, and BYE) are on purpose omitted here for clarity.

Signalling Part-B

Figure 17 is the continuation for the signalling detailed in Figure 16. In this part, the application requests the playing of a prompt and a collection of DTMF digits. Both of these atomic operations are bundled in a “group” to minimize traffic, as shown in the following figure.

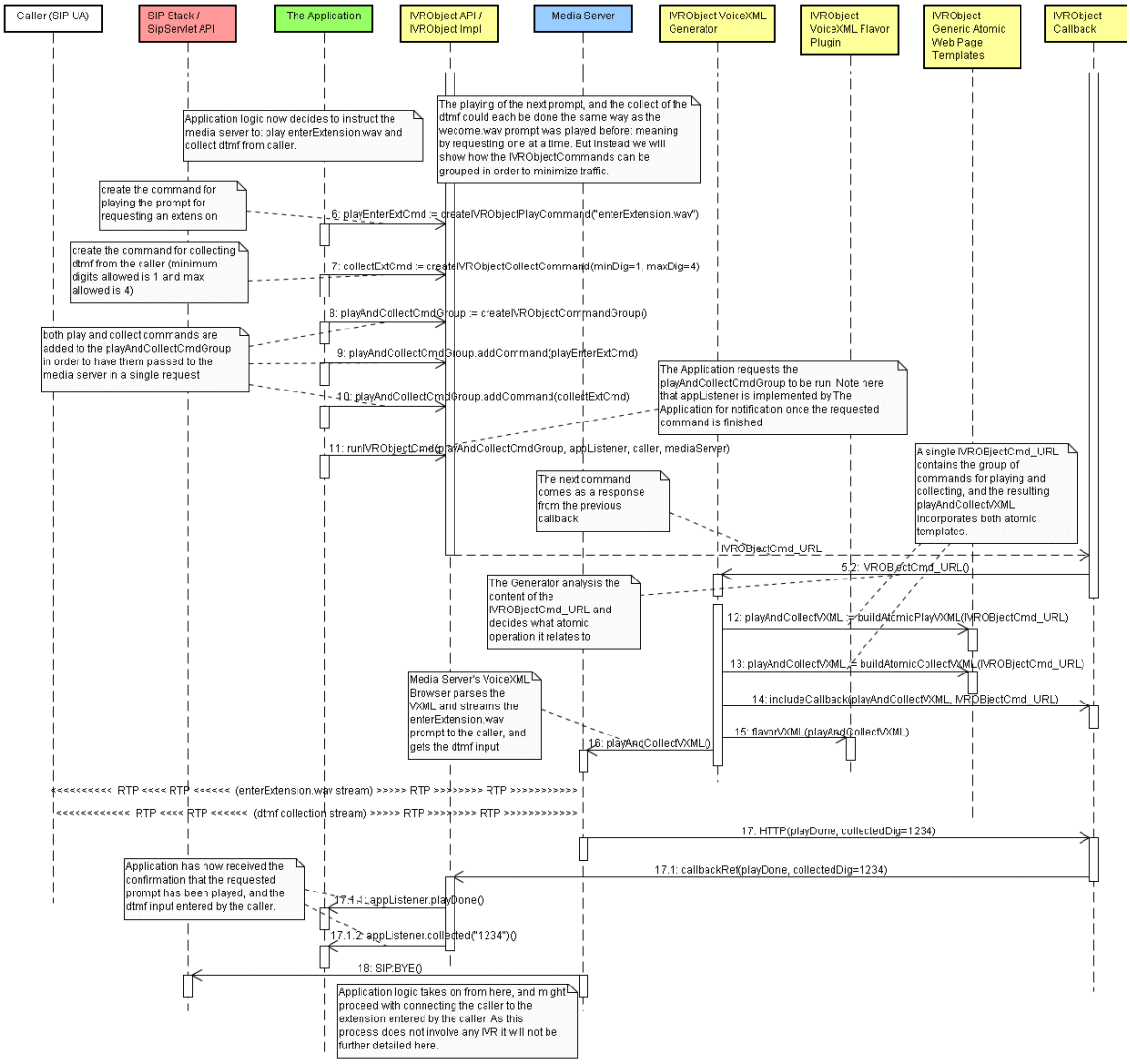

Figure 17 Auto-Attendant Signalling Part-B

One important aspect to note is the minimum common denominator (MCD concept, see section 2.3) is also being applied here for the Caller; that is, the caller is a simple SIPUA, and it is totally unaware of the signalling being used to reach the MS.

5.2.2 Use Case Call Flow - Step-by-Step Description

Table 10 makes references to the steps of the sequence diagrams in section 5.2.1 (for both Signalling Part-A and Signalling Part-B), giving further details. The description of the IVRObject API methods and classes is available in Appendix D.

Table 10 Auto-Attendant Signalling Details.

Step	Comments
1	Caller dials into SIPAS, the SIP INVITE sent by the caller UA is handled by the SIPAS SIP Stack and SipServlet API.
2	The application is invoked via the doRequest() method (method defined in the SipServlet API – used to notify application of incoming SIP requests)
Note	The application now (as per its business logic) decides to play a welcome prompt to the caller
2.1	The application invokes the "createIVRObjectPlayCommand" method defined in the IVRObject API in order to define what prompt to play ("welcome.wav" in this case). The command reference is named arbitrarily by the application as: playWelcomeCmd
2.2	The application invokes the "runIVRObjectCmd" method defined in the IVRObject API in order to request the previously created playWelcomeCmd to be run. Along with the playWelcomeCmd created in the previous step, there are also three parameters passed: - appListener (an instance of a listener that implements the IVRObjectListener, that will allow the application to be notified of the results of the requested command at a later point); - caller (that contains the caller details, including its SDP media options to be sent to the MS), - media server (the MS address to be used)
2.2.1	The IVRObject Impl considers all the parameters specified in the previous item and constructs a new INVITE request using the SIP Servlet API. This request will have the MS address as the reqURI target, the request payload will carry the caller SDP, and the request will have a "voicexml" parameter that is added to the reqURI with the content generically named here as "IVRObjectCmd_URL" (but note that in reality it contains something in the format: http://someserver.com?play=welcome.wav&callbackRef=<details>). **Note1**: the "callbackRef" is a reference to the IVRObjectImpl and is a hook that is passed all the way to the Web server via the media server, so the Web server can call the IVRObject Impl back to report on the atomic operations. **Note2**: passing a "voicexml" param in the reqURI is a standard way for letting a VXML-enabled media server know the Web server URL that will generate the VXML script that the media server will interpret and run – note that no new functionality is needed on the media server side for this.
3	The SIPAS via its SIP Servlet API and SIP Stack sends the SIP INVITE message to the MS.

Step	Comments
4	The MS will follow its expected behaviour and will first extract the "voicexml" parameter from the reqURI (again, generically called here "IVRObjectCmd_URL"), and then issue an HTTP request to this URL in order to get a VXML back so it can interpret and run.
Note	The IVRObject VoiceXML Generator will then interpret the content of the IVRObjectCmd_URL and invoke the different helper components that will build the playVXML (the VoiceXML script) to be returned back to the MS.
4.1	IVRObject VoiceXML Generator will first request the IVRObject Atomic Play (part of the IVRObject Generic Atomic Web Page Templates) in order to start adding the required VXML tags to form the playVXML.
4.2	IVRObject VoiceXML Generator will then request the IVRObject Call-back to include in the playVXML the required VXML tag to force the MS to issue a subsequent HTTP request back reporting the result of the VXML run (the Call-back instrumentation, i.e.: <submit> tag, is added to the last portion of the VXML)
4.3	IVRObject VoiceXML Generator will next request the IVRObject VoiceXML Flavour Plug-in to massage the current content of the playVXML that is being built in order to comply with the VXML flavour/version/extension that is supported by the VXML Browser of the current installed media server.
Note	Once the playVXML is passed to the MS. It will interpret and run it. At this point via an RTP session the content of the welcome.wav prompt is streamed to the caller.
5	Once the welcome.wav is played, the MS VoiceXML Browser will run the last tag in the playVXML script that instructs the MS to issue a new HTTP request (which carries parameters such as specifying that the prompt was played, and the instrumented callbackRef)
5.1	IVRObject Call-back will then use the callbackRef to reach the IVRObject Impl, and report on the completion of the requested command
5.1.1	The application listener implementation of the IVRObjectListener will have its "playDone()" method called.
Note	At this point, the application knows the welcome.wav has been played, and, following its business logic, instructs the media server to: play enterExtension.wav and collect DTMF digits from the caller.
Note	The playing of the next prompt, and the collection of the DTMF digits could each be done the same way as the welcome.wav prompt was played before: meaning by requesting one at a time. But instead we will show how the IVRObjectCommands can be grouped in order to minimize traffic.

Step	Comments
6	The application invokes the “createIVRObjectPlayCommand” method defined in the IVRObject API in order to define what prompt to play ("enterExtension.wav" in this case). The command reference is named arbitrarily by the application as: playEnter-ExtCmd.
7	The application invokes the “createIVRObjectCollectCommand” method defined in the IVRObject API in order to request the collection of DTMF input from the caller. The application also requests that the minimum number of digits allowed from the caller be 1 and that the maximum allowed be 4). The command reference is named arbitrarily by the application as: collectExtCmd.
8	The application invokes the “createIVRObjectCommandGroup” method defined in the IVRObject API in order to create a holder for a sequence of commands. Using a group has the advantage to provide a developer the ability to chain several related atomic operations into one logical unit, and to minimize the traffic required as all the commands will be passed to the media server at once instead of one at a time. This created group reference is named arbitrarily by the application as: playAndCollectCmdGroup
9	The playEnterExtCmd command (created in step-6) is added to the playAndCollectCmdGroup (created in step-8). This will add the first atomic operation to the group.
10	The collectExtCmd command (created in step-7) is also added to the playAndCollectCmdGroup (created in step-8). This will add the second and last atomic operation to this command group.
11	The application invokes the “runIVRObjectCmd” method defined in the IVRObject API in order to request the created group of commands playAndCollectCmdGroup to be run. Along with the playAndCollectCmdGroup, one additional parameter is passed: - appListener (an instance of a listener that implements the IVRObjectListener, that will allow the application to be notified back of the results for the requested command at a later point).
Note	The IVRObject Impl considers all the parameters specified in the previous item and this time, as it is a subsequent command request, instead of constructing a new INVITE, it uses the pending call-back synchronous request (issued originally in step-5.1) to return this new set of commands back to the Web server.
5.2	The next set of commands is passed to the IVRObject VoiceXML Generator, which will then interpret the content of the IVRObjectCmd_URL and invoke the different helper components that will build the playAndCollectVXML (the VXML script) to be returned back to the media server.

Step	Comments
12	IVRObject VoiceXML Generator will first request the IVRObject Atomic Play (part of the IVRObject Generic Atomic Web Page Templates) in order to start adding the required tags to form the playAndCollectVXML.
13	Next the IVRObject VoiceXML Generator will request the IVRObject Atomic Collect (part of the IVRObject Generic Atomic Web Page Templates) in order to add the required tags to form the playAndCollectVXML.
14	IVRObject VoiceXML Generator will then request the IVRObject Call-back to include in the playAndCollectVXML the required VXML tag to force the MS to issue a subsequent HTTP request back reporting the result of the VXML run (the Call-back instrumentation, i.e.: <submit> tag, is added to the last portion of the VXML)
15	IVRObject VoiceXML Generator will next request the IVRObject VoiceXML Flavour Plug-in to massage the current content of the playAndCollectVXML that is being built in order to comply with the VXML flavour/version/extension that is supported by the VoiceXML Browser in the media server.
16	Once the playAndCollectVXML is passed to the MS. It will interpret and run it. At this point via the same RTP session established before the content of the enterExtension.wav prompt is streamed to the caller, and the DTMF digits entered by the caller via the phone keypad are captured by the MS.
17	The MS VoiceXML Browser will run the last tag in the playVXML script that instructs the media server to issue a new HTTP request (which carries parameters such as specifying that the prompt was played, the collected digits and the instrumented callbackRef)
17.1	IVRObject Call-back will then use the callbackRef to reach the IVRObject Impl, and report on the completion of the requested command.
17.1.1	The application listener implementation of the IVRObjectListener will have its “playDone()” method called.
17.1.2	The application listener implementation of the IVRObjectListener will have its “collected()” method called with the digits entered by the caller ("1234" in this example).
18	As no other command was passed to the Web application, this is the end of the IVR interaction, and the MS sends a BYE.
Note	Application logic takes on from here, and proceeds with connecting the caller to the extension collected. As this process does not involve any IVR it will not be further detailed here.

5.3. Observations

As shown in the example call flow (section 5.2.1) and its detailed signalling description (section 5.2.2):

- The application using the IVRObject will only need to be coded in the SIPAS component.
- There is then an API that allows the application to be developed in the SIPAS to interact with the IVRObject framework to provide the IVR functionality. This API is called the "IVR Object API" (see Appendix D for API details), and its implementation (the "IVRObject API Impl") is responsible for the interaction with the "IVRObject-WS" that resides in the WS.
- The application only needs to interact with the IVRObject API in order to request the atomic IVR operations (play, collect, record). Everything else is transparent to the application and the developer does not need to worry about including/coding: a) the communication between the IVRObject-WS and the IVRObejct-SIPAS components, b) the generation of the VXML handled via the usage of the IVRObject Generic Atomic Web Page Templates, and c) the built-in call-back mechanism.
- The overall concept for the IVRObject is based on our observation that any IVR application can be broken down into three atomic operations (play/collect/record). What makes an IVR application specific/unique is how these operations are combined. Hence, the framework needs to provide a way to support these atomic operations. The control of what atomic operations should be invoked is all done in the SIPAS component by the application that is in control of the business logic.
- The WS that hosts the files that generate the VXML is generic, e.g., the IVRObject-WS component is application independent and no application developer needs to see these files or make changes to them, or even know they exist. This is all part of the IVRObject framework available to the developer and accessible to the SIPAS via the IVRObject API.
- The IVRObject-WS makes use of specific pluggable VXML implementations for each VXML flavour that needs to be supported. The IVRObject

VoiceXML Flavour Plug-in abstracts the nuances that exist among the different versions and flavours of VXML during the generation of the atomic operations that will be part of a generated VXML document.

- Also note that although we are having different plug-ins attached to the framework, the application that is developed in the SIPAS remains the same (no line of code needs to be changed on the application side if we are changing the MS vendor) – as long as the MS is VXML and SIP-enabled.

5.4. IVRObject Automated Test Strategy

The testing of IVR applications is one of the concerns discussed in the criteria of Chapter 3. Figure 18 provides an overview of how the "IVRObject API Impl" can be replaced with an "IVRObject API *Test* Impl". This is done at deployment time, via configuration of the IVRObject, to abstract the Web server as a whole and to have its call-back managed by the test driver in order to simulate different user inputs, and how the different SIP end points around SIPAS (including the user agents and the media server) could be replaced with mock ones.

Figure 18 highlights, in yellow, the components that are part of the IVRObject framework from a testing perspective. Note the IVRObject API Test Impl is deployed in the SIPAS and receives call-backs from the IVRObject Test Call-back Driver. The latter is also part of the IVRObject framework and its purpose is to insulate the test driver from knowing how to communicate with the IVRObject running in the SIPAS component.

This strategy allows the application (running in SIPAS) to be tested programmatically (i.e., using automated testing APIs such as SipUnit [16]) for different functional scenarios, without any need to be changed because the IVRObject API itself remains unchanged.

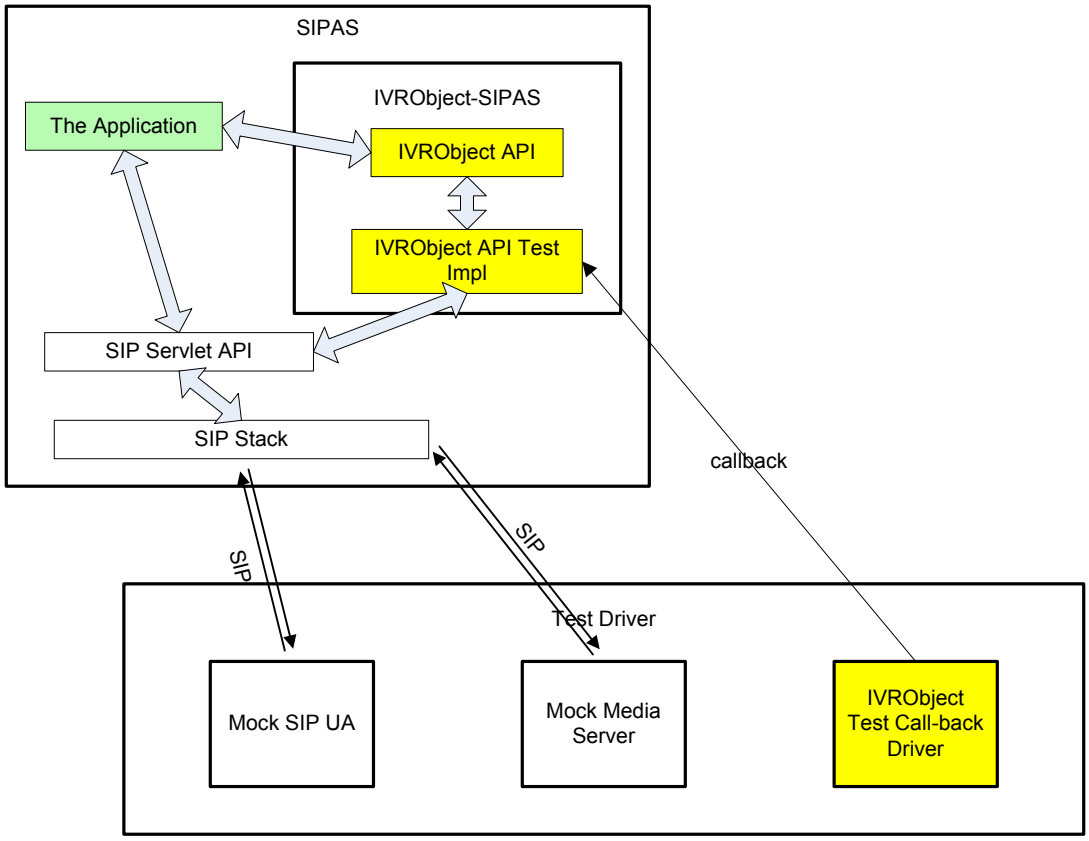

Figure 18 IVRObject Test Strategy

Figure 20 illustrates the usage of the IVRObject test strategy to implement the success path – the same scenario illustrated in section 5.2.1, but now simulating the end points.

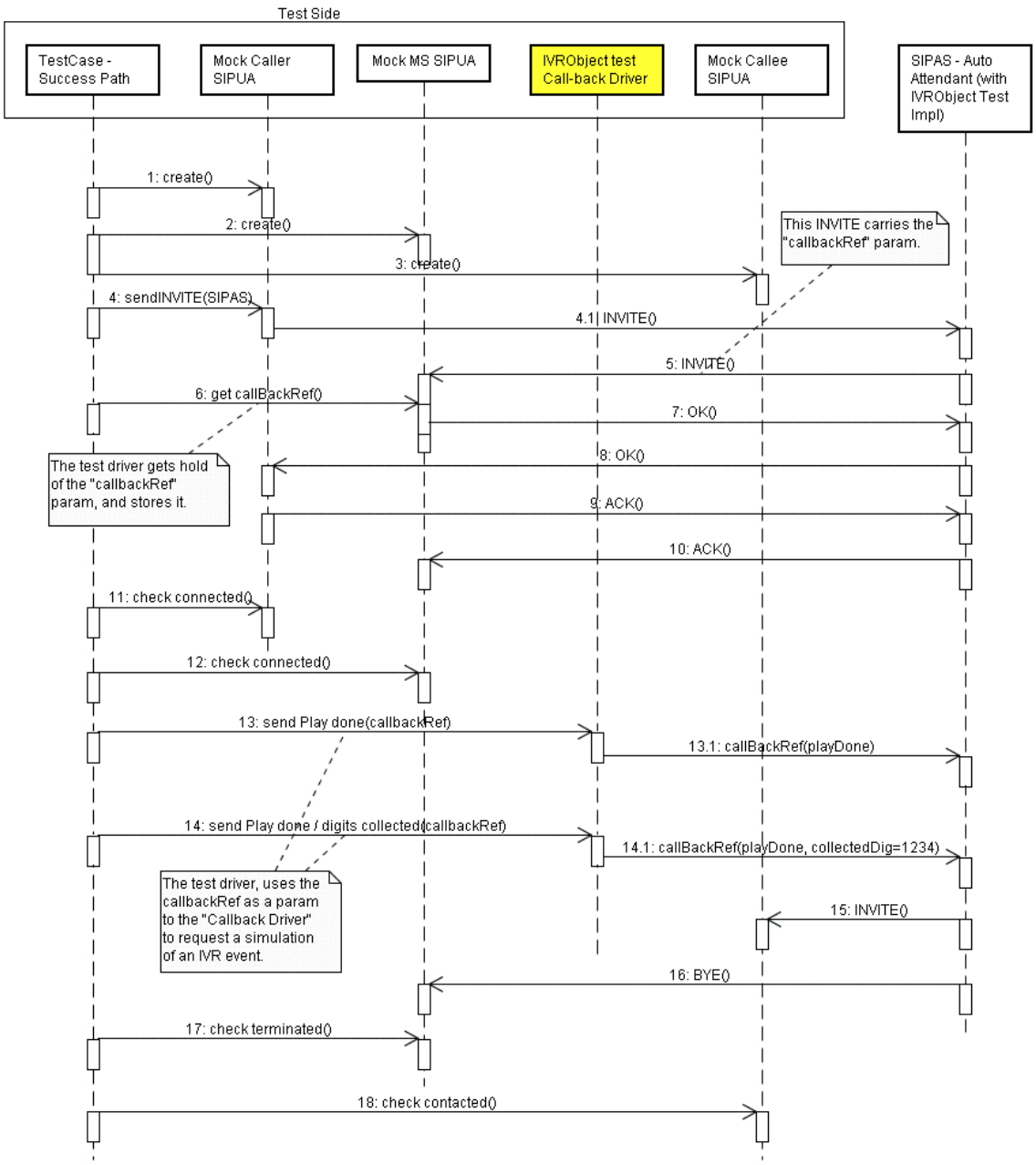

Figure 19 IVRObject Test Strategy for Success Path

Note in Figure 19 that message 5 corresponds to messages 2.2.1 and 3 in Figure 16. As commented earlier, the INVITE out to the MS carries a callbackRef. The only difference when running the test is that this callbackRef now refers to the IVRObject API *Test* Impl instead of the regular IVRObject API Impl, but this is transparent to the IVR application deployed in SIPAS.

Note also, in Figure 19, that the IVRObject Test Call-back Driver provides an API (detailed in Appendix E) to the test developer to report on the IVR events. For instance, message 13 instructs this driver to report the simulation that the prompt was played, and message 14 to report that the prompt was played and that DTMF digits were collected. All that the test developer needs to do is get hold of the callbackRef (as illustrated in message 6) via SipUnit and use that callbackRef when invoking the IVRObject Test Call-back Driver.

The next diagram (Figure 20) illustrates an alternative test scenario that simulates the situation where the MS is busy. A simple test to perform from a simulation perspective, but an extremely difficult scenario to test using a real MS, as it is not easy to overload a MS in order to have all its ports occupied so it replies with a busy signal:

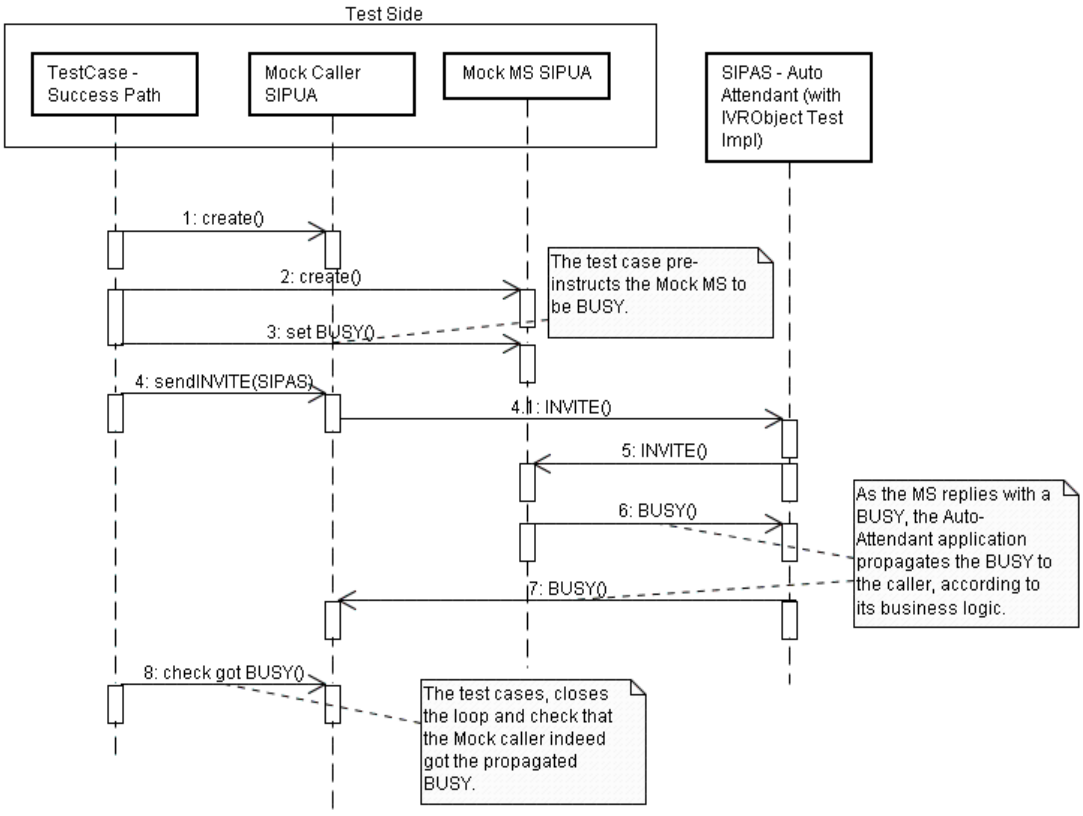

Figure 20 IVRObject Test Strategy For MS Busy Path

5.5. Chapter Summary

This chapter presented the main contribution of this thesis, IVRObject, as a mechanism for providing media control to SIP-based IVR applications.

This chapter started by providing a strategy for taking the best ideas from the current state-of-the-art approaches, and then provided specific details on how to achieve this integration goal. The chapter ends by proposing a test strategy part of the overall IVRObject approach in order to facilitate automated testing via call simulation without modifying the application.

The next chapter will analyze the run time data collected during the prototype developed for the IVRObject for the auto-attendant use case defined in section 3.2.

Chapter 6. IVRObject Prototype and Evaluation

We have already made use of the auto-attendant use case defined in section 3.2 for prototyping and collecting runtime data for both of the state-of-the-art approaches: the M*ML one (in section 4.1) and the VXML one (in section 4.2). These experiments made it possible for us to gain experience on each of the implementations and to understand their challenges and draw conclusions about them.

Once again we make use of the same auto-attendant use case, but now to have it prototyped using the IVRObject framework defined in Chapter 5. This prototype implements the signalling detailed in Figure 16 and Figure 17, given the deployment strategy defined in Figure 14.

We will collect the runtime data during the execution of the IVRObject prototype. This data, in conjunction with the analysis of the signalling output (Appendix C) and the lessons learned during the implementation, will enable the evaluation of the IVRObject as an alternative for orchestrating media capabilities of IVR applications that run in a SIP application server and have the media streamed to an end-user via a SIP-enabled media server. Using the same criteria and case study as for the other prototypes will also provide a common basis for comparison.

The sample code with a Java implementation for the auto-attendant can be found in Appendix F.

6.1. Runtime Data

The runtime data collected for the IVRObject-based prototype is sumarized next. For the full SIP signalling trace captured during this test, please refer to Appendix C.

SIP, HTTP and RMI Network Traffic Analysis

We collected the signalling messages that were detailed in Figure 16 and Figure 17, focusing both on the SIP and HTTP messages around the MS, and on the RMI messages

around the call-back mechanism between the WS and the SIPAS. Figure 21 highlights (with an ellipse) the area for which data is captured.

For the RMI messages specifically, we are looking at steps 5.1 in Figure 16 and 17.1 in Figure 17, which show the RMI traffic generated to report the collected DTMF digits by the MS back to SIPAS. The RMI call-back is issued by the WS via an HTTP Servlet.

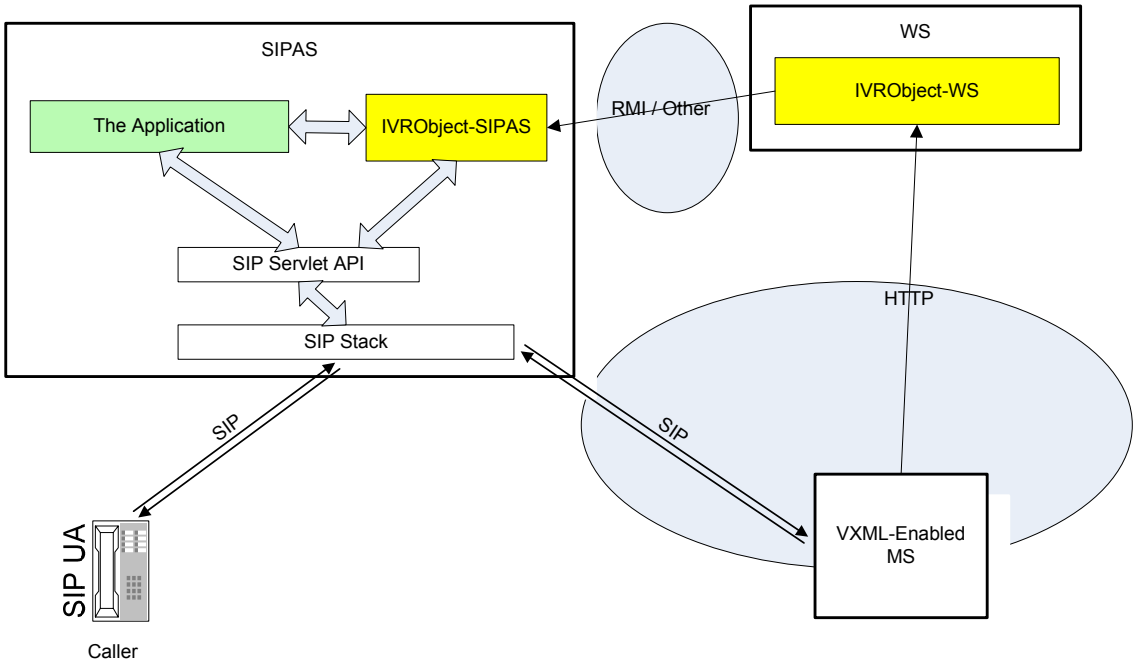

Figure 21 Capturing the Signalling for IVRObject

Figure 22 shows the network traffic data that was captured for the SIP, HTTP and RMI protocols while running the prototype for the IVRObject approach.

Legend:

- X-axis: time, in seconds
- Y-axis: traffic, in bytes
- Blue bar:
  SIP signalling
- Red bar:
  HTTP signalling
- Green bar:
  RMI signalling

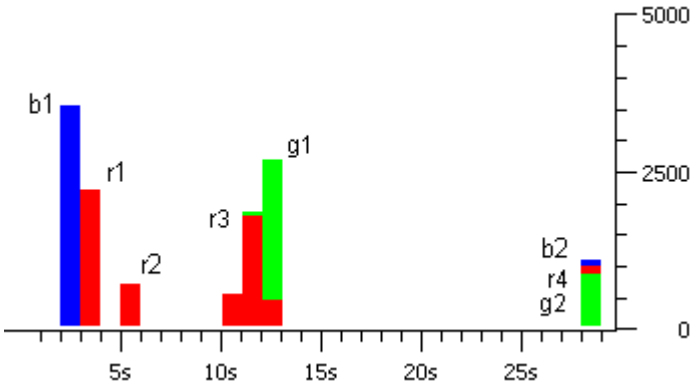

Figure 22 SIP, HTTP and RMI Network Traffic

We can see from Figure 22 that SIP and HTTP signalling required by the IVRObject implementation have the same magnitude. SIP had 7 messages while HTTP had 10, and SIP had an average of 722.8 bytes/message while HTTP had 844.5. This gives us an indication that the use of a VXML-enabled MS by the IVRObject is within the same magnitude as the ones observed for the approach that uses VXML (see Table 7).

In order to facilitate the correlation of the messages exchanged between the SIPAS and the MS for SIP messages, between the MS and WS for HTTP messages, and between WS and SIPAS for RMI messages detailed in Figure 16 and Figure 17, and the network analysis diagram in Figure 22, the following table is provided.

Table 11 SIP and HTTP Traffic for VoiceXML IVRObject using RMI Call-back

Point in Time	Details
b1	This is the SIP call set up from SIPAS to the MS. This corresponds to step 3 in Figure 16.
b2	Termination of MS. This corresponds to step 18 in Figure 17
r1	MS loading of the VXML from the WS that will greet the caller with the welcome.wav. This corresponds to steps 4 through 4.3 in Figure 16.
r2	MS invokes the WS Servlet to issue the call-back to SIPAS reporting the welcome.wav was played, and requesting the next set of instructions. This corresponds to the steps 5.1 in Figure 16, and its synchronous return step (dotted line after step 11) in Figure 17 that instructs the IVRObject to play the enterExtension.wav file and collect DTMF digits from the caller.
r3	MS invokes the WS Servlet to issue the call-back to SIPAS reporting on the data collected from the caller. This corresponds to the step 17 in Figure 17.
g1	WS call-back to SIPAS via RMI to report the welcome.wav was played and request the new set of commands. This corresponds to step 5.1 in Figure 16 and its synchronous return step (dotted line after step 11) in Figure 17, which instruct the IVRObject to play the enterExtension.wav file and collect DTMF digits from the caller.
g2	WS call-back to SIPAS via RMI to report that the enterExtension.wav was played and the DTMF digits were collected. This corresponds to step 17.1 in Figure 17.

HTTP, SIP and RMI Network Traffic Analysis Summary

Table 12 summarizes and compares the bandwidth usage for SIP, HTTP and RMI messages while running the IVRObject prototype. We can see that all protocols are in the same order of magnitude.

Table 12 SIP, HTTP and RMI Bandwidth Usage

	Total Bytes	Number of Messages	Avg Bytes/Message
SIP (i)	5,060 (26.7%)	7 (36.8%)	722.8
HTTP (ii)	8,445 (44.5%)	10 (52.6%)	844.5
RMI (iii)	5,478 (28.8%)	2 (10.5%)	2,739.0
Total	18,983	19	999.1

(i) Between SIPAS and MS

(ii) Between MS and WS

(iii) Between the WS and SIPAS

6.2. Evaluation

Table 13 summarizes the evaluation of our IVRObject approach for the 3 criteria defined in section 3.1:

Table 13 Evaluation Summary for IVRObject

Criteria	Score
Criterion-1) Ease of Development	2.8 (avg)
Criterion-2) Portability	3
Criterion-3) Signalling Load	9.8%

Figure 23 shows, once again, the deployment view that was introduced in Figure 14 , but now highlighting the positive, neutral and negative aspects around the different elements. The checkmark (✓) indicates a high score, the crossed-circle (⦸) suggests a neutral score, and the × means a low score for a given aspect being analysed. The numbering indexes will be referenced along the remaining part of this section where a detailed evaluation is provided.

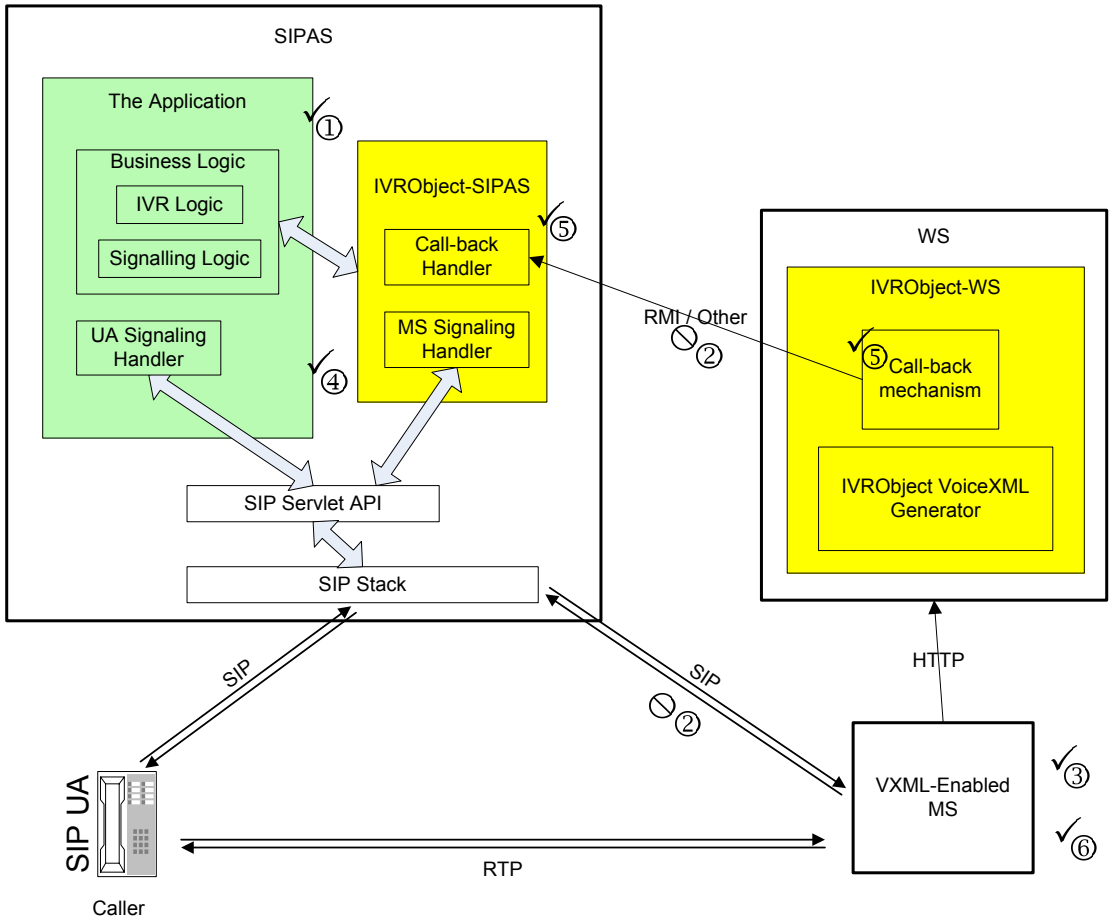

Figure 23 IVRObject - Deployment View – Evaluation Summary

Next, each of the criteria is analyzed in detail and given a proper score:

Criterion-1) Ease of Development

Simplicity of IVR Command Request Generation and Response Parsing

Evaluation mark: 3 points (High).

IVRObject does not rely on the M*ML usage, and no parsing/generation is needed from an application developer. Also, although IVRObject relies on VXML, it will use it generatively only, and the developer of the application will not even need to know that VXML is being used, i.e., all the complexity is hidden behind the IVRObject API.

This evaluation can be visualised in Figure 23 under index number ①; this is highlighting that the application has the coding facilitated by the presence of the IVRObject API.

Signalling Simplicity

Evaluation mark: 2 points (Normal).

IVRObject does not rely on SIP INFO messages for providing IVR capabilities, as the VXML code is generated in the WS and run by the MS. IVRObject also provides a mechanism for grouping the related IVR atomic operations to minimize the signalling traffic, as illustrated in section 5.2.1. On the other hand, the IVRObject has to rely on partial updates (call-backs) to the SIPAS in order to report the last results of the IVR and request subsequent commands, which adds some overhead to the signalling.

This evaluation can be visualised in Figure 23 under index number ②. This index is shown twice to highlight where the SIP and the call-back signalling take place.

Ease of SIP Unit Testing

Evaluation mark: 3 points (High).

Section 5.4 provides an overview of how the "IVRObject API Impl" could be replaced with a "IVRObject API Test Impl" to abstract the WS and have its call-back managed by the test driver in order to simulate different user inputs, and how the different SIP endpoints around SIPAS could be replaced with mock ones. This strategy allows the application to be tested programmatically (automated testing) for different functional scenarios, with no need for the application to be changed because the IVRObject API remains unchanged.

This evaluation can be visualised in Figure 23 under index number ③ .

Central Development

Evaluation mark: 3 points (High).

Similar to the the M*ML approach of section 4.1, the IVRObject approach also has a central development as the application resides in SIPAS only. This requires fewer integration points for the overall application, and as a consequence less expertise is needed from the developers.

This evaluation can be visualised in Figure 23 under index number ④.

Call-back Mechanism

Evaluation mark: 3 points (High).

IVRObject provides a built-in call-back mechanism that is transparent to the developer. It also decouples the application from the call-back mechanism used, which enables the use of a different mechanism (e.g., instead of RMI in the future) without affecting the application.

This evaluation can be visualised in Figure 23 under index number ⑤. Note that this index also appears twice, once in the IVRObject-WS to generate the call-back, and once in the IVRObject-SIPAS to receive the call-back and trigger the appropriate business logic.

Criterion-2) Portability

Evaluation mark: 3 points (High).

IVRObject relies on VXML under the hood, and there are more MS that support VoiceXML than M*ML. In addition, as IVRObject provides a mechanism for transparently dealing with the different VXML flavours, via plug-ins, it also provides an easier way to port the application across different VoiceXML MS vendors.

This approach also has the advantage of not relying on graphical user interfaces that generate VXML code. These tools often generate proprietary VXML flavours that make the resulting code not easily portable.

This evaluation can be visualised in Figure 23 under index number ⑥.

Criterion-3) Signalling Load

Evaluation mark: 9.8%.

From Table 12, we can see that the total number of bytes needed to fulfil the use case using IVRObject was 18,983 bytes, this value represents the IVRSignallingLoad.. Applying the formula detailed in section 3.1, this alternative gives us: SignallingLoad = 100% * (18,983 bytes / (174,560 bytes + 18,983 bytes)) = 9.8%.

6.3. Chapter Summary

In this chapter we analysed the IVRObject approach via a prototype that implements the concepts defined in Chapter 5. This prototype implements the auto-attendant use case and

follows the deployment strategy summarized in Figure 14 and the signalling details defined in Figure 16 and Figure 17.

The combination of the raw data analysis captured while running the IVRObject prototype, the IVRObject concepts defined in Chapter 5, and the resulting signalling observed (Appendix C) have given us sufficient information to assess the IVRObject approach against the evaluation criteria defined in section 3.1, whose results were given in section 6.2.

IVRObject has proven to be a viable option, and has scored quite well in two of the criteria evaluated, namely ease of development and portability.

The next chapter will summarize and put into perspective all of the 3 approaches analysed so far (the M*ML approach evaluated in section 4.1.4, the VXML approach evaluated in section 4.2.4, and the IVRObject evaluated in section 6.2), giving a better understanding on how each alternative compares with the others.

Chapter 7. Comparison and Analysis of Alternatives

Based on the evaluation results for the two state-of-the-art approaches (M*ML evaluated in section 4.1.4, and VXML in section 4.2.4), and based on the evaluation of the proposed IVRObject alternative (section 6.2), this chapter gives a comparative view of all prototypes, providing a summary and drawing several conclusions.

7.1. Comparison Summary

The following table gives a summary of all the results analyzed for each alternative for the 3 different criteria and sub-criteria:

Table 14 Criteria and Sub-criteria Comparison Summary

Criteria	Sub-Criteria	M*ML	VoiceXML	IVRObject
Criterion-1) Ease of Development	Simplicity of IVR Command Request Generation and Response Parsing	1 point (Low)	2 points (Neutral)	3 points (High)
	Signalling Simplicity	1 point (Low)	3 points (High)	2 points (Neutral)
	Ease of SIP Unit Testing	1 point (Low)	2 points (Neutral)	3 points (High)
	Central Development	3 points (High)	1 point (Low)	3 points (High)
	Call-back Mechanism	3 points (High)	1 point (Low)	3 points (High)
Criterion-2) Portability		1 point (Low)	2 points (Neutral)	3 points (High)
Criterion-3) Signalling Load		6.3% (Neutral)	7.2% (Neutral)	9.8% (Neutral)

Table 15 highlights the average score for each of the alternatives based only on the 3 main criteria:

Table 15 Comparison Summary

Criteria	M*ML	VoiceXML	IVRObject
Criterion-1) Ease of Development	1.8 points	1.8 points	2.8 points
Criterion-2) Portability	1 point	2 points	3 points
Criterion-3) Signalling Load	6.3%	7.2%	9.8%

Flexibility, development speed and portability are often conflicting with runtime speed and scalability in generic terms. In Table 15, we can clearly see this into play for all three alternatives, where the M*ML alternative had the lowest scores for criteria 1 and 2 (both of them in the "flexibility and portability" side) but had the best evaluation for criterion 3. A result in line with this common trade-off can be observed for the IVRObject alternative, but now on the other side of the spectrum, where it presented the highest scores for criteria 1 and 2 but the weakest evaluation for criterion 3. For the VoiceXML approach, the results are still in line with this trade-off as they present roughly a half way mark of the results for other two approaches.

This trade-off is the object of the analysis of the next section.

7.2. Network Traffic Analysis Summary

This section takes a closer look behind the Signalling Load numbers, as this represented the lowest score for the IVRObject evaluation, in order to try to indentify weather there is any specific problem.

All the data summarized in the following figures were captured during the evaluation of each of the approaches being compared in this work. More specifically, they were taken from Table 4 (for the M*ML approach), Table 7 (for the VXML approach), and Table 12 (for the IVRObject approach).

Figure 24 shows the number of messages that were needed across the network in order to support the IVR application for the different alternatives. Note that from the perspective of the total number of messages, there is no significant difference among the different implementations:

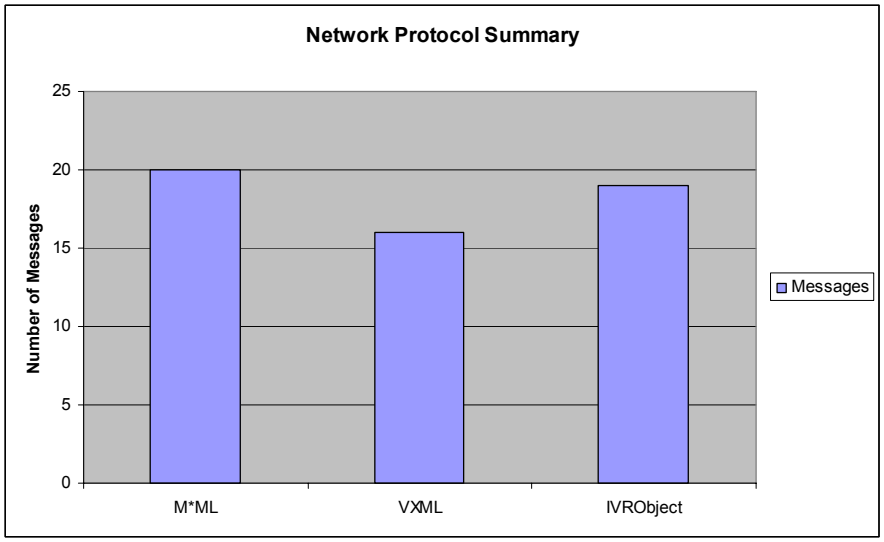

Figure 24 Number of Messages Comparison

Figure 25 breaks down the total number of messages by protocol to illustrate the different message types needed for each approach. Note that M*ML does not use RMI (it uses mostly SIP), that IVRObject uses twice as many RMI messages as VXML does, and that M*ML uses half the number of HTTP messages needed by VXML.

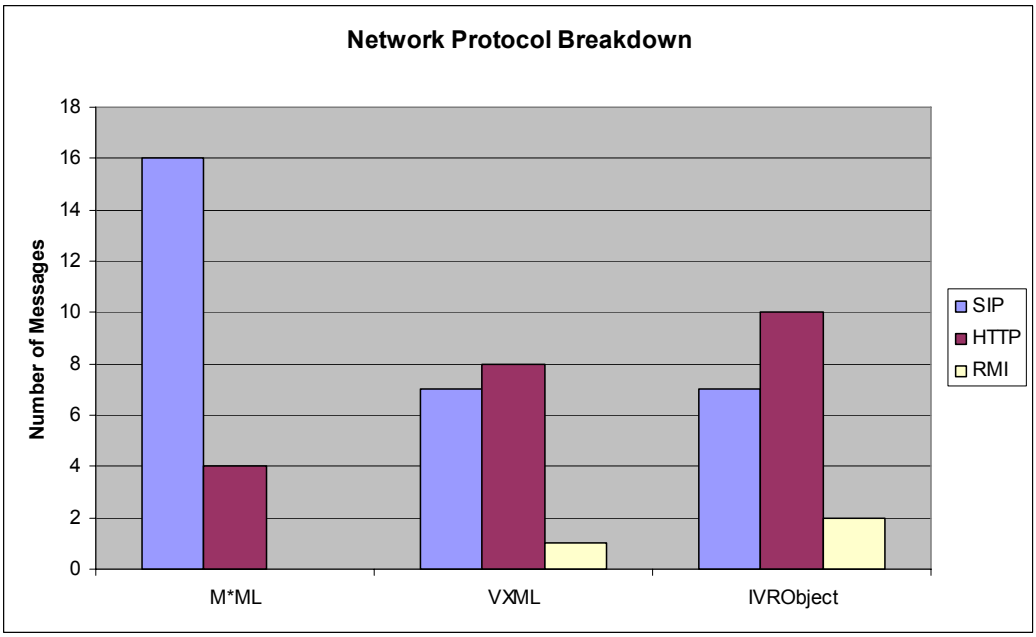

Figure 25 Number of Messages Breakdown Comparison

Figure 26 compares the different alternatives from the number of bytes needed to implement our sample IVR application from a signalling perspective (not taking into consideration the RTP load), that is, how many bytes were needed by the messages shown in Figure 24. Note that IVRObject requires 60% more network signalling traffic than M*ML, and 39% more than VXML to achieve the same auto-attendant IVR functionality:

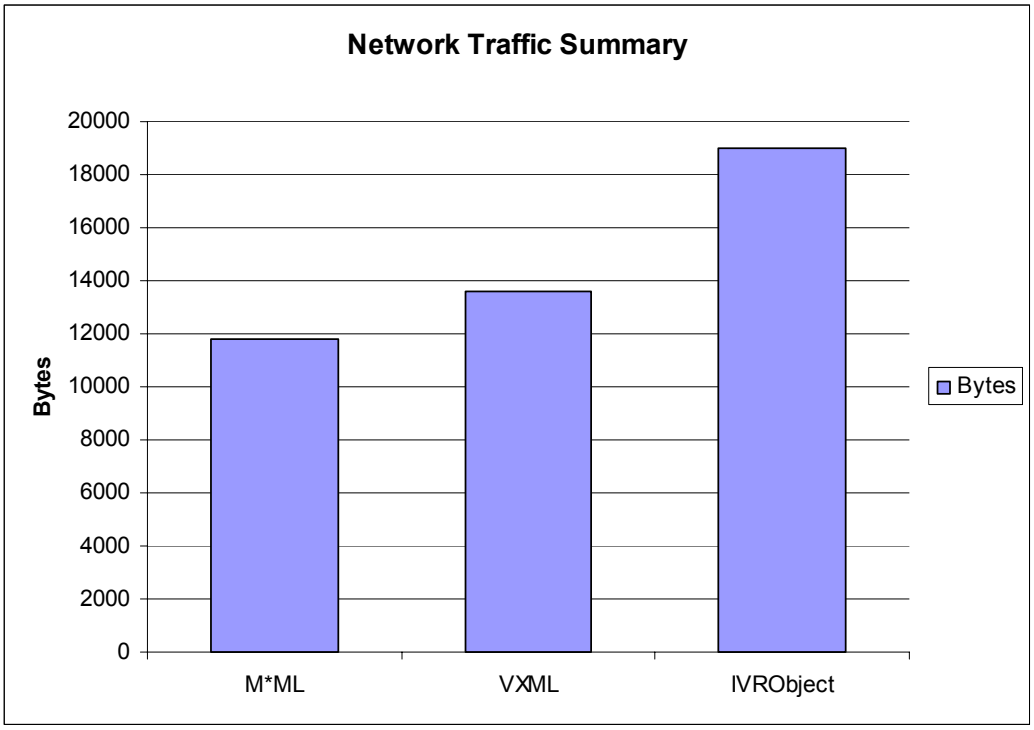

Figure 26 Number of Bytes Comparison

Figure 27 compares the different alternatives from the number of bytes needed to implement our sample IVR application from the signalling perspective, but in comparison to the average RTP load for a call of 25 seconds. Note that the signalling represents a small fraction for all the 3 approaches.

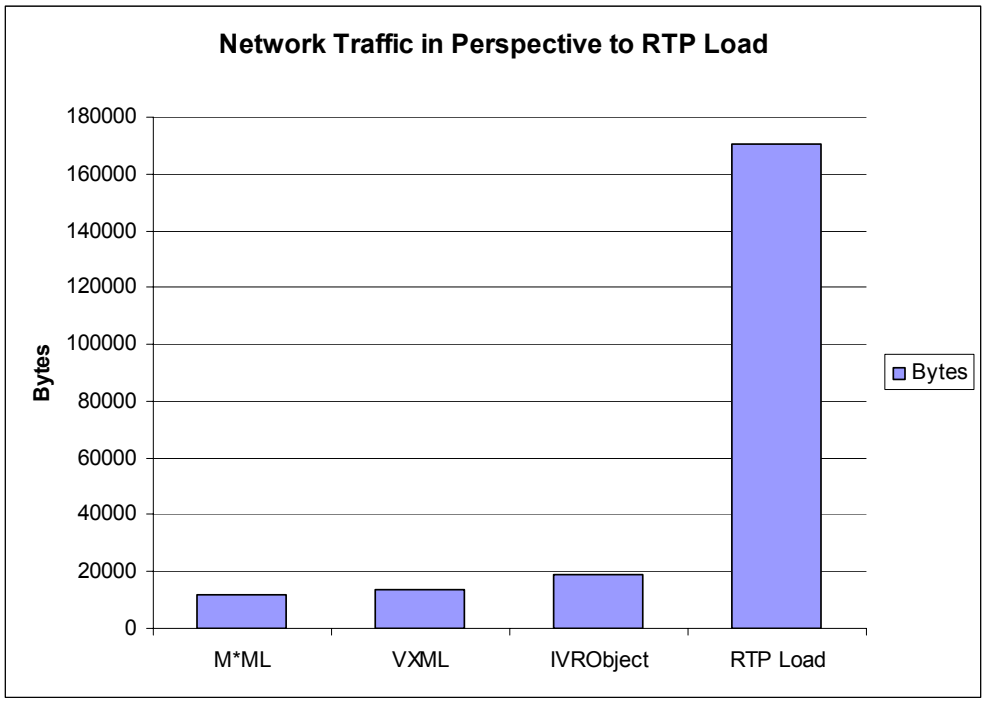

Figure 27 RTP Load Perspective

Figure 28 breaks down the total number of bytes needed by each IVR per protocol. We can observe that M*ML barely uses HTTP (it uses mostly SIP), and that IVRObject uses twice as many HTTP messages as VXML does. For RMI the same load is basically observed for both VXML and IVRObject – as a second RMI call (subsequent call) is much lighter than the first one. M*ML uses SIP more than the others.

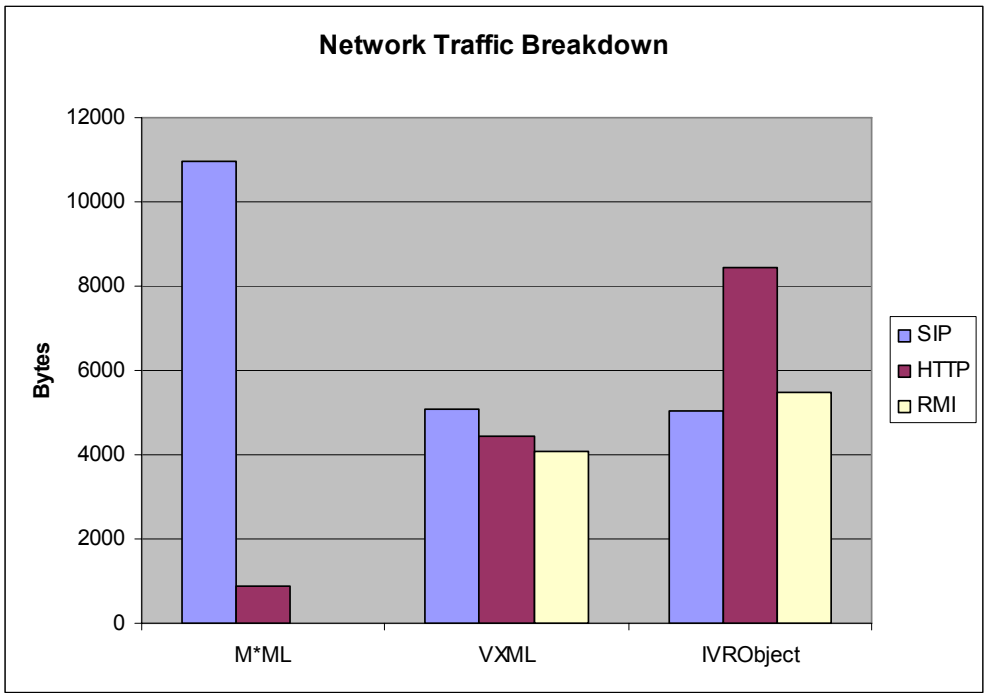

Figure 28 Number of Bytes Breakdown Comparison

We can conclude from the diagrams above that:

- Despite of using different protocols at times, and sometimes using one more than the others, the quantity of messages required by different approaches are roughly the same.
- When put into perspective against the RTP load, the signalling load is negligible, and no option has a clear advantage over the others, where one could be chosen instead of the others because of lighter or heavier signalling load, this criterion is then scored as 'neutral' for all of the three approaches.

We would also like to emphasize the CommandGroup in IVRObject as a mechanism to minimize the number of messages. So, wIt demonstrates how expensive the call-back mechanism between the WS and the SIPAS is. Whenever possible, the IVRObject should be used more efficiently by making use of the CommandGroup feature of the IVRObject (detailed in section 5.2.1, and defined in Appendix D) to bundle sequential IVR commands that do not require business logic in between them, in order to minimize the number of call-backs.

Although we have not noticed any latency in the playing of the prompts in any of the approaches, we have not formally done network latency measurements during this study. However, we believe that IVRObject should rate somewhere between the VXML alternative, which requires VXML reload once it needs business logic (fewer needed), and the M*ML alternative where there is heavy signalling due to its support of a single atomic operation at a time. Therefore, the IVRObject, especially when making usage of the CommandGroup, should require fewer callbacks to the SIPAS than M*ML.

7.3. Chapter Summary

This chapter gave us a comparison summary of the three approaches studied, discussed the trade-off that exists between how easy it is to develop and port an IVR application and its signalling load, and took a closer look at the signalling load as the IVRObject scored low on it, although it had the best score for the other two criteria, and we concluded that the signalling load cannot be considered as a differentiator for the approaches as it is negligible once considered against the RTP load.

The next chapter wraps up this study by drawing general conclusions, recalling the thesis contributions, and suggesting future work items that can be done in order to improve upon the ideas and concepts introduced here.

Chapter 8. Conclusions

Based on development, prototyping, and evaluation of the three alternative approaches to IVR development studied in this thesis (M*ML, VoiceXML and IVRObject), this section draws general conclusions and identifies future work items.

8.1. Conclusions

Some of the key conclusions around the proposed IVRObject approach that can be drawn from this study are:

- IVRObject has the potential for providing the quickest application development due to: a) its strength in not requiring generation/parsing of IVR commands; b) relatively simple signalling involved; c) its central development nature; and d) an existing built-in call-back mechanism that developers can simply use and leverage.
- IVRObject has the potential to be as scalable as the other approaches, as the signalling load for all of them is small compared to the RTP load.

8.2. Contributions

This thesis contributed the IVRObject approach, which builds on the best practices of two popular approaches (M*ML-based and VXML-based) by providing the IVR application running in a SIP application server a simple API to use. This promotes applications that are portable to different media servers, with light SIP signalling, that are easy to test, and that require development in a central component only. This makes the IVRObject approach a true alternative for IVR development and especially for enterprise-based applications where the call volume ranges from low to moderate.

In addition to the IVRObject framework and API, this thesis also contributed prototypes based on an auto-attendant application for the three approaches studied, together with a comparative assessment.

8.3. Future Work

The following are relevant future work items related to the IVRObject approach:

- Study the performance impact of IVRObject by having all the three approaches stress-tested to determine the exact limitations of each.
- Further enhance the IVRObject API to also include common alternative scenarios. For example, when requesting a CommandGroup with a play and collect, also include alternative prompts if the user does not answer a valid input or if she did not enter any input at all, as at times these messages are slightly different from the first message played. This would produce a richer VXML and further reduce the number of callbacks to SIPAS.
- Extend the IVRObject API in order to also support Text-to-Speech.
- Explore the extension of IVRObject to also support Interactive Video and Voice Response (IVVR) by having the Web server generate the proper Synchronized Multimedia Integration Language (SMIL [5]) to define the IVVR commands.
- Explore the usage of JSR-309, the Media Server Control API [3], as a possible implementation of the IVRObject API.
- Explore the converged container aspect of the JSR-289 [8] that specifies a combined HTTP and SIP container, and evaluate whether IVRObject can benefit from this and possibly optimize the call-back mechanism (by substituting the RMI to HTTP).
- This work has focused on a SIP-centric call control. We could also explore alternatives such as CCXML [1][2] to invoke IVRObject, and take advantage of the built in mechanism CCXML has for VXML call-backs.

References

All URLs have been last accessed in January 2009.

[1] Amyot, D. and Simoes, R.: Combining VoiceXML with CCXML. *IEEE Consumer Communications & Networking Conference (CCNC 2007)*, Las Vegas, USA, January, 342-346.

[2] Auburn, R., et al.: *Voice Browser Call Control: CCXML*, W3C Working Draft, January 2007

[3] Brandt, M. and Ericson, T.: *Media Server Control API*, Java Community Process, Draft JSR 309, October 2008.

[4] Burger, E.: *Media Server Control Language and Protocol Thoughts*, IETF Network Working Group, June 2006.

[5] Bulterman, D. et al.: *Synchronized Multimedia Integration Language (SMIL 3.0)*, W3C Recommendation, December 2008.

[6] Cisco, Document ID 7934, *Voice Over IP - Per Call Bandwidth Consumption*, February 2006. http://www.cisco.com/en/US/tech/tk652/tk698/technologies_tech_note09186a0080094ae2.shtml

[7] Combs, G.: *Wireshark*, http://www.wireshark.org

[8] Cosmadopoulos, Y. and Kulmaki, M.: *SipServlet API, Version 1.1*. Java Community Process, JSR-289, August 2008.

[9] Donavan, S.: *The SIP INFO Method*, IETF, RFC 2976, October 2000.

[10] Dyke, J., Burger, E., and Spitzer, A.: *Media Server Control Markup Language (MSCML) and Protocol*, IETF, RFC 4722, November 2006.

[11] Fielding, R.: *Hypertext Transfer Protocol – HTTP/1.1*, W3C Specification, 1999.

[12] Gold, R., et al.: *HttpUnit*, http://httpunit.sourceforge.net

[13] Handley, M., Jacobson, V., and Perkins, C.: *SDP: Session Description Protocol*, IETF, RFC 4566, July 2006.

[14] *Java Remote Method Invocation (Java RMI)*, URI: http://java.sun.com/docs/books/tutorial/rmi/index.html

[15] Kristensen, A.: *SipServlet API, Version 1.0*. Java Community Process, JSR-116, February 2003.

[16] McElroy, B, et al.: *SipUnit*, http://www.cafesip.org/projects/sipunit/index.html

[17] McGlasham, S., et al.: *Voice Extensible Markup Language (VoiceXML) Version 2.0*, W3C Recommendation, March 2004.

[18] Raggett, D., et al.: *HyperText Markup Language (HTML) Specification*, W3C Recommendation, December 1999.

[19] Rosenberg, J.: *The Session Initiation Protocol (SIP) UPDATE Method*, IETF, RFC 3311, September 2002.

[20] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and Schooler, E.: *SIP: Session Initiation Protocol*, IETF, RFC 3261, June 2002.

[21] Rosenberg, J., et al.: *Best Current Practices for Third Party Call Control in the Session Initiation Protocol*, draft-ietf-sipping-3pcc-06, December 2003

[22] Saleem, A. and Sharratt, G.: *Media Objects Markup Language (MOML)*, IETF SIPPING Internet Draft, draft-melanchuk-sipping-moml-06, October 2005.

[23] Saleem, A., Xin, Y. and Sharratt, G.: *Media Server Markup Language (MSML)*, IETF Tools Draft, August 2008.

[24] Schulzrinne, H. et al.: *RTP: Real-time Transport Protocol*, IETF, RFC 1889, January 1996.

[25] Voip-Info: *Terminal Adaptor*, http://www.voip-info.org/wiki/view/Analog+Telephone+Adapters

[26] *World Wide Web Consortium, W3C*, URI: http://www.w3.org

Appendix A: M*ML - SIP Trace

The following SIP call trace was captured during the evaluation of the M*ML approach to support the IVR capabilities, as detailed in section 4.1. During this test, a MSML-capable media server was used. Given its size, this trace is only available online, at: http://www.site.uottawa.ca/~damyot/students/simoes/AppendixA.txt.

Appendix B: VXML with RMI Call-back - SIP Trace

The following SIP call trace was captured during the evaluation of the VoiceXML approach to support the IVR capabilities, as detailed in section 4.2. Given its size, this trace is only available online, at:
http://www.site.uottawa.ca/~damyot/students/simoes/AppendixB.txt.

Appendix C: IVR Object - SIP Trace

The following SIP call trace was captured during the evaluation of the IVRObject approach to support the IVR capabilities, as detailed in Chapter 5. Given its size, this trace is only available online, at:
http://www.site.uottawa.ca/~damyot/students/simoes/AppendixC.txt.

Appendix D: IVRObject Class Diagram and API

This appendix details the IVRObject class diagram and its API.

The IVRObject class diagram shown in Figure 29 gives an overview of the classes that make part of the IVRObject. The developer starts by getting hold of the IVRObjectFactory via the static method *getIvrObjectFactory()*. Getting hold of the IVRObject instance (singleton implementation) allows the developer to start creating the different IVR commands: play (via *createIVRObjectPlayCommand()*), collect (via *createIVRObjectCollectCommand()*), and record (via *createIVRObjectRecordCommand()*).

All three IVR commands (IVRObjectPlayCommand, IVRObjectCollectCommand, and IVRObjectRecordCommand) extent the IVRObjectCommand, which is capable of running a command.

IVRObjectFactory also provides a way for creating a command group (via *createIVRObjectCommandGroup()*). The IVRObjectCommandGroup allows the developer to add IVRObjectCommand commands of any type that are to be run in sequence.

When a command or command group is run (via the *runIVRObjectCmd()*), an instance that implements the IVRObjectListener needs to be provided by the custom IVR application to be called back on IVR events (prompt played, error playing prompt, and others).

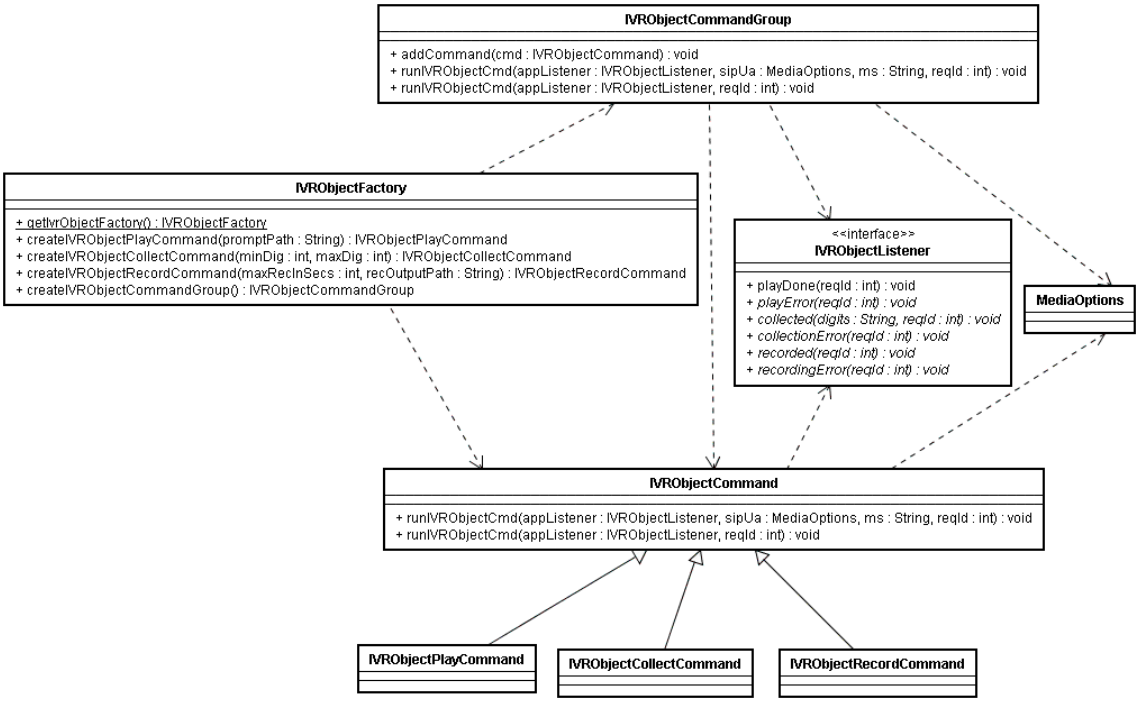

Figure 29 IVRObject Class Diagram

Figure 30 presents the IVRObjectFactory class API.

ivrobject

Class IVRObjectFactory

```
java.lang.Object
  └ ivrobject.IVRObjectFactory
```

```
public class IVRObjectFactory
extends java.lang.Object
```

The IVRObjectFactory provides a way for creating the IVR commands and command group.

Author:

Renato Simoes

Constructor Summary
IVRObjectFactory()

Method Summary	
IVRObjectCollectCommand	**createIVRObjectCollectCommand**(int minDig, int maxDig) From the factory, creates a collect DTMF digit command object
IVRObjectCommandGroup	**createIVRObjectCommandGroup**() Creates a place holder for grouping commands that are to run in sequence.
IVRObjectPlayCommand	**createIVRObjectPlayCommand**(java.lang.String promptPath) From the factory, creates a play command object
IVRObjectRecordCommand	**createIVRObjectRecordCommand**(int maxRecInSecs, java.lang.String recOutputPath) From the factory, creates a record command object
static IVRObjectFactory	**getIvrObjectFactory**() Singleton implementation.

Figure 30 IVRObject API – The IVRObjectFactory Class

Figure 31 presents the IVRObjectCommand class API.

ivrobject

Class IVRObjectCommand

```
java.lang.Object
  └ ivrobject.IVRObjectCommand
```

Direct Known Subclasses:

IVRObjectCollectCommand, IVRObjectPlayCommand, IVRObjectRecordCommand

---

```
public class IVRObjectCommand
extends java.lang.Object
```

Base class for the IVR commands.

Author:

Renato Simoes

Method Summary	
void	**runIVRObjectCmd**(IVRObjectListener appListener, int reqId) Runs a specific (subsequent) IVR command
void	**runIVRObjectCmd**(IVRObjectListener appListener, MediaOptions sipUa, java.lang.String ms, int reqId) Runs a specific (initial) IVR command

Figure 31 IVRObject API – The IVRObjectCommand Class

Figure 32 presents the IVRObjectCommandGroup class API.

ivrobject

Class IVRObjectCommandGroup

```
java.lang.Object
  └ ivrobject.IVRObjectCommandGroup
```

```
public class IVRObjectCommandGroup
extends java.lang.Object
```

Place holder for grouping commands that are to run in sequence

Author:

Renato Simoes

Constructor Summary

IVRObjectCommandGroup()

Method Summary

void	**addCommand**(IVRObjectCommand cmd) Adds a command to the command group.
void	**runIVRObjectCmd**(IVRObjectListener appListener, int reqId) Runs a specific (subsequent) IVR command group
void	**runIVRObjectCmd**(IVRObjectListener appListener, MediaOptions sipUa, java.lang.String ms, int reqId) Runs a specific (initial) IVR command group

Figure 32 IVRObject API – The IVRObjectCommandGroup Class

Figure 33 presents the IVRObjectPlayCommand class API.

ivrobject

Class IVRObjectPlayCommand

```
java.lang.Object
  └ ivrobject.IVRObjectCommand
      └ ivrobject.IVRObjectPlayCommand
```

```
public class IVRObjectPlayCommand
extends IVRObjectCommand
```

The command to request a prompt to be played to the end-user.

Author:

Renato Simoes

Method Summary	
java.lang.String	**getPromptPath**() Getter for the promptPath

Methods inherited from class ivrobject.IVRObjectCommand
runIVRObjectCmd, runIVRObjectCmd

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Figure 33 IVRObject API – The IVRObjectPlayCommand Class

Figure 34 presents the IVRObjectCollectCommand class API.

ivrobject

Class IVRObjectCollectCommand

```
java.lang.Object
  └ ivrobject.IVRObjectCommand
      └ ivrobject.IVRObjectCollectCommand
```

```
public class IVRObjectCollectCommand
extends IVRObjectCommand
```

The command to request DTMF digits to be collected from the end-user.

Author:
Renato Simoes

Method Summary	
int	**getMaxDig**() Getter for the maxDig
int	**getMinDig**() Getter for the minDig

Methods inherited from class ivrobject.IVRObjectCommand
runIVRObjectCmd, runIVRObjectCmd

Figure 34 IVRObject API – The IVRObjectCollectCommand Class

Figure 35 presents the IVRObjectRecordCommand class API.

ivrobject

Class IVRObjectRecordCommand

```
java.lang.Object
  └ ivrobject.IVRObjectCommand
      └ ivrobject.IVRObjectRecordCommand
```

```
public class IVRObjectRecordCommand
extends IVRObjectCommand
```

The command to record the end-user.

Author:

Renato Simoes

Method Summary	
int	**getMaxRecInSecs**() Getter for the maxRec in seconds
java.lang.String	**getRecOutputPath**() Getter for the recording output path

Methods inherited from class ivrobject.IVRObjectCommand
runIVRObjectCmd, runIVRObjectCmd

Figure 35 IVRObject API – The IVRObjectRecordCommand Class

Figure 36 presents the IVRObjectListener class API.

ivrobject

Interface IVRObjectListener

`public interface IVRObjectListener`

The application listener, that will be called back after an IVR operation

Author:

Renato Simoes

Method Summary	
void	**collected**(java.lang.String digits, int reqId) Reports that a DTMF digit collection request (issued via the IVRObjectCollectCommand) was successful.
void	**collectionError**(int reqId) Reports that a DTMF digit collection request (issued via the IVRObjectCollectCommand) was not successful.
void	**playDone**(int reqId) Reports that a play request (issued via the IVRObjectPlayCommand) was successful.
void	**playError**(int reqId) Reports that a play request (issued via the IVRObjectPlayCommand) was not successful.
void	**recorded**(int reqId) Reports that a recording request (issued via the IVRObjectRecordCommand) was successful.
void	**recordingError**(int reqId) Reports that a recording request (issued via the IVRObjectRecordCommand) was not successful.

Figure 36 IVRObject API – The IVRObjectListener Class

Figure 37 presents the MediaOptions class API.

ivrobject

Class MediaOptions

```
java.lang.Object
  └ ivrobject.MediaOptions
```

```
public class MediaOptions
extends java.lang.Object
```

Bean for storing the media options of an end-user that will participate in an IVR session

Author:

Renato Simoes

Constructor Summary
`MediaOptions(java.lang.String contentType, byte[] sdpContentBody)` Creates the media options object

Method Summary	
`java.lang.String`	`getContentType()` Getter for the content typ
`byte[]`	`getSdpContentBody()` Getter for the content body - SDP

Figure 37 IVRObject API – The MediaOptions Class

Appendix E: IVRObject Test Call-back Driver API

This appendix details the classes and illustrates the API usage for the "IVRObject Test Call-back Driver" that is part of the IVRObject framework for assisting the automated tests used to simulate IVR events.

The class diagram shown in Figure 38 gives an overview of the classes that make part of the IVRObject Test Call-back Driver. The test developer starts by getting hold of the IVRObjectTestCallbackFactory via the static method *getIvrObjectTestCallbackFactory()*. Getting hold of the IVRObjectTestCallbackFactory instance (singleton implementation) allows the test developer to start creating the simulation of the different IVR command responses to be sent to the IVRObject API Test Impl (refer to Figure 18 for the big picture) such as: play result (via *createIVRObjectPlayCommandResult()*), collect result (via *createIVRObjectCollectCommandResult()*), and record result (via *createIVRObjectRecordCommandResult()*).

All the three IVR command result simulation (IVRObjectPlayCommandResult, IVRObjectCollectCommandResult, and IVRObjectRecordCommandResult) extend the IVRObjectCommandResult which is capable of sending a call-back carrying the simulation of an IVR command result.

IVRObjectTestCallbackFactory also provides a way for creating a command group result (via *createIVRObjectCommandGroupResult()*). The IVRObjectCommandGroupResult allows the developer to add IVRObjectCommandResult of any type, that are to be sent back to the IVRObject API Test Impl via *sendIVRObjectCmdResult()*.

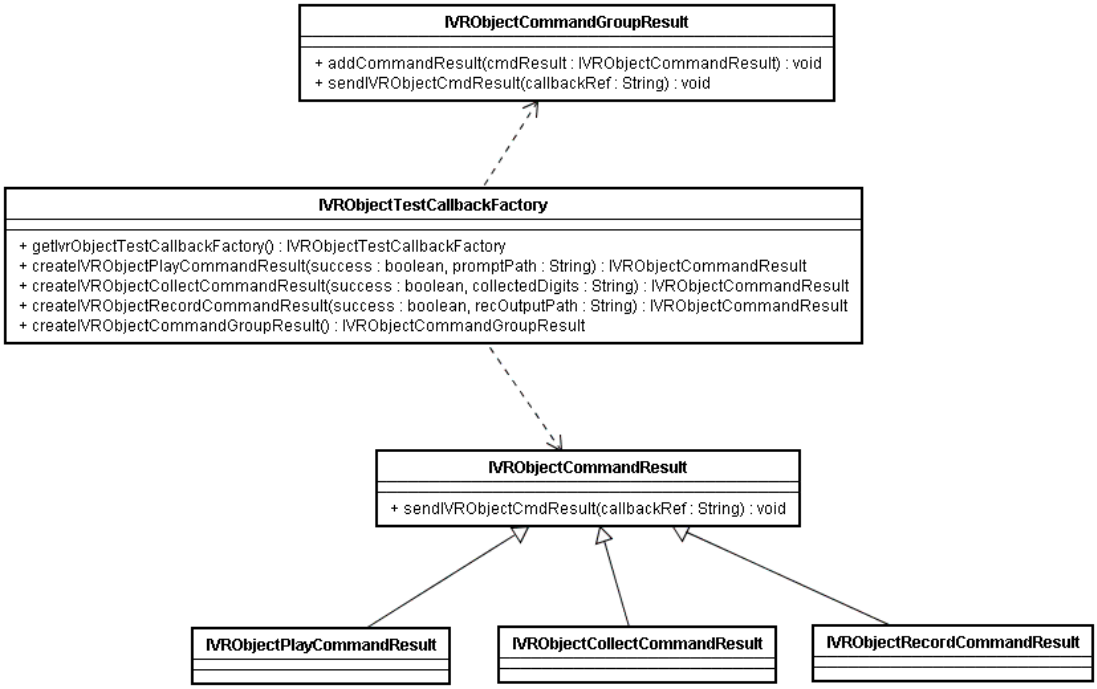

Figure 38 IVRObject Test Call-back Driver Class Diagram

Figure 39 shows a sequence diagram that details the use of the IVRObject Test Call-back Driver. These messages can be seen as a detailed version of messages 13 and 13.1 previously shown in Figure 19.

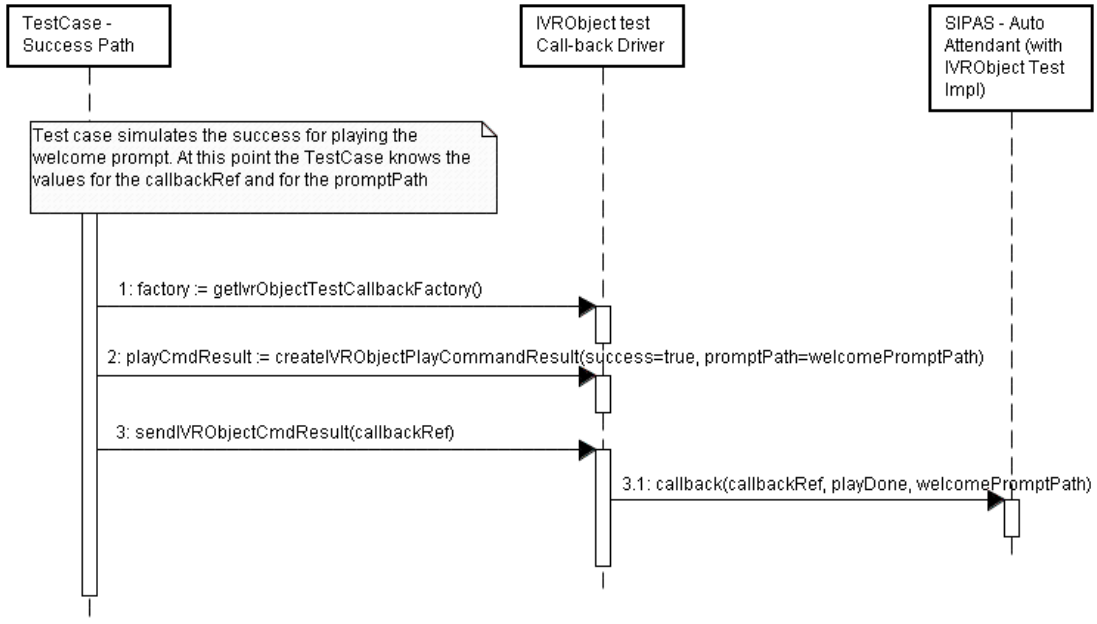

Figure 39 IVRObject Test Call-back Driver Sample Usage

Appendix F: IVRObject Auto-Attendant Sample Java Code

This section illustrates the auto-attendant implementation using the IVRObject API.

Note that although IVRObject is language independent where its concept can have the API developed in a multitude of languages, for illustration purposes, Java was chosen here.

In the code, steps refer to signalling messages that were detailed in Figure 16 and Figure 17, so one can correlate the code steps with these diagrams.

```
package ivrobject.autoattendant;

import ivrobject.IVRObjectCollectCommand;
import ivrobject.IVRObjectCommandGroup;
import ivrobject.IVRObjectFactory;
import ivrobject.IVRObjectListener;
import ivrobject.IVRObjectPlayCommand;
import ivrobject.MediaOptions;

public class AutoAttendant implements IVRObjectListener {

	private static final int WELCOME_PROMPT_REQ = 0;
	private static final int PROMPT_AND_COLLECT_REQ = 1;

	//this is the initial auto-attendant invocation
	//this is invoked from a SipServlet code
	public void runAutoAttendant(MediaOptions sipUa, String ms)
	{
		IVRObjectFactory ivrObjectFactory =
			IVRObjectFactory.getIvrObjectFactory();

		//Step-2.1
		IVRObjectPlayCommand playWelcomeCmd =
			ivrObjectFactory.createIVRObjectPlayCommand
			("welcome.wav");

		//Step-2.2
		playWelcomeCmd.runIVRObjectCmd(this, sipUa, ms,
					WELCOME_PROMPT_REQ);
	}

	/**
```

```
     * Implementation of IVRObjectListener
     * @see ivrobject.ivrobjectlistener#playDone()
     * Reports that a play request (issued via the
     * IVRObjectPlayCommand)
     * was successful.
     * @param reqId the id that identifies this request
     */
    public void playDone(int reqId) {

          switch (reqId) {

                case WELCOME_PROMPT_REQ:
                {
                      // Welcome prompt was played,
                      //now request the play and collection
                      //of extension

                      IVRObjectFactory ivrObjectFactory =
                            IVRObjectFactory.getIvrObjectFactory();

                      //Step-6
                      IVRObjectPlayCommand playEnterExtCmd =
                            ivrObjectFactory.
                            createIVRObjectPlayCommand
                            ("enterExtension.wav");

                      //Step-7
                      IVRObjectCollectCommand collectExtCmd =
                            ivrObjectFactory.
                            createIVRObjectCollectCommand(1, 4);

                      //Step-8
                      IVRObjectCommandGroup playAndCollectCmdGroup =
                            ivrObjectFactory.
                            createIVRObjectCommandGroup();

                      //Step-9
                      playAndCollectCmdGroup.
                            addCommand(playEnterExtCmd);

                      //Step-10
                      playAndCollectCmdGroup.
                            addCommand(collectExtCmd);

                      //Step-11
                      playAndCollectCmdGroup.runIVRObjectCmd(this,
                                        PROMPT_AND_COLLECT_REQ);

                }

                case PROMPT_AND_COLLECT_REQ:
                {
                      //Step-17.1.1
                      System.out.println("enterExtension.wav was
                            played, nothing to do here");
                }
          }
```

```
}

/**
 * Implementation of IVRObjectListener
 * @see ivrobject.ivrobjectlistener#playError()
 * Reports that a play request (issued via the
 * IVRObjectPlayCommand)
 * was not successful.
 * @param reqId the id that identifies this request
 */
public void playError(int reqId) {

      //one might decide to tear down the call
      //this requires invoking SipServlet API commands
      //this part is common to any of the approaches
      //code not relevant from a comparison perspective
}

/**
 * Implementation of IVRObjectListener
 * @see ivrobject.ivrobjectlistener#collected(java.lang.string)
 * Reports that a DTMF digit collection request (issued via the
 * IVRObjectCollectCommand) was successful.
 * @param digits
 * @param reqId the id that identifies this request
 */
public void collected(String digits, int reqId) {

      //Step-17.1.2

      System.out.println("Caller entered extension: " + digits);

      //this requires invoking SipServlet API commands
      //this part is common to any of the approaches
      //code not relevant from a comparison perspective
}

/**
 * Implementation of IVRObjectListener
 * @see ivrobject.ivrobjectlistener#collectionError()
 * Reports that a DTMF digit collection request (issued via the
 * IVRObjectCollectCommand) was not successful.
 * @param reqId the id that identifies this request
 */
public void collectionError(int reqId) {

      //one might decide to tear down the call
      //this requires invoking SipServlet API commands
      //this part is common to any of the approaches
      //code not relevant from a comparison perspective
}

/**
 * Implementation of IVRObjectListener
 * @see ivrobject.ivrobjectlistener#recorded()
 * Reports that a recording request (issued via the
 * IVRObjectRecordCommand) was successful.
```

```
         * @param reqId the id that identifies this request
         */
        public void recorded(int reqId) {

              //not called, there is no recording in the auto-attendant
              //use case
        }

        /**
         * Implementation of IVRObjectListener
         * @see ivrobject.ivrobjectlistener#recordingError()
         * Reports that a recording request (issued via the
         * IVRObjectRecordCommand) was not successful.
         * @param reqId the id that identifies this request
         */
        public void recordingError(int reqId) {

              //not called, there is no recording in the auto-attendant
use case
        }
}
```