A Virtualized Infrastructure for IVR Applications as Services

Christian Azar

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

November 2011

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Christian Azar

Entitled: "A Virtualized Infrastructure for IVR Applications as Services"

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

____ Chair
Dr. R. Raut

____ Examiner, External To the Program
Dr. J. Rilling, CSE

____ Examiner
Dr. F. Khendek

____ Supervisor
Dr. R. Glitho

Approved by: ____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

____________20_____ ______________________________________
Dr. Robin A. L. Drew
Dean, Faculty of Engineering and Computer Science

ABSTRACT

A Virtualized Infrastructure for IVR Applications as Services

Christian Azar

Interactive Voice Response (IVR) applications are ubiquitous nowadays. Automated attendant, bank teller and automated surveys are a few of many applications requiring IVR capabilities. Cloud computing is a paradigm gaining a lot of momentum. It has three major service models: Infrastructure as a service – IaaS, Platform as a service – PaaS, and Software as a Service – SaaS. It offers also several inherent benefits such as scalability, resource efficiency and easy introduction of new functionality. However, very few, if any, IVR applications are offered today in cloud-based settings despite of all its potential benefits.

This thesis deals with IaaS. Accordingly, we propose a novel architecture for a virtualized IVR infrastructure that relies on RESTFul Web services. The architecture proposes IVR substrates that are virtualized, composed, and assembled on the fly to build IVR applications. As a proof of concept, we have implemented an IaaS prototype on which performance measurements have been done to evaluate our architecture concept. In addition, a simple proof of concept PaaS consisting of a graphical user interface (GUI) has been built to enable the development and management of simple IVR services in the SaaS layer.

Acknowledgments

This thesis is a part of research work done in the Telecommunication Service Engineering (TSE) research lab. It is a great pleasure to express my sincere gratitude to all the persons who helped me throughout my thesis.

First and foremost, I owe my deepest gratitude to my supervisor Dr. Roch Glitho for all his support, assistance and guidance. This work would not have been possible without his encouragement, advices and countless drafts reviewed. It was a great pleasure working and learning from him during both research and thesis writing periods.

I'm grateful to Dr. Khendek, Dr. Raut, and Dr. Rilling for serving as members of my thesis committee and for their valuable comments and feedbacks on my thesis.

I also offer my gratitude to Dr. Fatna Belqasmi and Dr. Nadjia Kara for their collaborations, supports, and ideas throughout my research work. It is been a real pleasure working with you.

I am thankful for my colleagues at the TSE lab and all my friends in Canada for all their helps and suggestions. I have been fortunate to work with you.

Furthermore, I would like to acknowledge Concordia University and Dr. Roch Glitho for their financial support during my research study at Concordia.

Finally, the most special thanks go to my dearest family and my beloved girlfriend for their love and faith in me.

Thank you. This work is truly also yours!

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

API	Application Programming Interface
AS	Application Server
CPU	Central Processing Unit
DSM	Donkey State Machine
GUI	Graphical User Interface
HTTP	Hyper Text Transport Protocol
IaaS	Infrastructure as a Service
InfP	Infrastructure Provider
IP	Internet Protocol
ISS	IVR Service Substrate
IVR	Interactive Voice Response
JSON	JavaScript Object Notation
MS	Media Server
MSCML	Media Server Control Markup Language
OS	Operating System
PaaS	Platform as a Service
PP	Platform Provider
REST	Representational State Transfer
RTP	Real Real-time Transport Protocol
RTT	Round Trip Time
SaaS	Software as a Service

SEMS	SIP Express Media Server
ServP	Service Provider
SII	Substrate IVR Instance
SIP	Session Initiation Protocol
TTS	Text To Speech
URI	Uniform Resource Identifier
VMM	Virtual Machine Monitor
VoIP	Voice over IP
VXML	Voice Extensible Markup Language
WADL	Web Application Description Language
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Chapter 1

Introduction

This chapter provides a brief overview of the key concepts and ideas of this thesis. It discusses the motivation and identifies the problems that this research project addresses. Then, it lists the contributions and concludes by summarizing the thesis outline.

1.1 Research Domain

Interactive Voice Response (IVR) is a telephony technology that enables interactions with automated information systems. Its applications are numerous. One example is automated attendants, which replace live receptionists by transferring callers to the appropriate extensions they dial. Another example is automated meter readers that provide fully automated dialogs, which enable utilities customers to remotely enter their meter readings. There are several other examples including automated surveys, bank tellers and clinical trials.

Cloud computing is a promising paradigm with many inherent advantages, such as the easy introduction of new applications, resource efficiency, and scalability. There is not yet a standard technical definition for cloud computing. However, a consensus is emerging around the most critical facets it encompasses: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1].

Service providers use platforms offered as PaaS by platform providers to develop and manage applications on the fly. These applications are provisioned to end-users (or other applications) as SaaS on a pay-per-use basis. PaaS add one or more levels of abstraction to the infrastructures offered as IaaS by infrastructure providers. It provides environments that ease development and management of applications at the SaaS layer. Infrastructures are the actual dynamic pool of virtualized resources used to deploy the applications and run the operating systems.

Virtualization is the basis of cloud computing [2]. A virtualized infrastructure enables the co-existence of entities in general on the same substrates using software called hypervisor or virtual machine monitor (VMM) [3]. These entities may be operating systems co-existing on the same hardware, applications co-existing on the same operating system, or even full-blown networks co-existing on the same routers. The key characteristic of virtualization is the efficiency provided by maximizing the utilization of the underlying computing resources at the infrastructure layer of the cloud computing paradigm.

1.2 Motivation and Problem Statement

IVR has become an interesting technology in telecommunication applications. Nowadays, many industries rely on IVR applications and profit from its capabilities. In the meanwhile, cloud computing is a fast moving paradigm with a lot of inherent benefits. The motivation of this thesis is to exploit the benefit of cloud computing in developing and managing IVR applications in the cloud. This will provide a novel value added services for IVR service providers such as quick, easy, low-cost and on the fly development of applications at the SaaS layer. In addition, it will enable infrastructure providers at the IaaS layer to dynamically provision IVR substrates offered by different substrate providers. Furthermore, the virtualized infrastructure will give infrastructure providers the ability to instantiate on-demand these substrates and run them simultaneously on a single hardware which will improve the sharing of computing resources.

Several applications are offered today in cloud settings (e.g. enterprise databases, IT help desks). However, these offerings very rarely include IVR applications, despite all the obvious potential benefits, especially efficiency and scalability. One of the reasons is that there may be some lack of knowledge about the design of IaaS for IVR applications.

For example, an announcement player offered as a substrate could be virtualized and shared between different IVR applications such as automated survey, bank teller, and several other applications. It could also be dynamically allocated to these applications for scalability purposes. To the best of our knowledge, there is no full-fledged cloud environment that enables the development, management and offering of the full range of

IVR applications. Therefore, the problem that this thesis will address is how to ease the development and management of IVR applications in cloud with a focus on IaaS. Specifically IaaS will be composed of a set of IVR substrates supplied by different substrate providers.

1.3 Thesis Contributions

Cloud IVR applications are still an open research topic. However, this thesis constitutes a new contribution in this research domain. Hence, the aim of this thesis is to propose a novel architecture that relies on RESTFul Web services for a virtualized IVR infrastructure as a first step towards the deployment of full-fledged IVR applications in cloud-based settings.

The main contributions of this thesis are:

- Requirements for a virtualized infrastructure that ease the development and management of IVR applications as services, which describe our major challenges.

- Review of the most relevant state of the art in our research area with an evaluation summary comparing to our derived requirements.

- A business model which introduces the IVR substrate provider at the infrastructure layer as a new role in the cloud business model.

- A proposed architecture for a virtualized infrastructure for IVR applications as services that focuses on IaaS, relies on the proposed business model, and satisfies all our derived requirements.

- An overall designed software architecture based on the proposed architecture that describes all the entities, planes functionality, interfaces, and operational procedures.

- A proof of concept that demonstrates the architecture's potential and includes a subset of the software architecture along with a simple PaaS graphical user interface.

- Preliminary performance measurements of the implemented prototype along with the results evaluation.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 presents the basic concepts and definitions related to IVR, Cloud Computing, Virtualizations and RESTFul Web services that will illustrate the basic ideas relevant to this thesis.

Chapter 3 presents the derived requirements for having a virtualized infrastructure for IVR applications. Also, in this chapter we review the most relevant state of the art in our research area and evaluate them with respect to the derived requirements.

Chapter 4 introduces the proposed architecture for a virtualized IVR infrastructure along with the business model, the architectural components and all the interfaces.

Chapter 5 is devoted to the software architecture and the design and implementation of our prototype as a proof of concept where an automated attendant scenario is developed using the platform provider GUI. Also, this chapter includes the preliminary performance measurements.

Chapter 6 concludes the thesis by summarizing briefly the overall contributions, and suggests some future work issues.

It is preferable to go through these chapters in sequence. A list of acronyms and abbreviations used in this thesis is provided on page xi.

Chapter 2

Background on IVR, Cloud Computing, Virtualization, and RESTFul Web services

This chapter presents the essential background information on three main subjects covered in this thesis proposition: Interactive Voice Response (IVR), Cloud Computing, and Virtualization. It also gives a brief overview about RESTFul Web services as it is a key technology of our proposed architecture.

2.1 Interactive Voice Response

This section starts by a brief definition of IVR. It states its well-known characteristics and discusses their benefits. It also shows some of the applications that rely on IVR capabilities and concludes by presenting some related IVR challenges. However, although there is no refereed journal papers that fully address IVR benefits, capabilities and applications, this section refers to few papers that partially handle specific function in IVR. In addition it refers to the broadly known information of IVR in the telephony technologies, which will provide readers the required information through the thesis.

2.1.1 Definition

Interactive Voice Response (or simply IVR) is a telephony technology that interacts with end users through a configurable voice system. It is used to automate a wide range of services and data requests [29]. It also offers several functionalities and has various applications. It can be used in telephone counseling, message recording, balance inquiries and other business areas [30]. On the other hand, the caller communicates with IVR using a telephony interface (e.g. cell phone, soft phone) to hear instructions of IVR and the phone keypad or voice commands [29] to respond to the IVR system.

2.1.2 Features

IVR features vary between one IVR systems to another. Usually, most of the IVR systems are designed with specific capabilities and cannot be reusable [30]. However, at a minimum, IVR system may offer the ability to play or record prompts, detect a key, collect an extension, and transfer a call. In other words, an IVR application could be a composition of these features depending on the requirements of each service provider.

IVR may also provide the ability to integrate some other advanced features such as:

- **Database Interaction**

  IVR systems may require database interactivity to write or retrieve information to/from a database to process the request of customers [29].

- **Voice Recognition**

  Provide the ability to translate callers' voice command such as words [29] and numbers into responses understandable and recognized by the IVR system instead using the telephone keypad.

- **Voice to Email**

  This feature takes a voice message from the caller and sends it to a preconfigured email address [29].

- **Voice Messaging**

  This capability gives the ability to caller to leave, retrieve and manage a preconfigured voicemails system.

- **Text to Speech (TTS)**

  TTS engine provide the ability to build the voice files of IVR [31] by simply typing the message in a text field area. It permits for fast changes to a script with a high quality sounding voice. Also, it eliminates the need to record prompts in IVR system.

2.1.3 Benefits

The benefits of IVR applications are endless. By installing an IVR system, not only clients will profit from this service, but also the service providers such as banks, hospitals, and universities. With an IVR system, companies can reduce costs without the need of human agents. Also, they can improve customer's quality of service by providing a self-service [29] to customers 24 hours a day, 7 days a week.

IVR reduce caller wait times, since an IVR can never be busy, which give it the ability to server multiple users at the same time. In addition, IVR system can interact with multiple databases in the same time and using the Text to Speech engine callers can get information such as account balances, shipping details, and other information without any pre-recording messages [29, 30].

Furthermore, using the voice recognition module, callers can spell names, make reservation, and order products which makes customer interaction with the system quick, and convenient [29]. Finally, via the voice to email features, users will have the ability to access their voicemail from any internet machine which offer subscribers mobility and flexibility.

2.1.4 Applications

IVR applications are numerous and ubiquitous nowadays. Banks and credit card companies use IVR systems so that their customers can receive account information easily without having to speak directly to a live agent. Figure 2.1 shows a sample flow chart of a simple bank IVR.

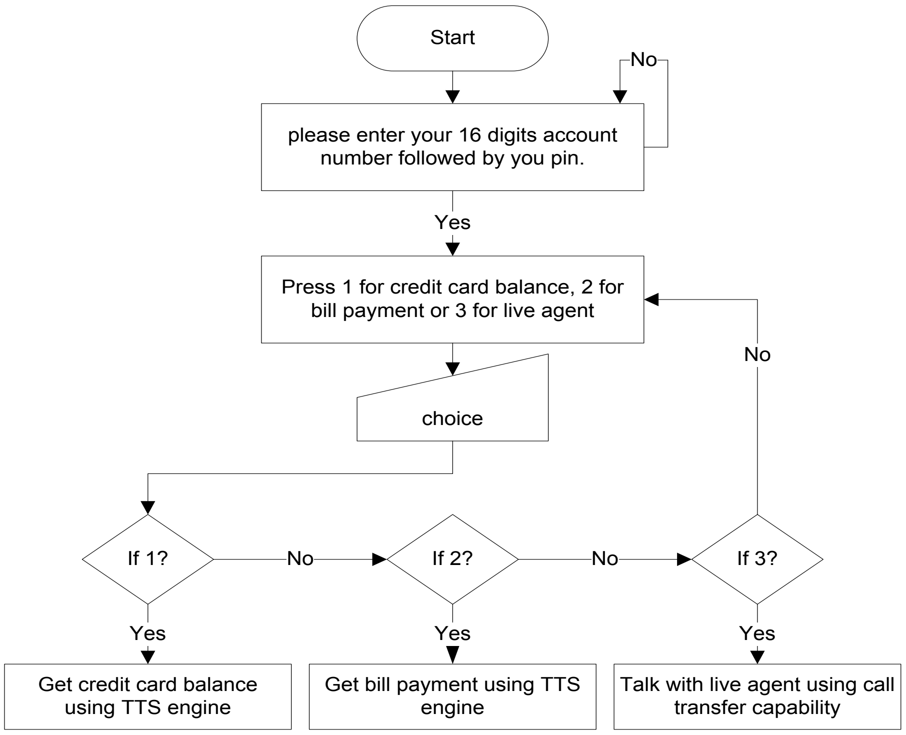

Start
No
please enter your 16 digits account number followed by you pin.
Yes
Press 1 for credit card balance, 2 for bill payment or 3 for live agent
No
choice
If 1?
No
If 2?
No
If 3?
Yes
Yes
Yes
Get credit card balance using TTS engine
Get bill payment using TTS engine
Talk with live agent using call transfer capability

Figure 2.1: Bank IVR Flow Chart

In this IVR application, the installed system asks the caller to enter the 16 digits account numbers, followed by the pin. Afterwards, the system consult the database to check if the account number correspond to the pin entered, if yes, it ask the customer to press 1 for credit card balance, 2 for bill payment or 3 for a live agent, and if no, the system replay the same prompt. Then, according to customer's choice, the system either read the balance or the payment amount from the database using the TTS module or route the call to a live agent using the call transfer feature. Therefore, this IVR application integrates play prompt, detect a key, text to speech, call transfer and database interaction features to deliver to customers this composed IVR service.

Automated survey also makes use of IVR systems. It is used to gather data, where the system make an outgoing call, then a recorded messages ask questions and requests some answers, like "yes," or "no" using the speech recognition module or via the phone's keypad by pressing 1 for yes, and 2 for no [29]. Moreover, call center [31, 29] make use of IVR system for call routing and call forwarding. Also, IVR is used for selective information lookup such as bus schedules, movie schedules and many more. Calling card is another app, requiring the use of IVR features. Therefore, it is noticeable that IVR applications are not restricted for special uses, but on the contrary, they are everywhere and for everyone.

2.1.5 Related challenges

IVR applications require some level of media processing. End terminals may execute media processing. However, centralized media processing server will offer more efficiency and more benefits such as scalability and redundancy. Media Server (MS) entity provides media processing for end users by mixing stream, converting media of one encoding to another, playing prompts, collecting digits, and so on.

Although the RTP [36] streaming flows directly from MS to the end user, the signaling flow such as SIP [37] does not, and an Application Server (AS) will act as a middle entity to isolate one from the other from the signaling point of view.

An AS is an entity container for handling and executing services in network. AS control MS using a control protocol (MSCML [38], VXML [39]) embedded in the body of the signaling request. For example, the AS may request the MS to play a specific file to an end user, where the name of the file is embedded in the body of the request. Here is a figure that illustrate the idea.

Furthermore, an IVR platform is required which is the hardware of the servers, operating systems and software on which IVR systems run. Also, in reality, a telephony infrastructure also is included which covers the telephone lines and the call switching equipment to process calls. Moreover, Database servers may exist on which the required customer information can be found. Figure 2.2 shows the interaction between the application server and the media server along with the media and the signaling flow between them.

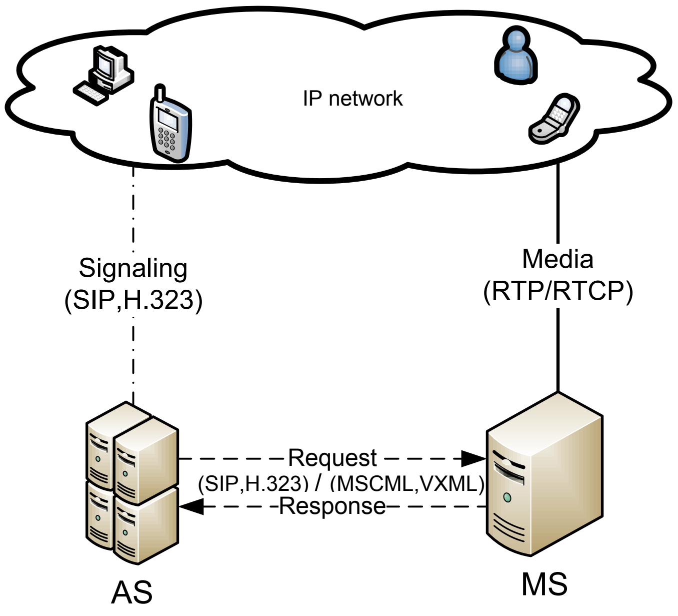


Figure 2.2: Application Server and Media Server interaction

2.2 Cloud Computing

This section pays attention to the Cloud Computing paradigm. First of all, we present an overview of the Cloud Computing concept. Then, we present the architecture of cloud computing, in which we discuss the service models, the business model, and the types of Cloud. Lastly, we present the characteristics of this paradigm.

2.2.1 Overview

Cloud computing is a newly emerging paradigm that host and provide services over the internet with many inherent benefits [2]. However, so far Cloud does not have a coherent and agreed definition [1]. In other words, there is not yet a standard technical definition for cloud computing. Nevertheless, a consensus is emerging around the most critical facets it encompasses: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1].

In addition, nobody denies that Clouds are a large interconnected pool of virtualized resources that are sharable and accessible over a network, usually the Internet [1, 16]. These resources could be data, services, development platform, and/or hardware [1]. With Cloud Computing, resources are provided on demand dynamically, and on a pay per use basis [24, 25]. These resources are placed in the cloud for many reasons such as scalability, efficiency and costs [19, 1].

2.2.2 Cloud Computing Architecture

This section describes the service models of Cloud Computing paradigm, IaaS, PaaS, and SaaS. It explains the business model of this paradigm, and concludes by listing the different types of the existing clouds.

2.2.2.1 Service models

The architecture of a Cloud Computing paradigm can be divided into 3 layers: the infrastructure layer - IaaS, the platform layer - PaaS, and the software layer - SaaS. However, IaaS is composed of the actual computing hardware (resources) and the infrastructure, PaaS of the platforms, and SaaS of the applications as illustrated in Figure 2.3.

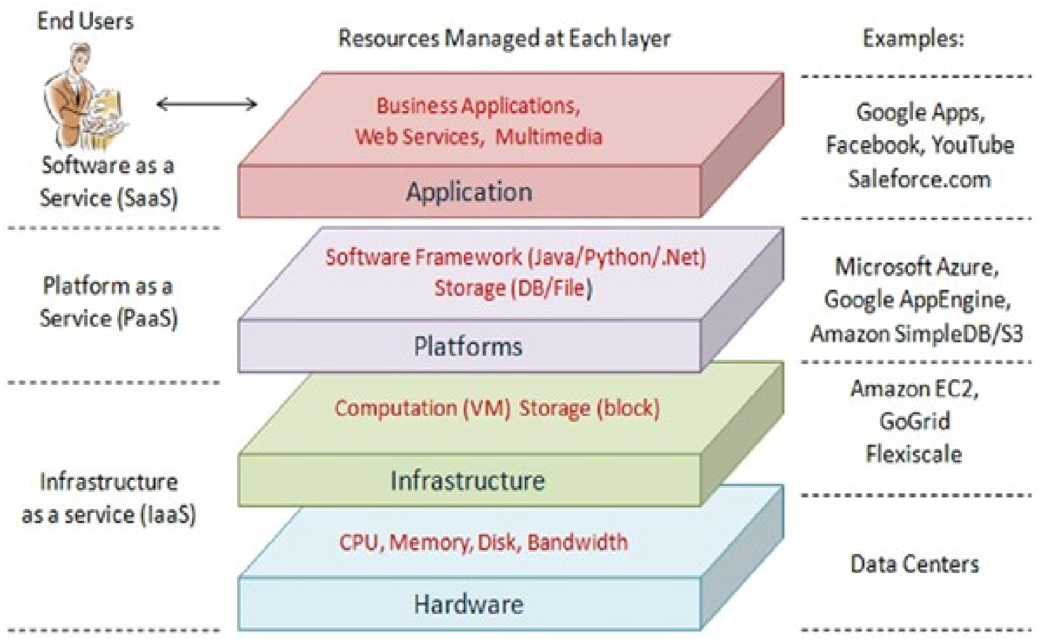


Figure 2.3: Cloud Computing Architecture

- **Infrastructure as a Service**

  The Infrastructure as a Service (IaaS) layer consists of two sub-layers: the hardware layer and the infrastructure layer.

- **Hardware layer**

  The responsibilities of this layer covers the managing of all the hardware resources of the cloud such as CPU, memory, and including all the physical servers and the network equipments such as routers, switches. The hardware layer is implemented in data centers in which thousands of servers are interconnected together with switches, routers or other fabrics [2].

- **Infrastructure layer**

  This layer is responsible for creating a pool of storage and virtualized computing instances by partitioning the hardware resources using one of the various virtualization technologies such as Xen [20] and VMware [21]. Besides, it is responsible for the dynamic resource allocation which is made through virtualization technologies [1, 2].

- **Platform as a Service**

  PaaS is the middle layer in the cloud paradigm. It enables developers to build application on the fly. It is composed of operating systems and application frameworks [2]. The purpose of the platform layer is to provide services for developers [1]. it add some additional level of abstraction to the virtualized resources offered by the infrastructure provider instead of supplying a virtualized infrastructure [1, 2].

- **Software as a Service**

  The SaaS is the top layer in the paradigm. It consists of the actual cloud applications and provides them to the end users [1] through standardized interfaces [2]. Based on the underlying resources, Cloud applications could provide many features comparing to the traditional applications such as scaling, availability, and cost [2].

2.2.2.2 Business model

Cloud computing rely on a service-driven business model. Specifically, physical hardware and platform are offered as services on-demand to consumers. Besides, each layer of Cloud computing architecture could provide a service to the layer above [2] as shown in Figure 2.4.

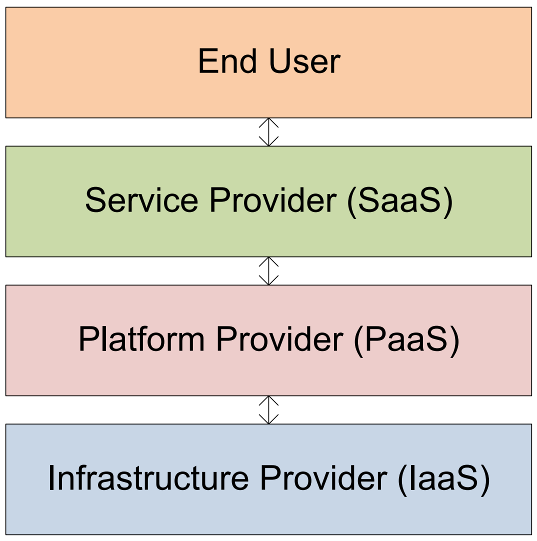


Figure 2.4: Cloud Business model

- **Infrastructure as a Service**

  IaaS provider is responsible for providing on-demand infrastructural resources such as CPU and memory, usually in terms of VMs to the platform provider [2]. Amazon EC2 [22] and GoGrid [26] are two of the well-known IaaS providers.

- **Platform as a Service**

  PaaS provider is responsible for providing platform layer resources, including operating system and software development to the service provider [2]. Google App Engine [23] and Salesforce [28] are two of the well-known Platform provider that offers APIs to develop and run applications in the cloud.

- **Software as a Service**

  SaaS provider is responsible for providing applications over the Internet to end users [2]. Examples of SaaS providers include Rackspace [27].

2.2.2.3 Types of Cloud

There are various issues should be taken into account when moving an application into the cloud such as reliability and security [2]. Consequently, cloud computing can be classifies as public cloud, private cloud, hybrid cloud [15], and virtual private cloud [2]. Each type has its own characteristics that differentiate it from the other types.

- **Public clouds**

  It is a cloud in which service providers provide their resources to the external parties [15]. However, this type lacks some control over data, network and security settings [2].

- **Private clouds**

  In contrast, this type is devoted to use by a single organization [16]. It may be developed and organized by the organization itself or by external providers. This type provides performance, reliability and security [2].

- **Hybird clouds**

  A hybrid cloud shares resources between public cloud and private cloud to address the problem of each approach [15, 2]. In other words, the service infrastructure of a hybrid cloud runs in private and public cloud simultaneously. The main benefit of hybrid clouds is that it provides more flexibility than both public and private clouds [2].

- **Virtual Private Cloud**

  It is another solution to address the limitations provided by both public and a private cloud. It is actually a platform that runs on top of public clouds. However, the only difference is that it allows service providers to create their own topology and security level such as firewall [2].

2.2.3 Characteristics

Cloud computing provides many characteristics that differentiate it from traditional paradigms hosting, which are summarized below [2]:

- **Shared resource pooling**

  The infrastructure provider offers a pool of shared computing resources that can be dynamically allocated to many resource consumers [2]. This feature provides an unlimited storage capacity [15] for end user, as well as reduces the costs associated with the maintenance of hardware and software resources [19, 1]. In addition, this feature allows providers to get a high level of server consolidation where resources are maximally utilized and reduce cost such as power consumption and cooling [2].

- **Utility-based pricing**

  Cloud computing employs a pricing model that lets you pay as you go [15]. Consequently, consumers are charged per-use basis for the rental period of resources [2].

- **Elasticity**

  Clouds provide on-demand resources. Therefore, Clouds should be elastic in terms of capacity. In other words, computations that need more resources, simply request them from the cloud [15].

- **Dynamic resource provisioning**

  This feature provides efficiency in resource management [15]. With dynamic resource provisioning, resources are obtained on-demand and released also on the fly [2] in which there is no wastage of resources.

- **Ubiquitous network access**

  Clouds resources are accessible through the Internet. Therefore, any device such as laptop, ipad, and mobile connected to the internet, will be able to access resources in the clouds [2]. Hence this capability provides mobility and maximum service utility of the resources which increase performance [2].

2.3 Virtualizations

The role of virtualization in Clouds is a key component [1, 24]. In this section, we first give a brief overview of the virtualization technology, and then we discuss the role of Virtual Machine Monitor or Hypervisor in the virtualization environment. Lastly, we mention the benefits of the virtualization technology.

2.3.1 Overview

Virtualization is the foundation technology of Clouds, as it is the basis for characteristics such as, dynamic assignment and on-demand sharing of resources [1, 2]. Virtualization provides virtual resources [2] instead of providing the actual ones. In other words, virtualization enables the partitioning of the resources using virtualization technologies such as Xen [20] and KVM [30] [2]. These resources could be operating systems coexisting on the same hardware server, applications coexisting on the same operating systems, or even networks coexisting on the same routes.

2.3.2 Virtual Machine Manager

In a virtualized environment, a piece of software called Virtual Machine Monitor (VMM) or Hypervisor is added to the underlying compute resources to provide virtualization [14]. This layer will improve the sharing and utilizing of the underlying resources in the virtualized system.

In a virtualized environment, VMM takes complete control of virtualized resources [3]. With virtualization, it is possible to create multiple virtual machines and run them on a single physical hardware to maximize utilization. Also, it enables the possibility to use multiple instances of a single application and run them simultaneously on a single physical server to improve application performance.

Figure 2.5 show the difference between a non-virtualized environments and a virtualized one. In the first figure a single operating system owns all hardware resources while in the second we notice that the hypervisor make it possible to run multiple instance of operating systems the same physical server.

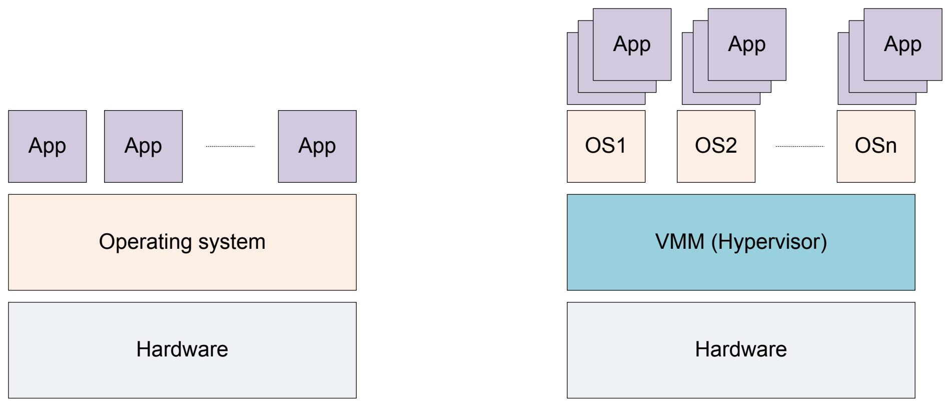


Figure 2.5: Difference between environments

2.3.3 Benefits

Virtualization provides has various benefits. The major reason that consumers would want to use virtualization technology is that it provides the maximum benefit out of a server, which will reduce physical infrastructure costs and increase the space utilization efficiency by the dynamic load balancing process [2]. This will help also in reducing maintenance cost, power consumption and cooling requirements [2].

Another advantage of virtualization is that it gives consumers the choice to deploy a variety of operating systems technologies at the same time on a single hardware platform [14]. Also, it provide more security in isolating the physical server into multiple independent guest virtual machines as they don't interfere with each other which guarantee availability of a computing system even when a guest OS fails [3].

2.4 RESTful Web services

RESTFul Web service is a key technology of our architecture. Accordingly, in this section we first present an overview on RESTFul Web services, and then we discuss the methods and the Web Application Description Language (WADL).

2.4.1 Overview

RESTful Web services follow the Representational State Transfer (REST) design paradigm. REST uses the Web's basic technologies (e.g. HTML, XML, HTTP, and URIs) as a platform to build and provision distributed services. It is one of the players of Web 2.0, a concept that promotes interactive information sharing and collaboration over the Web, as well as Web application consumption by software programs. REST adopts the client-server architecture. However, it does not restrict client-server communication to a particular protocol, but more work has been done on using REST with HTTP, as HTTP is the primary transfer protocol of the Web [17].

RESTful Web services can be described using the Web Application Description Language (WADL [18]). A WADL file describes the requests that can legitimately be addressed to a service, including the service's URI and the data the service expects and serves [17].

REST supports a wide range of representation formats, including plain text, HTML, XML and JavaScript Object Notation (JSON). RESTful Web services are perceived to be simple and easy for clients to use because REST leverages existing well-known Web standards (e.g. HTTP, XML) and the necessary infrastructure has already become pervasive.

RESTful Web services' clients (i.e. HTTP clients) are simple and HTTP clients and servers are available for all major programming languages and operating system/hardware platforms [17].

2.4.2 REST methods

REST uses the HTTP methods [17]:

- **POST** to create a new resource on the server, where the server create the URI of this resource
- **PUT** to modify a resource on the server
- **GET** to retrieve a representation of a resource
- **DELETE** to delete an existing resource

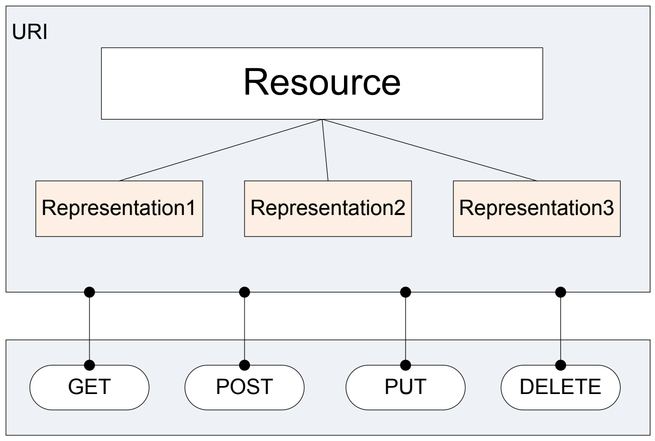


Figure 2.6: REST Web services design structure

Figure 2.6 shows the REST web service diagram. Each resources is represented by one or more representation and identified by a unique URI. REST use HTTP methods GET, POST, PUT, and DELETE to interact with the service. Client initiaite a request to servers, servers process the request and return appropriate responses. Client may use GET for discovery and POST for publishing.

2.4.3 Web Application Description Language (WADL)

WADL is an XML-based data format that provides a machine process-able description of HTTP-based Web applications [18]. These applications are specifically RESTFul web services.

Applications are described in a machine process-able format using [18]:

- Set of resources provided by the application.
- Relationship between resources that links between them.
- Methods that can be applied to each resource.
- Resource representation formats.

The <application> element is the root of a WADL description and may has <doc>, and <resources> as child elements. All the WADL elements have "http://wadl.dev.java.net/2009.02" as XML namespace name [17].

<doc> child elements is used to document the base element. It has xml:lang and title as attributes. xml:lang defines the language for the title attribute, whereas title describe the elements being documented [18].

<resources> is the container for the all the resources provided by the application. It has a base attribute which is the base URI for each child resource identifier. It has <resource> as a child element, which describe the resources provided by the application [18].

<resource> element describes a set of resources where each one is identified by URI. Each resource may have some optional attributes such as id, path, and type. Also, it may have some child elements such as <doc>, <param>, and <method> [18].

<method> element describes the input to and output from an HTTP method that may be applied to a resource. It has a name attribute that specify the HTTP method used such as GET or PUT. Also it may has zero or more <request> and <response> child elements [18].

<request> element specifies how to represent the input to be included when executing an HTTP method to a resource. It has no attributes. However, it may have <doc>, <representation>, and <param> as an optional child elements [18].

<response> describes the output from executing an HTTP method on a resource. It has status as an optional attribute, and the same child elements of the <request> element [18].

<representation> decribes a representation of resource's state. It has id, mediaType, element, and profile as attributes [18].

<param> describes a parameterized component of a parent element. It has id, name, style, type, path, fixed, and repeating as attributes [18].

Figure 2.7 shows a simple WADL file. A resource identified by a static URI: http://localhost:8080 /msg. On executing a get method on this URI, the reply will be a response of type text/plain.

```
<?xml version="1.0"?>
<application xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd">
    <resources base="http://localhost:8080">
        <resource path="/msg">
            <method name="GET" id="getMessage">
                <response>
                    <representation mediaType="text/plain">
                </response>
            </method>
        </resource>
    </resources>
</application>
```

Figure 2.7: WADL file sample

2.5 Chapter summary

This chapter has presented an introduction for all the concepts and keywords that are going to pass through this thesis. We first gave an overview about IVR and mentioned its features, benefits, some of its application and the challenges for developing IVR application.

Afterwards, we have discussed the cloud computing paradigm, its architecture, business model and characteristics. Also we have provided an overview of virtualization technology which is the foundation of cloud computing paradigm and lastly we have presented a brief overview about RESTFul Web service technology. In the next chapter, we are going to derive the set of requirements on having a virtualized infrastructure for IVR applications and review the most relevant state of the art.

Chapter 3

A Virtualized Infrastructure for IVR Applications as Services: Requirements and State of the art Evaluation

Existing IVR applications such as automated attendant or bank tellers can be offered in cloud-based settings to provide various benefits. In this thesis, we are looking forward to provide an IVR application as a service with a virtualized infrastructure. However, in order to develop this integrated structure, a set of challenges and requirements should be imposed and taken into account.

Accordingly, this chapter is composed of three sections. We first derive a set of requirements for providing an IVR application as a service with a virtualized infrastructure. Afterwards, we discuss some of the most existing related works in our research area and evaluate these related works based on our requirements. Finally, a summary is presented to reviews the main things mentioned in this chapter.

3.1 Requirements for a Virtualized Infrastructure for IVR Applications as Services

This section is divided into two set of requirements for having a virtualized infrastructure for IVR applications as service. We first present the general derived requirements for the overall system. Then, in the second part, we discuss some specific requirements for having a virtualized IaaS.

3.1.1 General Requirements

As it was discussed in Chapter 2, Cloud computing is a paradigm with three service models: SaaS, PaaS, and IaaS. In addition, virtualization is a technology that enables the coexistence of entities on the same hardware, network or virtual machine and allows resources to be allocated dynamically on the fly. However, an IVR application offered in cloud-based settings and with a virtualized infrastructure must meet challenging requirements.

Therefore, our first requirement in our proposed architecture is that the broker should enable publication and discovery of IVR substrates and IVR substrate instances. In addition, it should enable each player (e.g. service provider), to discover the other players (e.g. platform providers).

The second requirement we have in mind is that the platform provider on the top of the infrastructure layer should be able to add some level of abstractions such as GUIs and APIs to the substrates available in the infrastructure layer. Consequently, the service provider in the SaaS layer should be able to compose the substrates available in the infrastructure into powerful IVR applications, using appropriate platforms. The service provider should also be able to manage the applications using the same platforms.

Let us suppose we have different IVR applications in different domains and composed by different service provider. The last requirement is that the applications should be able to share the same instances of substrates available in the infrastructure layer. Conversely, an IVR application should also be able to use many instances of a same substrate, for scalability.

3.1.2 Virtualized IaaS requirements

In addition to the general requirements mentioned in the first part, there are some other specific requirements related to the virtualized IaaS layer.

First of all, with Cloud infrastructure, it should be possible to instantiate the composed applications and its related substrate. However, the virtual substrates available in IaaS could belong to the IaaS provider or third parties. This will enable the availability of a wide range of virtual substrates.

As the next requirement, Infrastructure providers should offer the virtual substrates as they are offered by substrates provider. Also it should be possible to offer the composed services to end users or other applications.

Furthermore, with virtual infrastructure, the execution of composed virtual substrates should be coordinated by the infrastructure provider and be transparent to the third parties which would offer the substrates and are part of the composed service.

General requirements	1. Ability to publish and discover substrates and substrate instances
	2. Platform provider should add some level of abstractions to the substrates available in the infrastructure
	3. Service provider in the SaaS layer should be able to compose the substrates
	4. Different IVR applications in different domains should be able to share the same substrates
	5. IVR application should also be able to use many instances of a same substrate
Virtualized IaaS requirements	1. Applications and substrates instantiation
	2. Offering of substrate and newly composed applications
	3. Transparent execution to third parties

Table 3.1: Derived Requirements

Table 3.1 summarizes our general requirements as well as the IaaS specific requirements discussed above.

3.2 Related Work and Evaluation on the Virtualized Infrastructure for IVR Application as Services

Several commercial hosted IVRs such as call centres allow users to remotely access IVR applications as services. However, to the best of our knowledge these applications do not rely on virtualized infrastructures. Embryonic architectures have been proposed that could offer a few applications, which may include selected IVR features, as services in clouds. One example is conferencing that can include announcement player and digit collector.

In this section we introduce some related works in our research interest, a virtualized infrastructure for IVR applications as services. Afterwards, we evaluate these related works based on our requirements.

3.2.1 Related Work

In this subsection, we organize the related works into two approaches: the cloud overall approaches and IaaS approaches. The cloud overall approach includes the general existence works related to our research interest while the infrastructure (IaaS) approaches includes some specific IaaS related works that focuses more on the infrastructure architecture.

3.2.1.1 Cloud overall approaches

P. Rodriguez et al. [6] propose a video conferencing application as SaaS, along with the required supporting virtualized infrastructure. The goal is to ease the integration of video conferencing with other applications such as facebook or igoogle. Therefore, according to authors, users will be able to collaborate among themselves easily with audio, video, share application, IM, etc. The proposal has some requirements, the main ones are: at first, the management of users is the responsibility of the third parties ‘applications and the second one is that the proposal should focus on conferences room only. The following figure will illustrates the idea of the proposal.

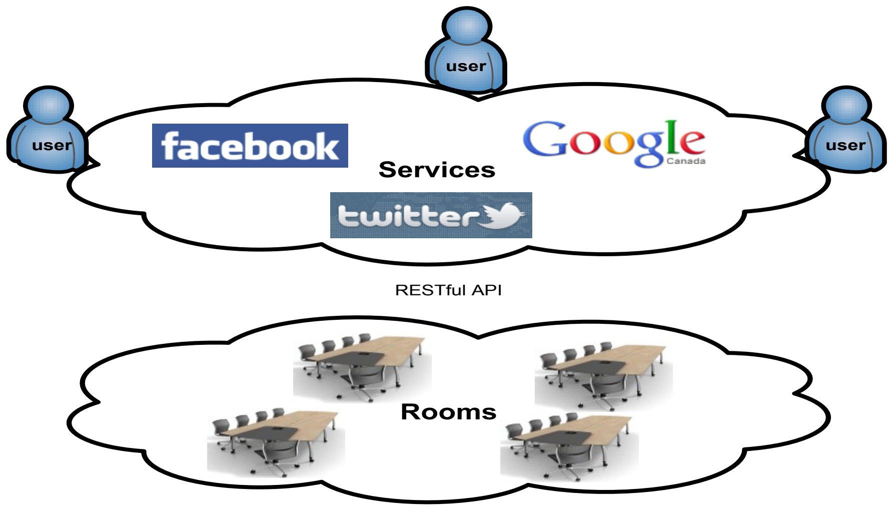


Figure 3.1: Design of a Videoconference as a Service

Figure 3.1 models the architecture of a videoconference room provider where it shows three resources, services, rooms, and users. The services are the third party’s applications that want to use the rooms to give access to users in order to communicate in each of these rooms, Facebook as an example. Rooms are the main resources and the virtual instances where the users will collaborate among themselves with video and audio. Users represent the third parties' users who will communicate with others in each room. Also REST web services API are used to GET, POST, PUT and DELETE resources between the third parties application and the infrastructure layer.

However, the infrastructure described above is limited to one virtual full-blown video conferencing server. The key drawback of the proposal is that it follows a very coarse-grained approach. It does not provide an infrastructure with substrates that can be published, discovered and dynamically shared as we envision in the architecture proposed in this thesis. Furthermore it doesn't offer service providers the ability to compose a service. Finally, it does not include any IVR feature. It focuses only on the videoconference room application.

The architecture defined in [7] is another example that also discusses audio/video conferencing as SaaS. It was mentioned in [7]: "In our viewpoint, multimedia conferencing can be also seen as a service called conference service. It is surely a software application, which follows the concept of SaaS. In this way, we would not worry about the size of conference, since the cloud's wonderful characteristics makes this problem easy to resolve."

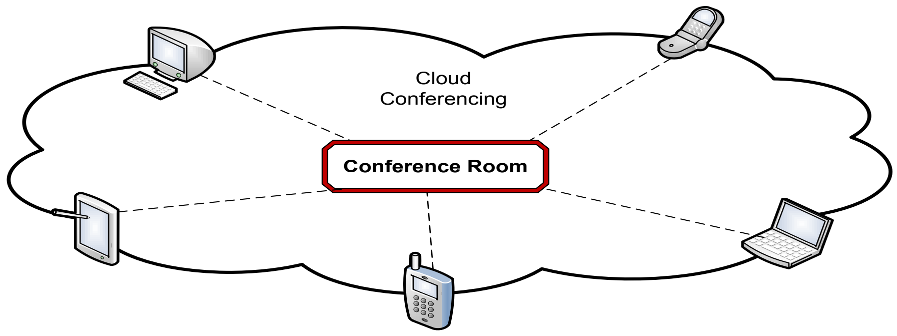


Figure 3.2: Cloud Conference Room

Accordingly, they propose the viability of cloud conferencing, and design a practical framework. The centralized conferencing room shown in Figure 3.2 adopts a tightly coupled structure, so when the number of participant increases, the conference room enlarges its size by get resource from resource pool in cloud. When participants leave, the resource would be returned to the pool.

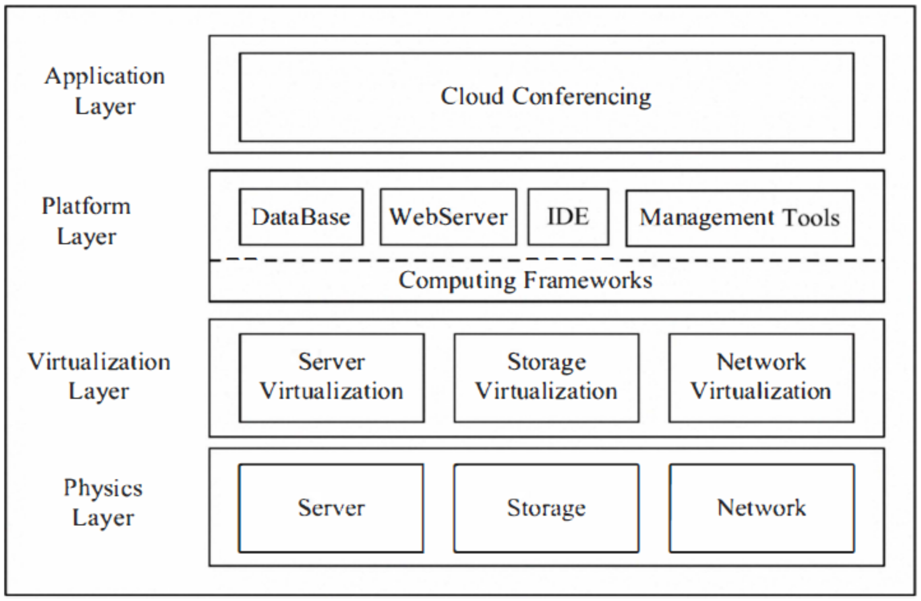


Figure 3.3: Software Structure

Figure 3.3 shows the software architecture of the proposal. In this architecture, the infrastructure virtualizes the server, storage, and the network.

However, this proposal also shares the drawbacks of the previous proposal in terms of the infrastructure. In addition, no fine-grain substrate is proposed and the entire conferencing server is virtualized. In other words, substrate providers don’t exist in the architecture, and service providers are not able to develop their own applications.

3.2.1.2 IaaS approaches

Substrates have been proposed for virtualized infrastructures that support applications with no bearing on IVR. Another related work is the infrastructure proposed for presence service in [8]. “Presence service is used for the discovery, the retrieval of, and the subscription to changes in end-users’ context information (e.g. on-line/off-line status, willingness to communicate, locations).” That proposed architecture is composed of three layers (a virtual presence service layer, a presence service virtualization layer, and a presence substrate layer) and two planes (presence service and virtualization control and management). It also includes a business model. However, the work is still at a preliminary stage and thus far only one substrate has been identified for the application. In addition, publication, discovery and composition have therefore not yet been addressed. Accordingly, this architecture doesn’t support the composition and the instantiation of new service.

Furthermore, several virtualized infrastructures have been proposed for Future Internet [9, 10, 11]. These rely on virtualized nodes and links as substrates. However, they focus on the core infrastructure of the Internet and do not address issues related to the virtualization of the edge infrastructure of the Internet, where applications such as those of IVR reside. Also, these references don’t support applications composition and instantiation.

For example, in [11] authors discuss the network virtualization for future internet architecture where new component are added to enable the coexistence of multiple networking approach. In this article, they are evaluating the influence of virtualization on the underlying network infrastructure in which it will allows the deployment of various new architectures and protocols over this shared physical infrastructure.

Besides, a few architectures do deal with some of the issues related to application development by composition in a cloud environment. However, none of them addresses the specific case of IVR applications. Reference [12] provides an example. It targets applications at large and proposes a virtualized mash up container. Application substrates, called service components in the proposal, are described using existing technologies such as REST, JSON, and RSSFeed. A service proxy maps these descriptions into the internal description technology supported by the container. Unfortunately, little detail is provided on this internal format. Furthermore, the specific publication and discovery mechanisms are not discussed. Also the instantiation of the composed service is not supported in the mentioned reference.

In [13] they provide another example. It proposes a virtualized infrastructure where music content providers federate with value-added service providers (e.g. security, audit) to build virtual music stores on the fly. Several registries (e.g. client, capability registry) are mentioned in the paper. However, no detailed information is provided about how the content of these registries is described, published and discovered.

3.2.2 Evaluation summary

After presenting the most relevant related works in our research domain, we can observe that none of them fully satisfy our requirements.

Reference [6] does not provide an infrastructure with substrates that can be published, discovered and dynamically shared. Furthermore it does not include any IVR feature. It focuses only on the videoconference room application.

Reference [7] shares the drawbacks of the previous proposal in terms of the infrastructure. In addition, no fine-grain substrate is proposed and the entire conferencing server is virtualized.

Reference [8] focuses on the presence service. Also, publication, discovery and composition have therefore not yet been addressed.

References [9, 10, 11] rely on virtualized nodes and links as substrates. However, they focus on the core infrastructure of the Internet and do not address issues related to the virtualization of the edge infrastructure of the Internet, where applications such as those of IVR reside. Also, these references don’t support applications composition and instantiation.

References [12, 13] deal with the composition process. However, it does not include IVR features and no detailed information is provided about how the content of these registries is described, published and discovered. Besides, it doesn’t support the instantiation of the composed service.

Finally, we can conclude that there is no full-fledged cloud environment that enables the development, management and offering of the full range of IVR applications.

3.3 Chapter summary

In this chapter, we have first derived a set of requirements consists of a set of general requirements on which we built our architecture and some other virtualized IaaS requirements in which we mentioned some of the expectations we are looking for in this architecture. Also, in the second section we have presented some of the relevant works in having a virtualized infrastructure for IVR applications as services, and then evaluated these related works based on our requirements. In the next chapter, we will present our proposed architecture based on the requirements presented in this chapter.

Chapter 4

Proposed Architecture[1]

In the previous chapter, we have derived the appropriate requirements for having a virtualized infrastructure for IVR applications as services. Accordingly, the purpose of this chapter is to propose a suitable architecture based on these derived requirements.

This chapter is organized into three sections. It starts with a general overview of the proposal. Then, in the second section, it presents the proposed business model. Lastly, it introduces the proposed architecture along with its entities, planes functionality, interfaces, and operational procedures.

[1] This chapter is an extended version of the paper F. Belqasmi, C. Azar, M. Soualhia, N. Kara, and R.Glitho, "A Virtualized Infrastructure for IVR Applications as Services", accepted for presentation at the ITU Kaleidoscope conference, Cape Town, December 2011.

4.1 General Overview

In this thesis, we propose a novel architecture for a virtualized IVR infrastructure as a first step towards the development and management of full-fledged IVR application in cloud settings.

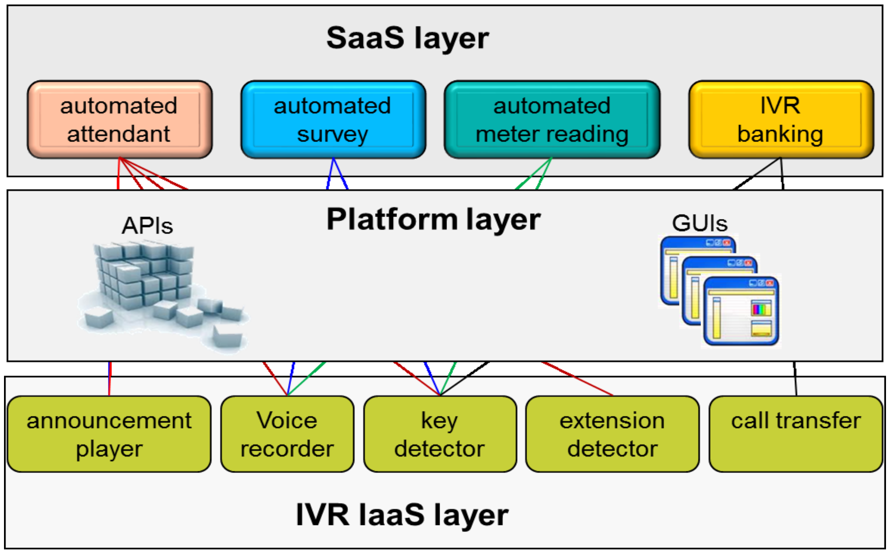


Figure 4.1: Different services share the same substrates

Figure 4.1 depicts our vision.

- The bottom layer shows a simplified IVR IaaS layer with the following virtualized IVR substrates: announcement player, voice recorder, key detector, extension detector, and call transfer.

- At the top layer, we have a simplified SaaS layer with applications such as automated attendant, automated meter reader, automated survey, and IVR banking that share the IVR substrates available in the infrastructure layer.

- The middle layer is the platform layer. PaaS adds some level of abstractions to the substrates available in the virtualized infrastructure using graphical user interfaces (GUIs) and application programming interfaces (APIs) that may ease the development and management of the applications in the top layer.

Consequently, different applications in the SaaS layer will be able to share the same substrates and also will be able to use several instances of the same substrates in the IaaS layer to develop applications.

For example, an IVR banking service can be developed by composing the announcement player, key detector, extension detector and call transfer. In addition, an automated survey service can be developed by composing the announcement player and the key detector while an automated attendant service can be developed by composing the announcement player, voice recorder, extension detector and call transfer.

Accordingly, the three developed services will be able to share the announcement player substrate. IVR banking and automated attendant will be able to share the extension detector and call transfer substrates. Finally the IVR banking and automated survey will be able to share the key detector substrate. Also, the three services are able to use several instances of the same substrates.

4.2 Business model

According to the requirements derived in the previous chapter, we propose a business model that will symbolizes our viewpoint. The business model is composed of an IVR substrates provider, an IVR infrastructure provider, an IVR platform provider, an IVR service provider, a Broker, and a Connectivity provider. Figure 4.2 shows the proposed business model.

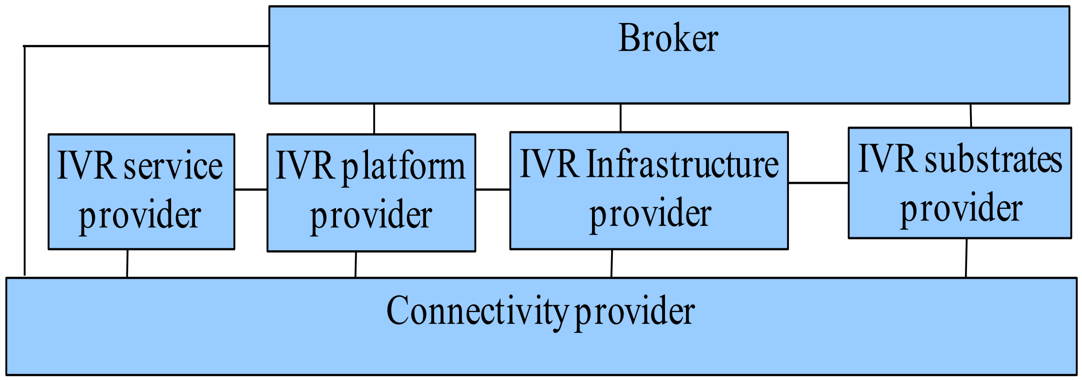


Figure 4.2: Proposed business model

- **IVR Substrates provider**

  IVR substrates providers are responsible for offering their IVR substrates such as announcement player and voice recorder to IVR infrastructure provider. However, a given IVR substrate provider may interact with several IVR infrastructure providers and offer them the same IVR substrates, since these substrates are sharable.

- **IVR Infrastructure provider**

  IVR infrastructure provider is responsible for providing IVR substrates as a service to the platform provider. However, a given IVR infrastructure provider may also interact with several IVR substrate providers.

- **IVR Platform provider**

  IVR platform provider adds some levels of abstractions to IVR substrates available in the infrastructures to facilitate IVR application development and management by IVR service providers. Also, IVR platform is provided as a service, for one or several IVR service provider.

- **IVR Service provider**

  IVR service provider develops and manages these applications using the IVR platform offered by IVR platform providers. It also offers IVR applications as SaaS, accessible by end-users and other applications.

- **Broker**

  The broker acts as a substrates storage repository. It enables the platform, infrastructure, and substrate providers to publish and discover substrates.

- **Connectivity provider**

  Connectivity provider enables connectivity between the different actors in the proposed business model.

According to this business model, different service providers may use the Platform provider's GUI to develop services such as automated attendant and IVR banking by composing the substrates provided by IVR infrastructure provider to the IVR platform provider.

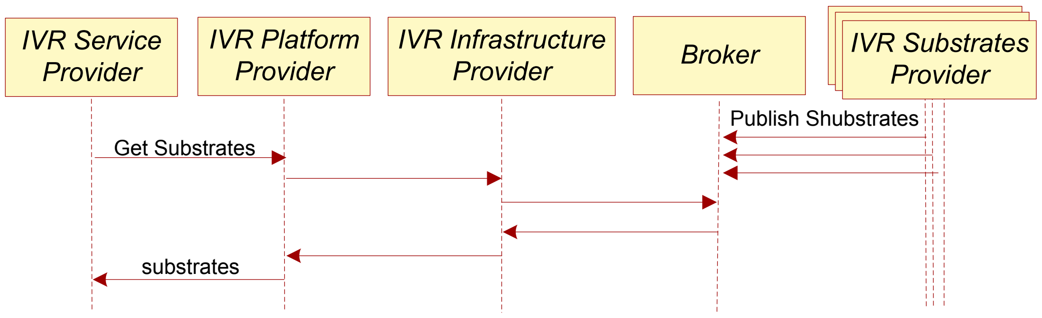


Figure 4.3: Business model Sequence diagram

Figure 4.3 illustrates the business model sequence diagram, where it shows how different IVR substrates providers publish their IVR substrates first such as announcement player, voice recorder, key detector, extension detector, and call transfer.

When an IVR service provider wants to develop an IVR service such as automated attendant or IVR banking, it sends a discovery request to the IVR platform provider to get all the available IVR substrates. When The IVR platform provider receives the request, it forwards it to the IVR infrastructure provider that sends it to the broker to get all the substrates. Lastly, the IVR infrastructure provider gets the substrates from the broker and sends them to the IVR platform provider which displays them in the appropriate GUI to the service provider to compose its own IVR service.

4.3 Architecture

In this section, we will first present our proposed architecture and then in the following subsection we are going to describe the entities, the interfaces, the planes functionalities, and the operational procedures.

4.3.1 Overview

The proposed architecture comprises two layers, three planes architecture and a repository. The first layer contains the functional entities that realize the infrastructure provider role. The second layer is comprised of entities that realize the IVR substrate provider role. The interactions between the two layers are organized via three planes: service, management and composition. The repository realizes the role of the broker. Figure 4.4 shows the proposed architecture.

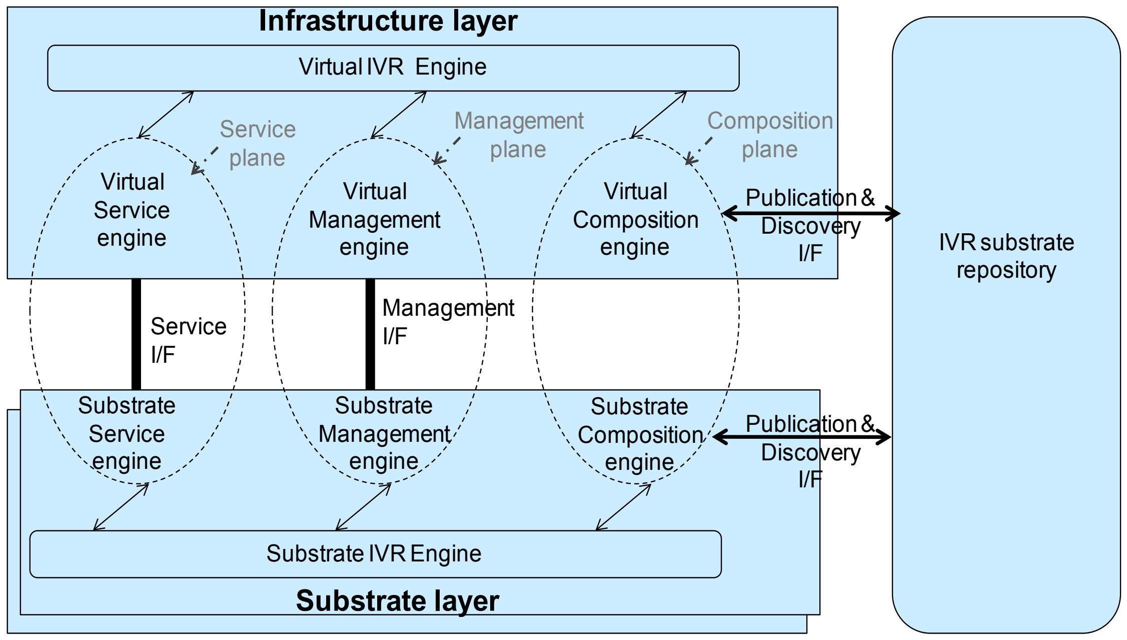


Figure 4.4: Overall architecture

4.3.2 Entities

In our architecture we have two key entities. The key functional entity of the infrastructure layer is the virtual IVR engine. Besides, the key entity of the substrate layer is the substrate IVR engine.

- **Virtual IVR Engine**

  The virtual IVR engine coordinates the activities of the virtual service engine, the virtual management engine and the virtual composition engine. The virtual IVR engine interacts with several substrate IVR engines--or more precisely, it interacts with the engines of all the substrates that make up a given composed service.

- **Substrate IVR Engine**

  The substrate IVR engine coordinates the activities of the substrate service engine, the substrate management engine, and the substrate composition engine.

4.3.3 Planes functionality

Three planes exist in our overall architecture: composition plane, management plane, and service plane.

- **Composition plane**

  The composition plane creates composed virtual IVR service by composing different Substrate IVR Instances (SIIs) from the same or different IVR Service Substrate (ISS) providers. Besides, it manages the execution of composed virtual IVR services. It also interacts with the repository and enables the publication and discovery of the substrates and substrate instances that are used in composition.

- **Management plane**

  The management plane handles the actual control and management of substrate resources. It enables the instantiation of IVR applications and related substrates, and their configuration. For example, it creates and manages ISIs, reserves and manages resources. It also enables fault monitoring, performance monitoring, and the accounting for charging purposes.

- **Service plane**

  The service plan handles the IVR service provisioning tasks. It is responsible for the execution of the IVR services. These services could be composed of one or more IVR substrate. In addition, the IVR substrates could be provided by one or more substrate provider. However, the main functionality handled in the service plane is mediation.

4.3.4 Interfaces

Three types of interfaces exist in our architecture: service I/F, management I/F, and publication & discovery I/F.

- **Service I/F**

  The virtual service engine and the substrate service engine communicate via the interface supported by the substrate service engine. This is motivated by the fact that existing potential IVR substrates come with a plurality of interfaces (e.g. VoiceXML, Session Inition Protocol-SIP, Media Server Control Markup Language-MSCML). They communicate with the virtual service engine via mediators that are incorporated in the virtual service engine. However, for each I/F type, a new plugging can be added or implemented by the ISS client.

- **Management I/F**

  A key requirement for the management interface is to accommodate a plurality of resource description mechanisms (e.g. XML, plain text). Another requirement is that the interface should be supported by commonly used virtualization servers such as XEN to ease the creation of instances. These requirements have led to our selection of RESTful Web services as the natural choice. Therefore, REST-based I/F are used at both layers, the virtualization and the substrate layer.

- **Publication & Discovery I/F**

  We have also decided to use RESTful Web services for the publication/discovery interfaces to minimize the number of interfaces in our proposed architecture. The fact that RESTful services support a wide range of resource description mechanisms is also an advantage when it comes to publication and discovery.

4.3.5 Operational Procedures

Our design covers application creation (including the pre-required publication and discovery of substrates), management and execution. However, the management phase is also restricted to activation and the creation for composition.

4.3.5.1 Creation

In the creation phase, the substrate composition engine of each substrate provider publishes its substrate to the repository using a *PUT* request, with the WADL [18] description of the substrate as argument. Next, the virtual composition engine of the infrastructure provider sends a *GET* request to get the list of available substrates with their descriptions.

Figure 4.5 describes the flow for application creation, where four substrate providers use the *PUT* request to publish their substrates such as announcement player and voice recorder into the broker. Then, the infrastructure provider uses the *GET* request to discover all the stored substrates by different substrates provider.

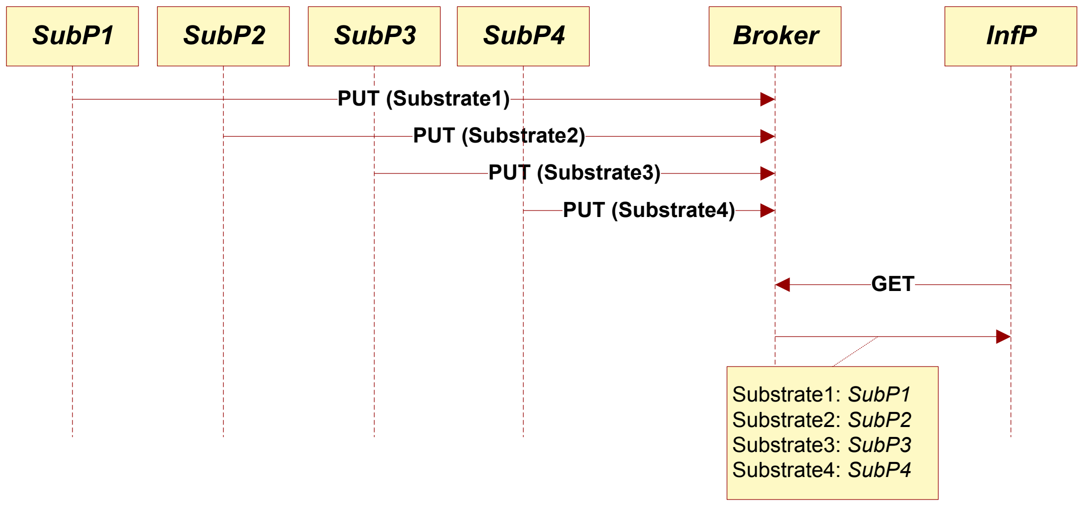


Figure 4.5: Substrate publication and discover

It should be noted that the *GET* request could have been sent anytime during the process to get the list of substrates available at that point in time. The list is then made available to the platform engine that adds a level of abstraction to the substrates by making them visible through a GUI. For example, the first service provider uses the GUI to develop an automated attendant by composing the substrates, while the second one develops an automated survey by the same process.

4.3.5.2 Activation

In the activation phase, the infrastructure provider's virtual composition engine uses the description file of the composed service (created in the previous step) and creates a different instance of the virtual management engine to communicate with each of the substrates involved in the composition process. Then, it instructs these instances to create a substrate IVR instance (SII) at each substrate.

The instantiation is done by sending a *POST* request to the appropriate substrate, along with the description file of the service instance to create and the identifier of the IVR service provider. When a substrate management engine receives an instantiation request, it verifies resource availability and then creates a new SII and allocates the necessary resources. Figure 4.6 describes the flow for application activation.

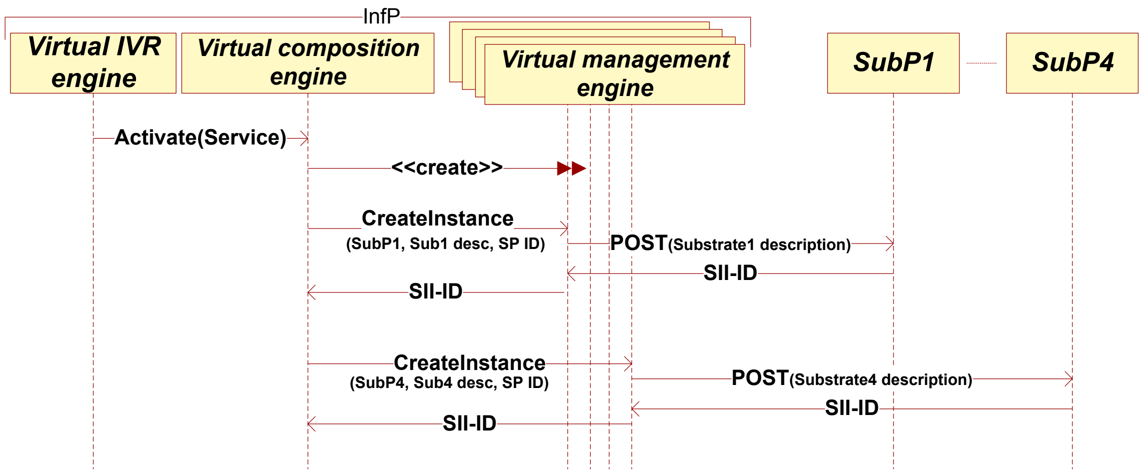

InfP
Virtual IVR engine
Virtual composition engine
Virtual management engine
SubP1
SubP4
Activate(Service)
<<create>>
CreateInstance (SubP1, Sub1 desc, SP ID)
POST(Substrate1 description)
SII-ID
SII-ID
CreateInstance (SubP4, Sub4 desc, SP ID)
POST(Substrate4 description)
SII-ID
SII-ID

Figure 4.6: Service activation

Finally, the created virtual IVR is published, so it can be used by other end users or service developers.

4.3.5.3 Execution

In the execution phase, the virtual IVR engine of the infrastructure provider receives an IVR request from a service provider. It gets the service description from the virtual composition engine (including the SIIs to use). Then it creates a virtual service engine instances to communicate with each of the substrates, and instructs the different instances to execute the appropriate sub-requests, following the request plan.

A request plan is a set of sub-requests and their execution sequence, along with the relevant substrates/SIIs that are required to answer an IVR request. The request plan is created by the virtual composition engine during the service creation phase.

When a substrate service engine receives a service execution request from the virtual service engine, it forwards the request to the appropriate SII, which then executes the request and replies back to the address provided in the request. Figure 4.7 describes the flow for application execution.

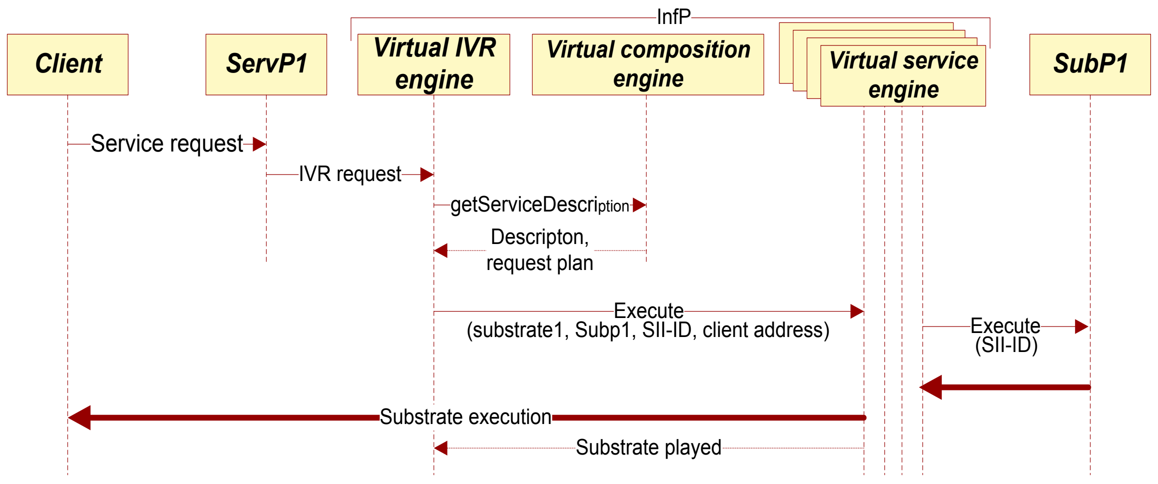


Figure 4.7: Service Execution

In this figure, the client call is transferred to the first substrate to be executed. When virtual IVR engine receive an IVR request, it first asks the virtual composition engine for the SII(s) and the request plan. Then the virtual IVR engine instructs the different instances to execute related sub-requests following the request plan. In the substrate service engine, the request is forwarded to the appropriate IVR instance using the SII-ID. At the end, the service instance executes the request to the appropriate client

The following chapter will add more information, where all the entities in our software architecture are presented and described.

4.4 Chapter summary

In this chapter, we first proposed our business model and then based on it we have presented our overall proposed architecture. As it was seen, the main functional entities of the architecture are the virtual IVR engine and the substrate IVR engine. In addition, our architecture is composed of three planes: service plane, management plane, and composition plane. However, these entities, planes and all the components are interconnected using three types of interfaces: service I/F, management I/F, and publication & discovery I/F. Lastly we have demonstrated the operational procedures of the architecture using three phases: creation, activation, and execution.

The refined architecture satisfies all the requirements derived in the previous chapter. First of all, using the publication & discovery I/F, the broker will enable the discovery and publication of substrates and substrate instances. In addition, the composition engine, allows service provider to compose their own applications. Also, using the management engine, different applications will be able to use the same substrate to develop IVR applications with the instantiation process of the IVR substrate available in the infrastructure layer. Besides, the platform provider will be able to add some level of abstractions to the substrates offered by the infrastructure provider. Lastly, the execution of a certain service will be transparent to third parties where the infrastructure provider will maintain the execution process.

In the next chapter, we will present the overall software architecture and then we will discuss the implemented IaaS prototype and the performance measurements that have been done to evaluate our architecture concept. Also, a simple proof of concept PaaS consisting of a GUI has been built to enable the development and management of simple IVR services in the SaaS layer.

Chapter 5

Validation: Prototype and Evaluation[2]

In the previous chapter we have proposed architecture for having a virtualized infrastructure for IVR applications as services. Consequently, in this chapter we focus on designing the software architecture and on validating the proposed architecture through an implemented prototype.

In the first section we present the overall software architecture. In the second, we describe the prototype that we have implemented as a proof of concept. Lastly, in order to validate our architecture, a scenario is presented and some performance measurements are collected and analyzed.

[2] This chapter is an extended version of the paper F. Belqasmi, C. Azar, M. Soualhia, N. Kara, and R.Glitho, "A Virtualized Infrastructure for IVR Applications as Services" accepted for presentation at the ITU Kaleidoscope conference, Cape Town, December 2011.

5.1 Overall Software architecture

In this section, we first describe our overall software architecture and then we describe the operational procedures of this architecture.

5.1.1 Software description

The overall software architecture is depicted in Figure 5.1. It consists of the infrastructure layer, the substrate IVR layer, and the IVR substrate repository.

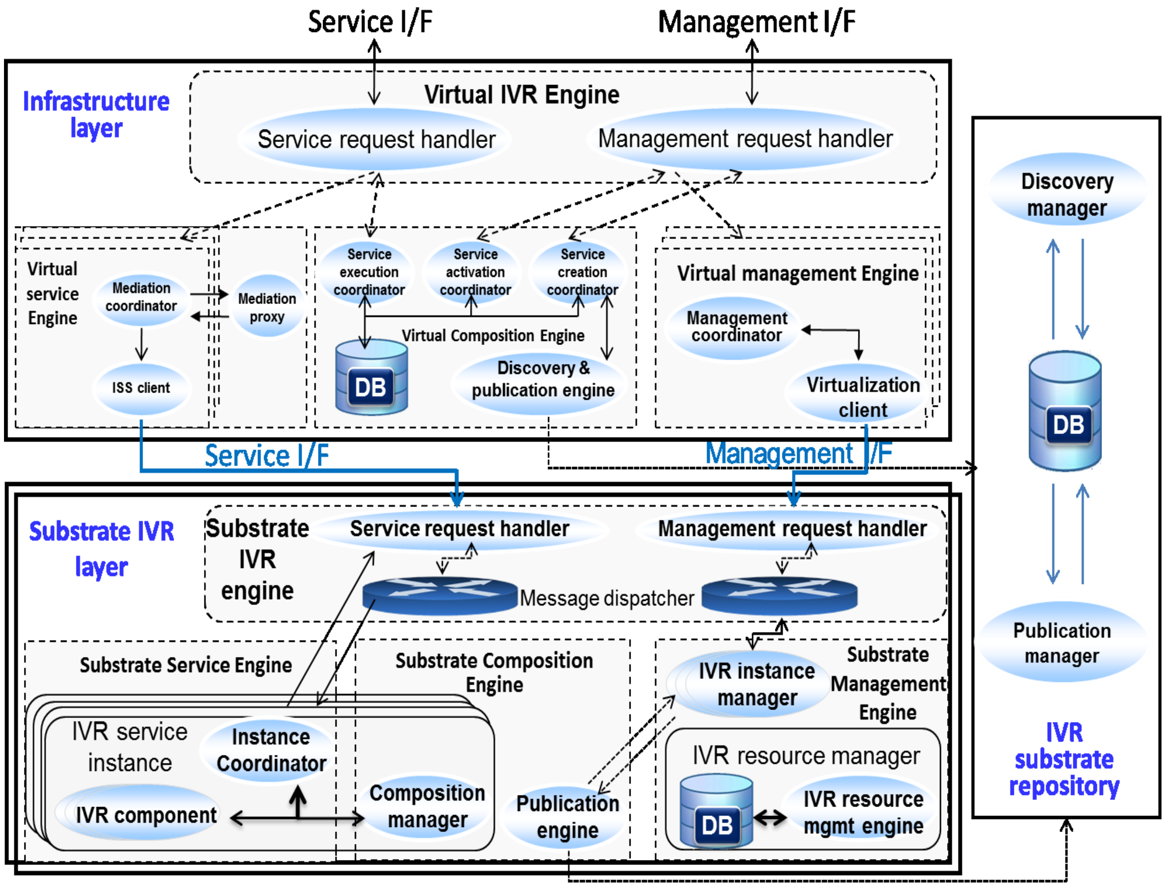


Figure 5.1: Overall Software architecture

5.1.1.1 Infrastructure Layer

The infrastructure layer also known as the virtualization layer is consists of the virtual IVR engine and the three engines discussed in the previous chapter: the virtual composition engine, the virtual management engine, and the virtual service engine.

- The virtual IVR engine handles requests and process responses from/to the platform layer. It includes two entities: the management request handler and the service request handler. The management request handler realizes the management interface provided to the cloud platform provider, whereas the service request handler realizes the execution interface provided to the cloud platform provider.

- The virtual composition engine includes four functional entities: the service creation coordinator, the service activation coordinator, the service execution coordinator and the discovery & publication engine. The service creation coordinator is responsible for managing and coordinating the composition of a new IVR service. The service activation coordinator is responsible for managing and coordinating the instantiation of new SIIs of the composed service. The service execution coordinator responsible for managing and coordinating the execution of a given request. Finally, the discovery & publication engine is used to publish the virtual IVR services and to discover the available substrates from the repository.

- The virtual management engine includes two entities: the management coordinator and the virtualization client. The management coordinator translates the requests received from the service activation coordinator into requests that the virtualization client should then send to the target substrate. The virtualization client allows the communication with a substrate at the creation and activation.

- The virtual service engine includes two entities: the mediation coordinator, and the ISS client. The mediation coordinator translates the request received from the service execution coordinator into requests the ISS client should send to the target substrate. The ISS client allows the communication with a substrate at execution. Since many substrate instances can be composed to provide a given virtual IVR service and these substrate may be provided by different substrate providers, therefore a mediation proxy is used to manage the execution of the execution of these substrates.

5.1.1.2 Substrate IVR Layer

The substrate IVR layer consists of the substrate IVR engine and also the three engines: the substrate composition engine, the substrate management engine, and the substrate service engine.

- The substrate IVR engine includes message dispatcher and the two entities: the management request handler and the service request handler. The message dispatcher dispatches the received management messages such as release/update an IVR instance to the appropriate IVR instance manager, whereas it forwarded the service plane messages to the target IVR service instance based on the instance ID. The management request handler provides a management interface for virtual IVR provisioning which implement also the management interface at the substrate level. Besides, the service request handler implements the service execution interface at the substrate level.

- The substrate management engine consists of IVR instance manager and IVR resource manager. The IVR instance manager is responsible for creating instances. Each instance is managed by a separate IVR instance manager. The relationships between service instances and their managers are saved when the service instances are created. Also, the created instance is only used by the provider for which it was created. The IVR resource manager controls the access to the IVR substrate resources. For example, it maintains and monitors the current state of the resources, it allocates the necessary resources for new instances, and it informs the appropriate IVR instance manager about the resources usage status or about a hardware problem that may affects an instance managed by an IVR instance manager.

- The substrate composition engine includes a publication engine and a composition manager. The publication engine is responsible for publishing the IVR substrates at each substrate provider. Also, it handles the publication of the created instances by the IVR instance manager at the activation phase. The composition manager allows creating a new IVR service component from existing IVR service components at each substrate provider and is a part of the IVR service instance.

- The substrate service engine includes an instance coordinator and an IVR component. Together with the composition manager form an IVR service instance. The IVR service component represents the substrate part of the IVR that includes a set of services such as announcement player and voice recorder. This component will handle the requests coming through the service interface from different users who are using the new IVR. The Instance coordinator allows identifying which service component to look for when the IVR service instance receives the execution requests coming through the service interface and when the composition manager sends a request to create a new IVR service component. Each instance is used by one and only one infrastructure provider and the instances are separate and independent.

5.1.1.3 IVR substrate repository

The IVR substrate repository also known as broker includes a publication manager, and a discovery manager.

- The discovery engine handles the discovery requests. It used to retrieve the IVR substrate instances saved in the Database (DB).

- The publication engine handles the publication requests. It used to store substrates, substrate instances, or even a composed IVR service along with it is description file, name, and uniform resource identifier (URI). In addition, the published service descriptions are stored in a local database.

5.1.2 Software Operational Procedures

The three operational phases discussed in the previous chapter, are explained more in this subsection in terms of the entities constituting the software architecture. Again, the software architecture offers service provider the ability to first create, activate, and execute an IVR service.

5.1.2.1 Creation

This phase is executed at the IVR Infrastructure layer. It may be handled via a graphical user interface offered by the platform providers. However, when the management request handler gets a creation request, it forwards it to the service creation coordinator in the virtual composition engine.

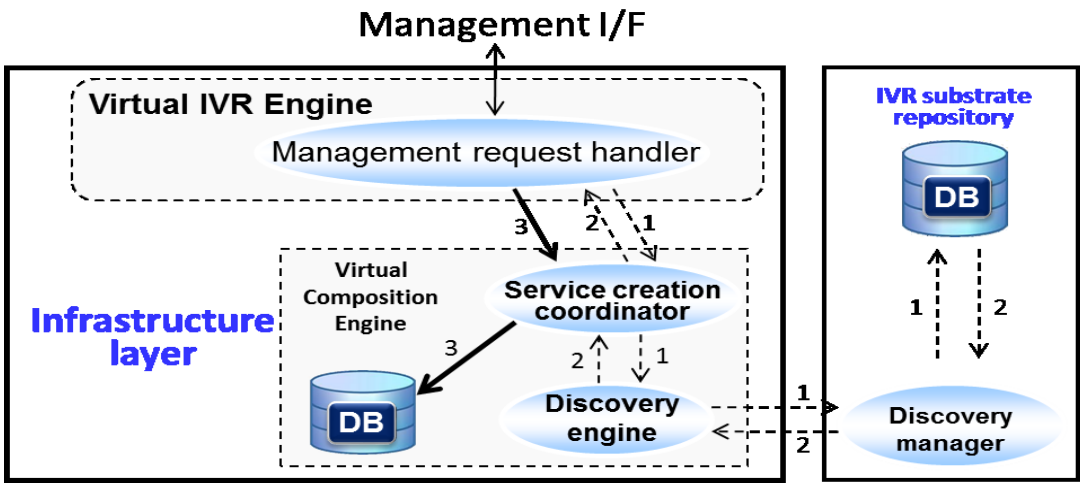


Figure 5.2: Creation phase

The service creation coordinator will use the discovery engine to get all the available substrates from the IVR substrate repository and sends them to the platform provider through the management request handler. In addition, when service providers compose a service, the service creation coordinator will take the inputs from the platform provider GUI for the service composition, create the description file for the composed service along with the request plan and store them in the local database. These files include the list and coordinates of the chosen IVR service substrates and the execution plan of the composed service.

5.1.2.2 Activation

At the infrastructure layer, when the management request handler gets an activation request, it forwards it to the service activation coordinator. In this phase, the service activation coordinator is responsible for managing and coordinating the instantiation of new SIIs. Therefore, it gets the description file from the local DB, and it creates a different instance of the IVR management engine to communicate with each of the IVR service substrates. Then, it instructs the instances to create a SII at each IVR service substrate.

At the IVR substrate layer, when the management request handler gets an instantiation request, it forwards it to the dispatcher. The dispatcher will spawn a new IVR instance manager that will create it and the relationships between service instances and their managers are saved using the publication engine. Lastly, the IVR instance manager forwards the request to the IVR resource manager which will check if there are enough resources available. If yes, it will create a new IVR service instance and allocates the necessary resources.

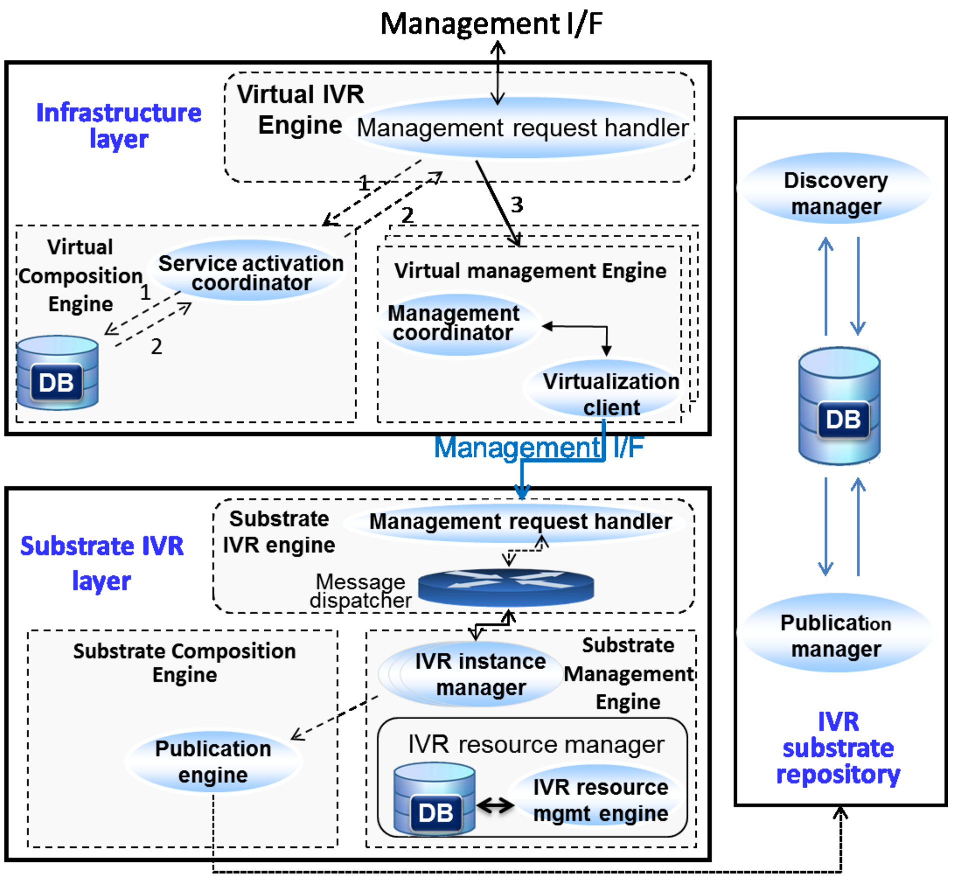


Figure 5.3: Activation phase

5.1.2.3 Execution

At the infrastructure layer, when the service request handler gets an IVR request which is for execution for example play announcement and collect digit. It asks the service execution coordinator, which is responsible for execution, for the IVR substrates that provide the virtual SIIs. Accordingly, the service execution coordinator brings the information which was stored in the DB when the virtual IVR was created. This includes the coordinates of the substrate, the type of the substrate, the instance ID, the substrate provider ID, the client address. Besides, it also asks for the request plan which is the set of sub-request and their execution sequence, along with the relevant ISSs/SIIs that are required to answer a client request (also was stored in the DB when the virtual IVR was created). Afterwards, the service request handler will create different service engine instance for each different substrate and then instructs the different instances to execute related sub-requests following the request plan. The mediation proxy will receive the request and forward it to the mediation proxy which will record the request's client address and the substrate provider ID and respond with its own address.

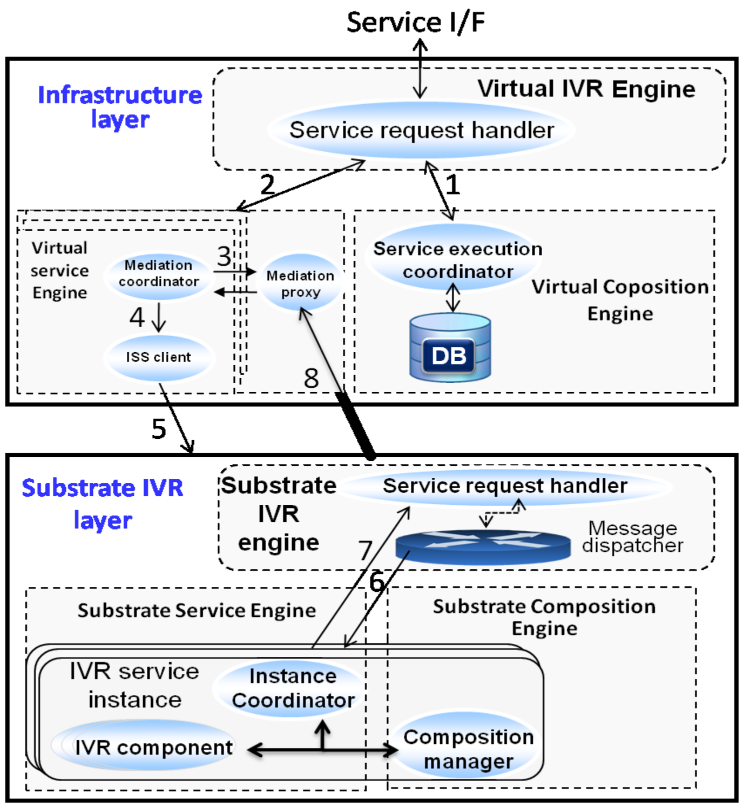


Figure 5.4: Execution phase

At the substrate IVR layer, the service request handler gets the request. It uses the IVR message dispatcher to forward the request to the appropriate IVR service instance using the instance ID. Finally, the service instance executes the requested replies back to the mediation proxy that forwards the response to the appropriate client.

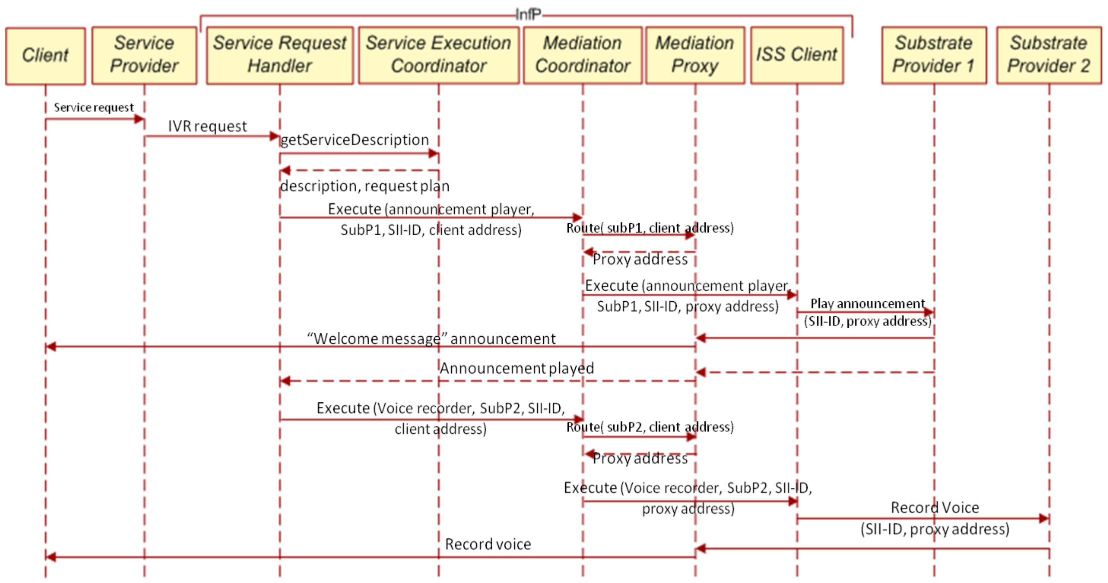

InfP
Client
Service Provider
Service Request Handler
Service Execution Coordinator
Mediation Coordinator
Mediation Proxy
ISS Client
Substrate Provider 1
Substrate Provider 2
Service request
IVR request
getServiceDescription
description, request plan
Execute (announcement player, SubP1, SII-ID, client address)
Route(subP1, client address)
Proxy address
Execute (announcement player, SubP1, SII-ID, proxy address)
Play announcement (SII-ID, proxy address)
"Welcome message" announcement
Announcement played
Execute (Voice recorder, SubP2, SII-ID, client address)
Route(subP2, client address)
Proxy address
Execute (Voice recorder, SubP2, SII-ID, proxy address)
Record Voice (SII-ID, proxy address)
Record voice

Figure 5.5: Execution call flow

Figure 5.5 shows the execution call flow diagram where two substrate providers are invoked in the execution of the substrate composing the IVR service requested by the client. Since the execution is fully transparent to third parties, we can observe that the execution of the announcement and the record substrates are managed by the infrastructure layer and specifically by the mediation proxy.

5.2 Prototype Design and Implementation

As a proof of concept, we have implemented a prototype that includes an end-to-end service plane and a sub-set of the composition plane. We start off this section with the assumptions and feature of our prototype, followed by the prototype architecture, the software tools that we have used for implementation and a simple scenario.

5.2.1 General assumptions

In the implemented prototype, we assume that the broker is composed of the five substrates announcement player, voice recorder, key detector, extension detector and call transfer. In addition, we assume that the substrates are supplied by one substrate provider. We further assume that we have one infrastructure provider (InfP), one platform provider (PP), two service providers (ServP1, ServP2) and one end-user with a subscription to one of the two service providers.

The substrates are described using WADL as implied by our choice of RESTful Web services as technology, and also with Donkey State Machine (DSM) [4]. The DSM description represents the substrate behaviour as a state machine and is included under the <doc> element of the WADL description. The use of DSM is motivated by the fact that it is used by the SIP Express Media Server (SEMS) [5] used in our prototyping environment, and because it makes the substrates' composition easier. DSM enables a textual description of applications (substrates in our case) that can be directly executed by interpreters hosted by the SEMS.

5.2.2 Prototype architecture

The general architecture of the implemented prototype is given in the figure 5.6. First of all, the service provider using the PP GUI will discover all the available IVR substrates using the GET method of the RESTFul Web services. The service provider will compose its own IVR service and using the PUT method the service will be published. Here we have assumed that the service is activated and all the resources are reserved before it is published. After that, the SerP will be able to provide the service to the clients and other applications. The second figure shows the execution phase of the prototype. The composed service is accessed via SIP.

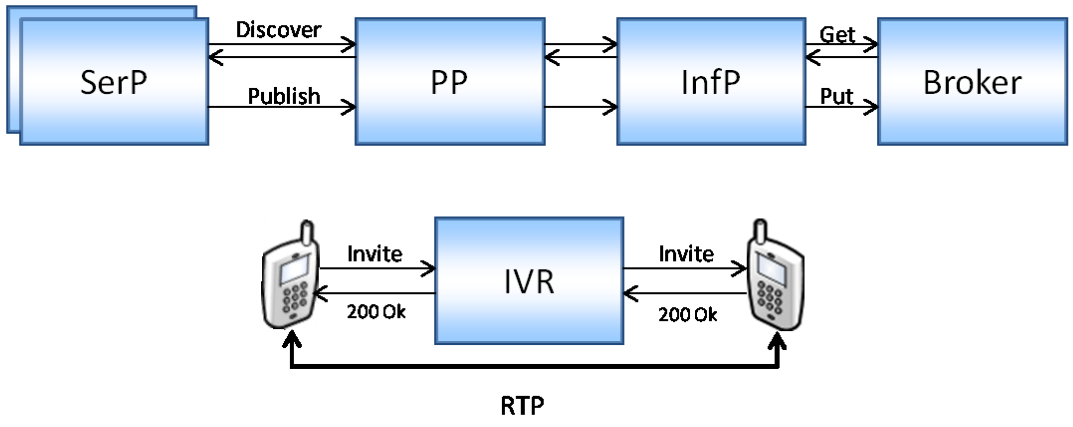


Figure 5.6: General Prototype architecture

5.2.3 Environmental settings

In order to test our implemented prototype, 5 nodes (4 laptops and 1 virtual machine) are used. All the laptops are equipped by 802.11 (WLAN) adaptive cards. Laptops are Intel core i7, all with 8 GB RAM whereas the virtual machine is running on the Xen server and managed by Xencenter with 4 GB of RAM. Also the laptops run windows 7 OS whereas the virtual machine runs Linux Ubuntu OS.

5.2.4 Software Tools

Several tools were used for the implementation of our prototype. In this subsection, we divide the tools into five categories: IVR substrates, virtualization, interfaces, repository and lastly some additional software used for execution and creation.

5.2.4.1 IVR Substrates

The IVR substrates are implemented using DSM [4] module and deployed on SEMS [5]. This sub-section gives a brief overview about the Sip Express Media Server (SEMS), its Donkey State Machine (DSM) module and its configuration steps.

5.2.4.1.1 SEMS overview

SEMS is a free, open source, and scalable media server for SIP based VoIP services [5]. It is responsible for media processing for applications requiring IVR functions. The core of SEMS is implemented to support the basic call and audio processing. However, various types of plug-ins may extend the system. For example, the application plug-ins is responsible for implementing new services' logic whereas audio plug-ins allows the addition of new codecs and file formats to the system [5].

SEMS is shipped with several applications such as conferencing and voicemail applications. However, developers can extend it by creating their own plug-ins using SEMS framework API in C++, python, or the DSM [5].

In the prototype DSM was chosen to extend SEMS and develop our substrates. Donkey State Machine is a scripting service development platform that makes it simple and rapid to implement powerful services. DSM enables a textual description of applications (substrates in our case) that can be directly executed by the DSM module as application in SEMS [4].

With DSM, the service logic is defined as a state machine. It consists of a states and transitions between the states as illustrated in Figure 5.7. The first state is the initial state with which the execution of the DSM logic will start. The transitions have a set of conditions. For example, when an event occurs, the transitions are checked in order. If all the conditions match, the transition is made. Transitions can have a set of actions also, which are executed when the transition is made. In addition, each state may have a set of actions that are executed on entering the state, and on leaving the state [4].

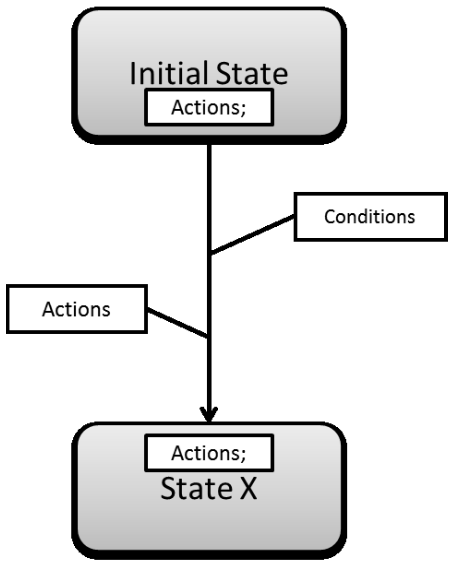
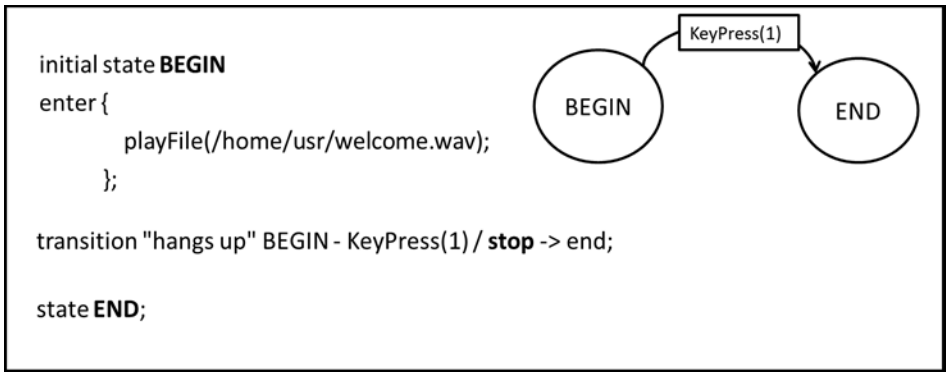


Figure 5.7: DSM state machine

Some of the DSM core conditions are test(#key==prefix*), keyPress(nb), invite, and sessionStart. Also, some of the core actions are playFile(), recordFile(), set, append, and stop. However, these conditions and actions could be extended by few additional DSM modules which provide more actions and conditions such as mod_mysql, mod_conference, and mod_dllg. These modules, in addition to the core conditions and actions, give developers the flexibility to build more advanced services.

Figure 5.8 shows a simple DSM example. In this example, there are two states: BEGIN and END. At first the state BEGIN will play a file, and then when the caller press 1, an event is fired, the transition is made, and the action is executed which is Stop.

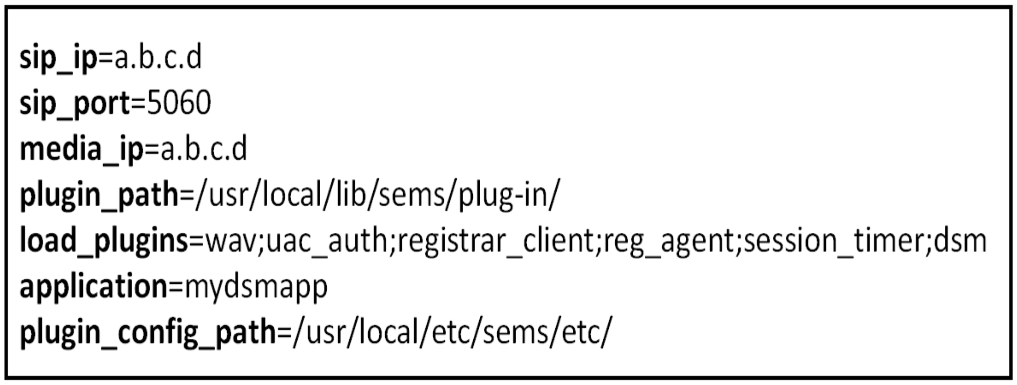


```
initial state BEGIN
enter {
        playFile(/home/usr/welcome.wav);
    };

transition "hangs up" BEGIN - KeyPress(1) / stop -> end;

state END;
```

Figure 5.8: DSM example

5.2.4.1.2 SEMS Configuration

1. Adapt **sems.conf** /usr/local/etc/sems/sems.conf:



```
sip_ip=a.b.c.d
sip_port=5060
media_ip=a.b.c.d
plugin_path=/usr/local/lib/sems/plug-in/
load_plugins=wav;uac_auth;registrar_client;reg_agent;session_timer;dsm
application=mydsmapp
plugin_config_path=/usr/local/etc/sems/etc/
```

Figure 5.9: SEMS configuration

- sip_ip and media_ip are optional parameters. They inform SEMS about the SIP IP where its SIP stack is bound to or should be bound to and about the IP address or interface that SEMS uses to send and receive media. If not set, defaults to first non-loopback interface.

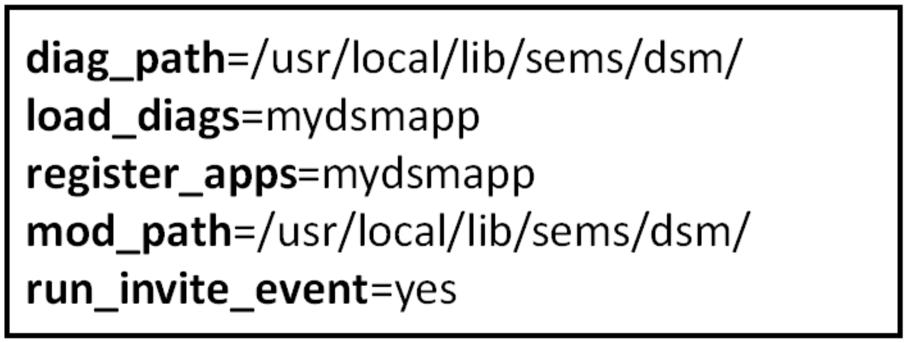
- sip_port is an optional parameter also. It informs SEMS about the port where its SIP stack is bound to or should be bound to. The default port is 5060.
- plugin_path sets the path to the plug-ins' binaries

- load_plugins list of modules to load. If it is empty, all modules in plugin_path are loaded.

- Application controls which application is to be executed if there is no explicit application requested from the SIP stack

- plugin_config_path in this path configuration files of the applications (e.g. mydsmapp.conf) is searched

2. Adapt **dsm.conf** /usr/local/etc/sems/etc/dsm.conf:



```
diag_path=/usr/local/lib/sems/dsm/
load_diags=mydsmapp
register_apps=mydsmapp
mod_path=/usr/local/lib/sems/dsm/
run_invite_event=yes
```

Figure 5.10: DSM configuration

- diag_path is the path of the DSM descriptions.
- load_diags loads the application (mydsmapp.dsm) in the path specified above
- register_apps register the mydsmapp.dsm diagrams as application in SEMS so it can be used in application=mydsmapp in sems.conf
- mod_path specify the path for importing the dsm modules for import(mod_name)
- run_invite_event is set so it can be used for early media

3. Adapt **mydsmapp.dsm** in /usr/local/lib/sems/dsm/
    - Mydsmapp.dsm contains the DSM description of the IVR substrate services.

4. Run SEMS from the command line like this:

    /usr/local/sbin/sems -f /usr/local/etc/sems/sems.conf -D 3 –E

5.2.4.2 Virtualization

For virtualization, XEN [20] was selected. It is open source software for virtualization technology. XenServer is the cloud-proven virtualization platform that consists of all the required capacity to create and manage a virtual infrastructure [20]. It is allows multiple guest operating systems (OSs) to run in virtual containers called domains in a single machine. It is a widely approved and agreed technology as the fastest and the most secure virtualization software in the industry [20]. As shown in figure 5.11, the Xen hypervisor provides a thin software virtualization layer between the guest OS and the underlying hardware.

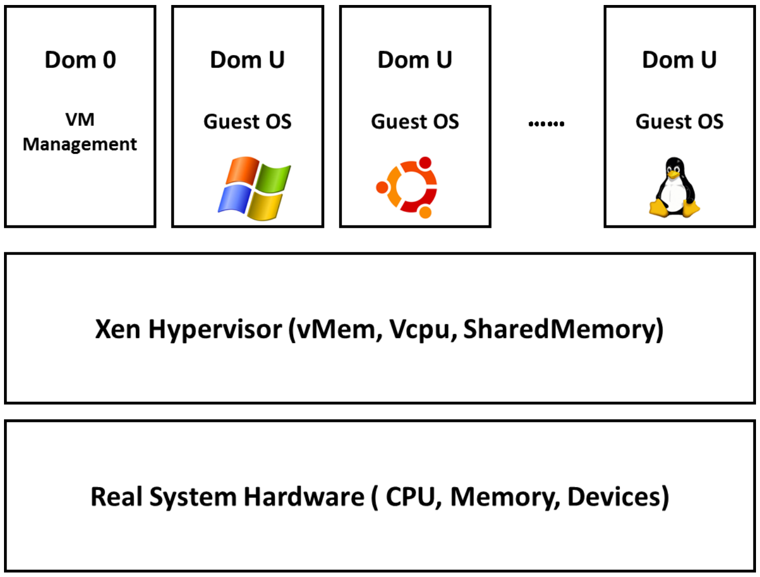

Dom 0
VM Management
Dom U
Guest OS
Dom U
Guest OS
......
Dom U
Guest OS
Xen Hypervisor (vMem, Vcpu, SharedMemory)
Real System Hardware (CPU, Memory, Devices)

Figure 5.11: XEN architecture

Domain 0, it is the privileged domain (Dom0) that has direct access on the hardware and responsible for managing multiple Domain Guest (Dom U).

XenServer version	5.6 Feature Pack 1
Edition	Citrix XenServer
CPU 0 – 11	Intel(R) Xeon(R) CPU Speed: 2532 MHz
Interfaces NIC 0 – 3	Speed 1000 Mbit/s
Server Memory	14326 MB
Local Storage	600 GB

Table 5.1: Xen Server general properties

The hypervisor contains several modules for managing the resources between the OSs such as the CPU scheduler that implements various scheduling policies and the memory management unit [32]. Table 5.1 shows the properties of the XEN server.

5.2.4.3 Interfaces and Repository

The REST interface of the substrate management engine is implemented using Jersey, an open source reference implementation of JSR 311. The interface module is deployed on a Glassfish server and it communicates with the SEMS using sockets. The implementation of the IVR substrate repository is also based on Jersey and deployed on a Glassfish server.

5.2.4.4 Additional Software tools

We choose Java based development APIs for the implementation of our platform provider GUI. We use Eclipse IDE for Java and Report Developers version Helios

Service Release 2. Eclipse offers an integrated development environment (IDE) to build java applications and provides a ready environment to deploy applications on Glassfish Server. Glassfish server is used to deploy the web application. It is an open-source, java enterprise edition compliant application server [34]. Also Jersey APIs (JSR 311) were used for building RESTful web services [35]. At the service plane, the composed IVR service is accessed via SIP. Lastly, a free SIP client, X-Lite [33], is used as the IVR client.

5.2.5 Simple experimentation

As a prototype, we implemented an automated attendant service. The client calls a company`s generic number, gets an announcement inviting him/her to enter the extension of the person to reach, the client provides the extension and is then connected to that person. If that person does not answer, the client can leave a voice message. Accordingly, this scenario is composed of the following substrates:

- Play announcement
- Voice recorder
- Extension detector
- Call bridge

Also, if the destination number wasn't available, a "No Answer" condition is added to the service description and a "Leave a Message" action is executed. Figure 5.12 shows a simplified WADL description of the Announcement player substrate.

```
<?xml version="1.0"?>
<application xmlns:xsi:shemaLocation="http://wadl.dev.java.net/2009/02">
<doc xml:lang="DSM" title="Play announcement substrate service">
     state Play enter{playFile(/home/user/welcome.wav); };
</doc>
  <resources base="http://substrateProvider1.com/">
      <resource path="playAnouncement">
      <method name="POST" id="instantiate">
      <request>
          <representation mediaType="application/xml" >
          <param name="dsm_description" type="xsd:string" required="true"/>
          <param name= "serviceProvider" type="xsd:string" required="true"/>
          </representation>
      </request>
      <response status="200">
          <representation mediaType="application/xml" >
          <param name= "resourceURI" type="xsd:anyURI" required="true"/>
          </representation>
      </response>
      </method>
      </resource>
  </resources>
</application>
```

Figure 5.12: Announcement player WADL file

Figure 5.13 illustrate the idea. When IVR receive an incoming call:

- It will play a welcome prompt to the caller.
- It will request the caller to enter the extension number.
- It will capture the DTMF input (extension number) entered by the caller via its telephone keypad.
- If the extension exist, IVR application contacts the extension requested by the caller and bridges the caller to the callee
- If the extension is incorrect, IVR application will play a prompt and re-ask for the extension.
- If the callee doesn’t respond, a no answer condition is fired and the caller will be able to leave a message then hang up.

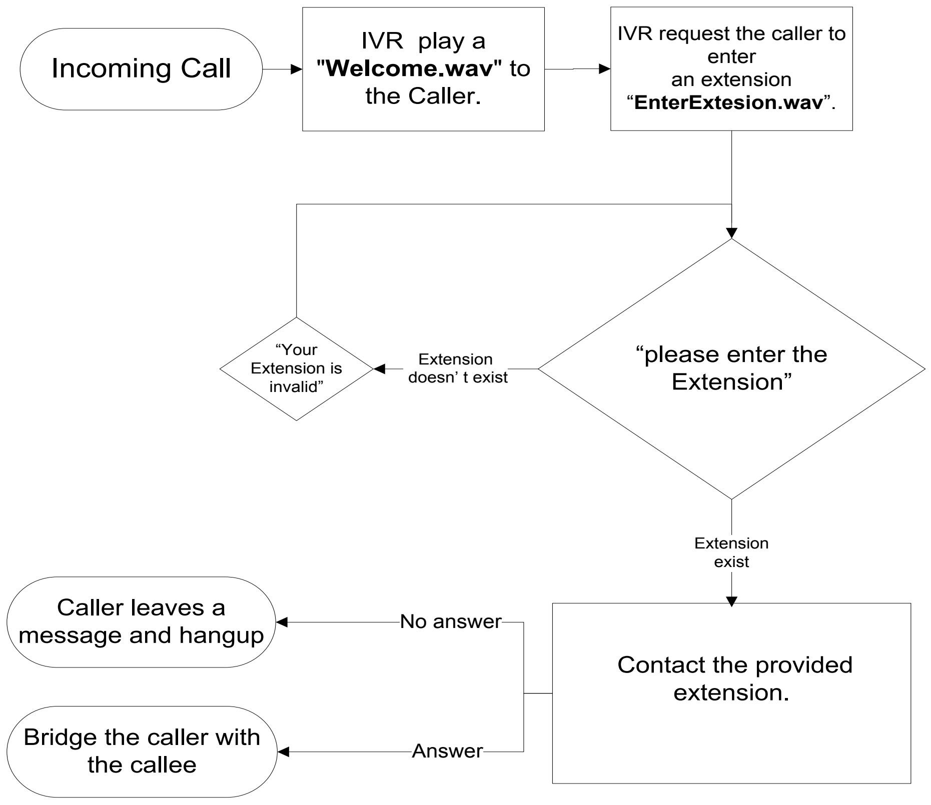

Incoming Call
IVR play a "Welcome.wav" to the Caller.
IVR request the caller to enter an extension "EnterExtesion.wav".
"please enter the Extension"
Extension doesn' t exist
"Your Extension is invalid"
Extension exist
Contact the provided extension.
No answer
Answer
Caller leaves a message and hangup
Bridge the caller with the callee

Figure 5.13: Automated Attendant Flow chart

5.3 Preliminary Validation and Performance Evaluation

5.3.1 Testing Scenario

As a prototype, we implemented the scenario where the first service provider creates and provisions an automated attendant service.

Figure 5.14 presents the GUI offered by the platform provider to the service provider. Service providers will use this GUI to develop and manage IVR applications.

First of all, the service provider pushes the “Connect” button in order to establish a connection with the platform provider.

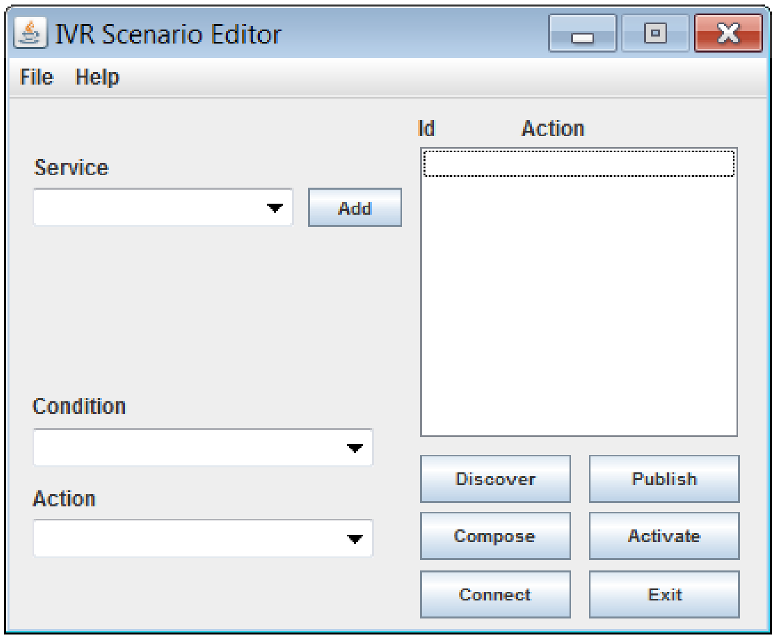


Figure 5.14: Platform Provider GUI

The service provider won’t be able to build an IVR application unless the Connect button is pushed first. If an attempt occurred and the Discover button is pressed before the Connect, a pop-up message will appear asking the service provider to push Connect first. Figure 5.15 shows that “IVR Scenario Editor – Connected” on the header of the GUI, which mean that the service provider is now connected to the platform provider and is able to develop and manage an IVR application using this platform graphical user interface.

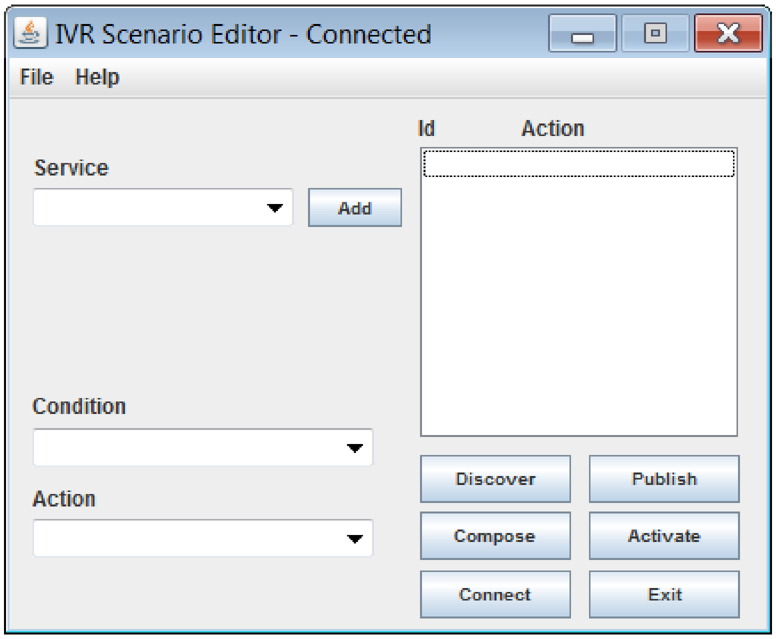


Figure 5.15: Platform Provider Connect

Now, using the "Discover" button, the GUI will get all the name, description and URI of the existing substrate services, which is discovered by interrogating the substrate repository.

Figure 5.16 shows the discovered substrates in the list. The substrates are developed using DSM and deployed on SEMS as it was mentioned before. In addition, they are described using WADL.

Now, the service provider is able to compose and manage a new service from the existing substrates. The "Add" button is used to add substrates to the new composed service. Also, a condition and an action could be added between the substrates in which when a condition occurs it will fire an action.

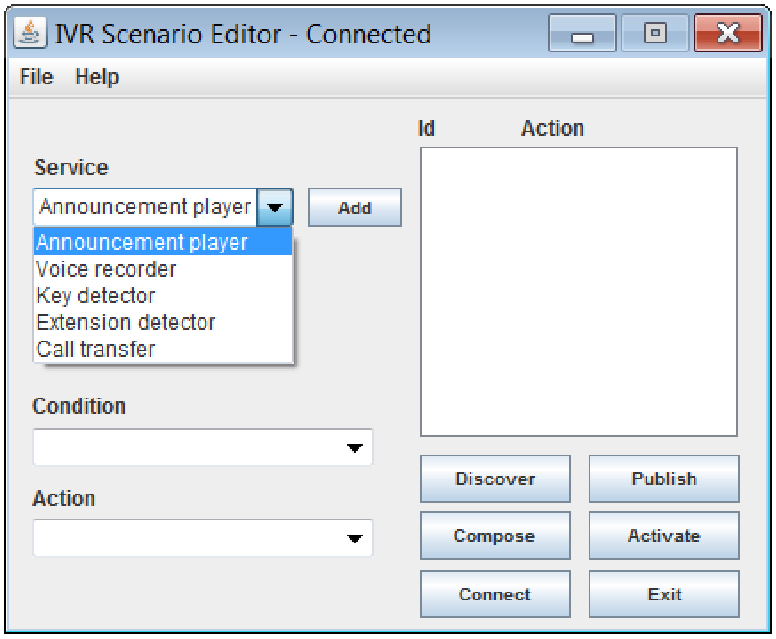
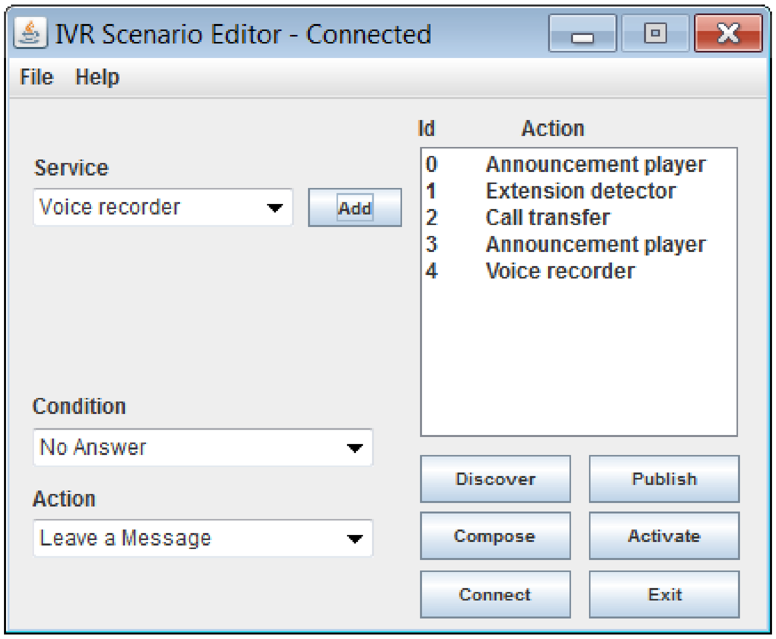


Figure 5.16: Platform provider discover

The GUI allows the service provider to create its composed service by choosing the substrates to compose and then ordering them graphically. As a prototype, we implemented an automated attendant service.



Service: Voice recorder

Condition: No Answer

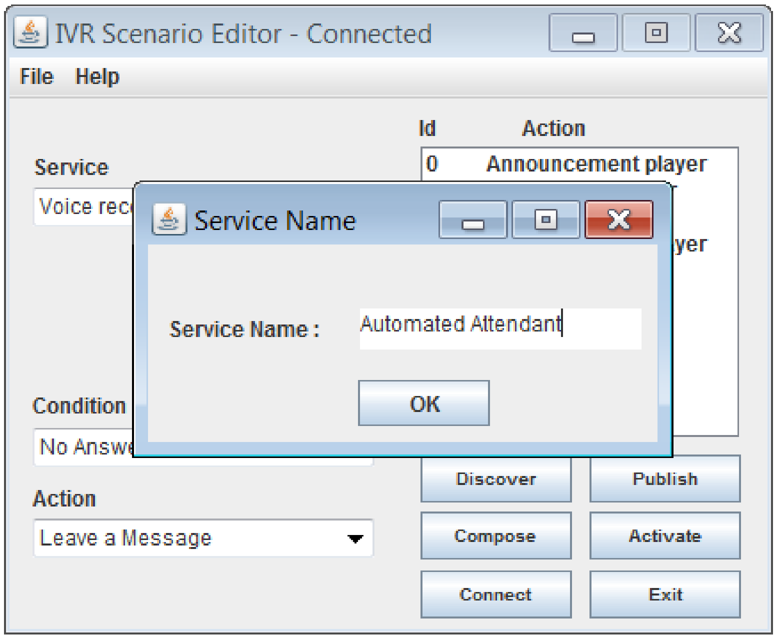
Action: Leave a Message

Id	Action
0	Announcement player
1	Extension detector
2	Call transfer
3	Announcement player
4	Voice recorder

Figure 5.17: Platform provider compose

Accordingly, the scenario is composed of the substrates shown in figure 5.17. Also, a "No Answer" condition is added to the service description in case the destination number wasn't available and a "Leave a Message" action is executed.

When the "Compose" button is pushed, the service provider should give a name to the new composed service. Also, a request is sent to the service creation coordinator in order to generate the DSM description of the composed service. The service creation coordinator also generates the execution plan for the composed service and stores the DSM description and the execution plane in the local database. Figure 5.18 shows the creation of the automated attendant service.



Voice rec

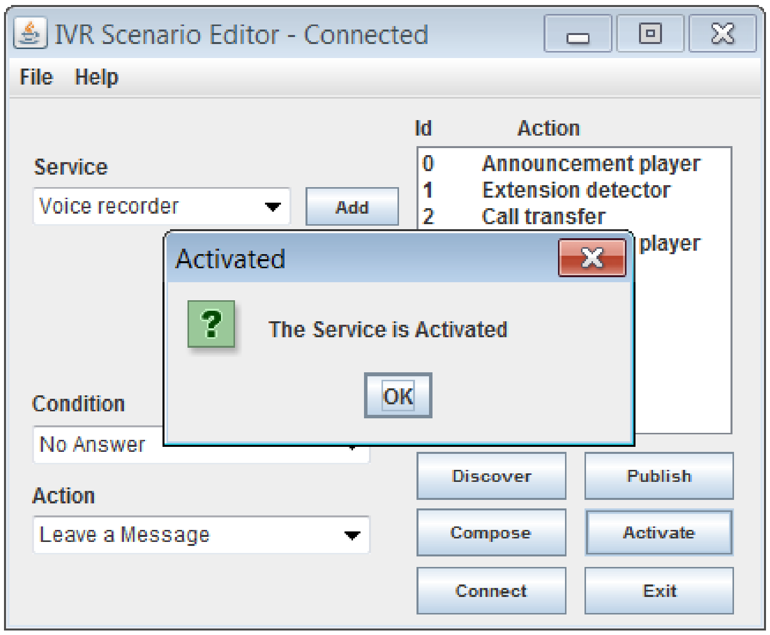
No Answe

Leave a Message

Service Name : Automated Attendant

Figure 5.18: Platform provider create

Figure 5.19 shows that the service is activated by pressing the Activate button. Here, as it was mentioned before we assumed that the service is activated and all the computing resources are allocated.



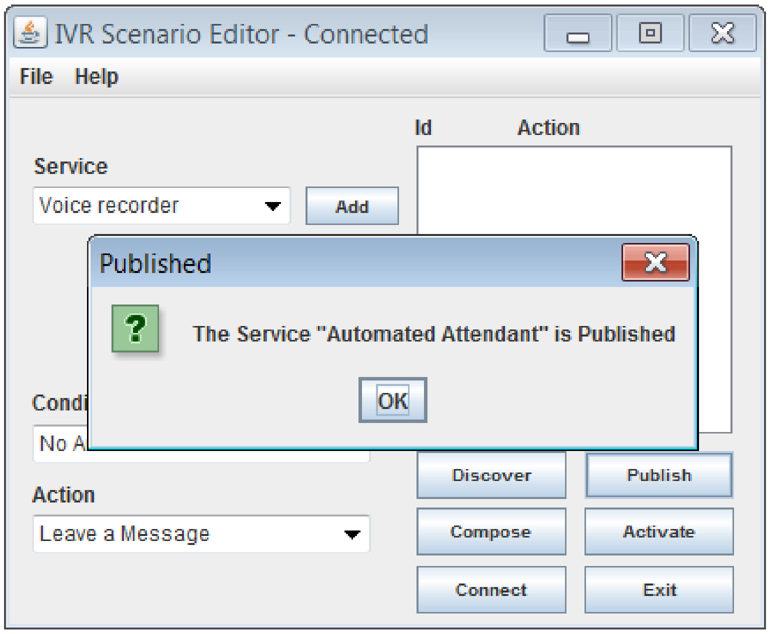
Activated

The Service is Activated

OK

Figure 5.19: Platform Provider Activate

Lastly, after the composed service is created, the service provider can publish it to the substrate repository using the "Publish" button, so that other service providers or clients can use it.



Published

The Service "Automated Attendant" is Published

OK

Figure 5.20: Platform Provider Publish

Now, if the second service provider presses the "Discover" button, the Automated Attendant service will appear between the discovered substrates. Figure 5.21 illustrates the idea where the list of existing substrate services and the composed service are shown.

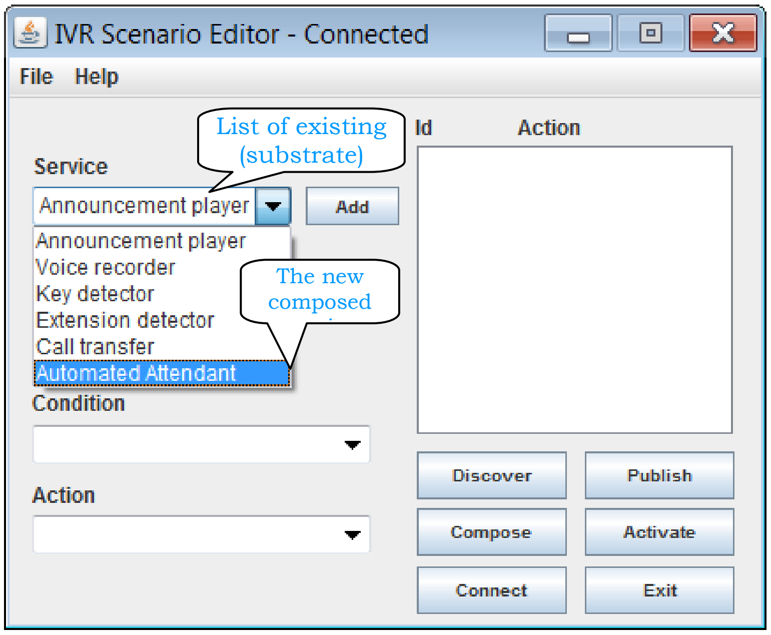


Figure 5.21: Platform Provider re-discover

The developed Automated Attendant service was tested where an end user calls the virtualized IVR, and after entering the designated extension number, the IVR transfer the call to the dialed extension in which the two end users can talk.

5.3.2 Measurements setup and analysis

This subsection presents the measurements setup of the prototype and analyzes the collected measurements.

5.3.2.1 Measurements setup

For the measurements, we have first measured the time needed to discover all the substrates available in the broker, publish a new developed service, and again re-discover all the substrates/service(s) after publishing our new composed service using the platform provider GUI. We have run the service provider, platform provider, infrastructure provider and the broker on different laptops in the same network and collected the measurements. Also, for the execution plane, the automated attendant service has shown a great efficiency. The measurements done cover the total time needed for the SIP INVITE message to travel between the two clients through a virtualized IVR and non-virtualized IVR. However, to eliminate all the delay from the played files by the IVR and the delay from human being to enter an extension number, we have adapted the IVR service in which when the IVR receive an INVITE message, it calls directly the designated client. Furthermore, we ran the two client (xlite v3 and xlite v4) on the same machine to avoid the time synchronization whereas the IVR (SEMS) on the virtual machine of the Xen server in the virtualized environment and on a regular laptop in the non-virtualized environment. Furthermore the two clients and IVR were registered with the registrar of the SEMS “iptel” registrar.

5.3.2.2 Measurements analysis

Table 5.2 shows the collected measurements of the composition plane. The purpose of this table is to show the time it take from prototype point of view to execute different operation. The first column represents the round-trip time (RTT) needed for the service provider to discover all the available IVR substrate in the broker. The second column represents the RTT needed to publish a service into the repository as it was shown in the previous subsection. Lastly, the third column represents the RTT needed to discover all the available services again after publishing the new composed service by a different service provider.

As we can notice from Table 5.2, the RTT to discover all the IVR substrates is 330.2ms. The time to publish a new composed service is approximately 446.6ms. Lastly, the time to discover again all the services including the new created one is 357.8ms which is slightly affected after publishing the automated attendant services and not observed by the service providers.

Discover	Publish	Re-Discover
327	515	343
421	546	419
278	498	315
219	485	356
446	519	384
418	483	421
291	435	331
364	358	394
274	284	268
294	343	347
330.2 ms	446.6 ms	357.8 ms

Table 5.2: Composition plane performance measurements

In addition, Table 5.3 shows the collected measurements of the execution plane of the composed service. Wireshark [40] tool was used to capture the network traffic and collect the times. The first column represents the time needed for the SIP INVITE message to travel between the two clients in a non-virtualized environment whereas the second columns shows the same measurements where IVR run on in a virtualized environment.

Non-Virtualized Environment	Virtualized Environment
498	**556**
522	**626**
496	**559**
509	**557**
502	**549**
508	**583**
622	**553**
502	**547**
498	**560**
523	**567**
518 ms	**566 ms**

Table 5.3: Execution plane performance measurements

The purpose of this table is to show that a very low overhead is imposed in the virtualized environment due to the hardware sharing. This time difference between the two environments is barely observed by end-users. Therefore, the use of the proposed novel architecture does not penalize the system's performance. On the contrary, it enables the service providers to develop and manage its own IVR services in virtualized environments full of benefits.

5.4 Chapter summary

In this chapter, we have first introduced our overall software architecture and then we have presented the prototype implemented as a proof of concept. The implemented prototype is based on RESTFul Web services. The IVR substrates were developed using DSM and deployed on SEMS. Xen server was used for virtualization. We ran the prototype in environment consisting of two substrate provider and one service provider and we conclude that it works properly. To validate our proposed architecture, an automated attendant scenario was examined and some performance measurements were collected. Through these experiments, we have learned that our proposed architecture is a valid and promising approach for having a virtualized infrastructure for IVR applications as services. In the next chapter, we will summarize the contribution of the thesis and propose some additional future works.

Chapter 6

Conclusion and Future Work

In this chapter, we will first summarize the contributions of this thesis and then we will discuss the remaining items which can be considered as the future work.

6.1 Summary of the Contributions

Several commercial hosted IVRs allow users to remotely access IVR applications as services. However, to the best of our knowledge these applications do not rely on virtualized infrastructures.

As a part of the contributions of this thesis, we have first identified the related requirements for a virtualized infrastructure for IVR applications as services which describe our major challenges. These requirements fall into two principal categories. The first category concerns the general requirement for having a virtualized infrastructure for IVR applications as services. The second concerns some other specific requirements related to the virtualized Infrastructure as a Service (IaaS) layer. Also, we have performed a detailed survey on the existing related works for having a virtualized infrastructure for IVR applications as services. We have divided these related works into two approaches: cloud overall approaches and IaaS approaches. Afterwards, we have related these approaches to our requirements and defined their usage in this thesis and accordingly, we observed that none of them meets all of our requirements.

As the core contribution of this thesis, we have first proposed a business model which introduces the IVR substrate provider at the infrastructure layer as a new role in the cloud business model. This business model is composed of an IVR substrates provider, an IVR infrastructure provider, an IVR platform provider, an IVR service provider, a Broker, and a Connectivity provider. Then, based on the business model, we have proposed a novel architecture for a virtualized IVR infrastructure that fulfils the requirements derived before. This architecture allows different IVR service providers to share the same IVR substrates. Also it enables easy development and management of new IVR-based applications via a simplified platform. The main functional entities of the architecture are the virtual IVR engine and the substrate IVR engine. It is also composed of three planes: service plane, management plane, and composition plane. These entities, planes and all the components in the architecture are interconnected using three types of interfaces: service I/F, management I/F, and publication & discovery I/F. Lastly we have demonstrated the operational procedures of the architecture using three phases: creation, activation, and execution. We have also designed the overall software architecture where all the entities, the planes and operational procedures have been discussed.

As a proof of concept, we have implemented an IaaS prototype and a simple PaaS graphical user interface that shows how a new service (i.e. automated attendant) can be created by composing a number of existing simple IVR services provided by the infrastructure. XEN server [20] was selected for virtualization to ease the creation of IVR substrate instances whereas RESTFul Web services for publication/discovery interfaces. IVR substrates are implemented using DSM and deployed on a SEMS server. PaaS consisting of a graphical user interface (GUI) has been built to enable the development and management of simple IVR services in the SaaS layer. To test our prototype, we implemented the scenario where a service provider creates and provisions an automated attendant service. Lastly, to validate our prototype, a preliminary performance evaluation of the overall architecture has been taken. Based on these results, we conclude that our architecture is valid and promising approach for a virtualized infrastructure for IVR applications as services.

6.2 Future Work

The defined architecture gives a global solution form providing a virtualized infrastructure for IVR applications in the cloud environment. It concentrates on allowing different IVR service providers to share the same IVR substrates, and enables easy development and management of new IVR-based applications via a simplified platform. However, we still need more details about some functional engines and entities in our proposed architecture such as the substrate composition engine for the appropriate resource allocation and management. Also the running simulation needs to be extended to the overall architecture including the activation plane. We need to study the performance impact of the procedures by having all the three approaches tested to determine the exact limitations of each. My colleague is currently working on this phase which is the activation plane which helps improving the resource utility in a virtualized platform for IVR applications and the next step will be to run the overall software architecture.

Finally, our future works also include further research for applications other than IVR such as audio presence, video conferencing and IPTV that can be offered in cloud settings.

Bibliography

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition", *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, January 2009.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud Computing: State of the Art and Research Challenges", *Journal of Internet Services and Applications, Springer*, Vol. 1, no. 1, 2010

[3] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim, "Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones", *Consumer Communications and Networking conference, CCNC, 5th IEEE*, 2008.

[4] Iptel.org, DSM: State machine notation for VoIP applications, available at: http://ftp.iptel.org/pub/sems/doc/current/ModuleDoc_dsm.html, Accessed February 20 2011

[5] Iptel.org, SIP Express Media Server (SEMS), available at: http://www.iptel.org/sems, Accessed February 20 2011.

[6] P. Rodriguez, D. Gallego, J. Cervino, F. Escribano, J. Quemada, and J. Salvachua, “VaaS: Videoconferencing as a Service”, *5th International Conference on Collaborative Computing: Networking, Application and Worksharing,* 2009.

[7] J. Li, R. Guo and X. Zhang, “Study on Service Oriented Cloud Conferencing”, *Third IEEE International Conference on Computer Science and Information Technology*, 2010.

[8] F. Belqasmi, N. Kara, R. Glitho, “A novel virtualized presence service for future Internet”, *Workshop on Future Networks, IEEE International Conference on Communications,* 2011.

[9] T. Aoyama, “Overview of the new generation network R&D”, *Japan, 4th CFI, Seoul, Korea*, 2009.

[10] K. Tutshuku, T. Zinner, A. Nakao, and P. Tran-Gia, “Network Virtualization: Implementation Steps Towards the Future Internet”, *Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen*, 2009.

[11] J. Carapinha, J. Jiménez, “Network virtualization: a view from the bottom”, *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures (VISA-09)*, pp. 73-80, 2009.

[12] M. Stecca and M. Maresca, "An Architecture for a Mashup Container in Virtualized Environment", *IEEE 3rd International Conference on Cloud Computing*, 2010.

[13] P. de Leusse, P. Periorellis, P. Watson, and A. Maierhofer "Secure and Rapid Composition of Infrastructure Services in the Cloud", *Sensor Technologies and Applications, SENSORCOMM*, 2008.

[14] R. Uhlig, G. Neiger, D. Rodgers, and A. L. Santoni, "Intel Virtualization Technology", *Intel Corp., USA*, vol. 38, pp. 48-56, 2005.

[15] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud Computing: An Overview", *CloudCom Lecture Notes In Computer Science*, vol. 5931, pp. 626-631, 2009.

[16] R. L. Grossman, "The Case for Cloud Computing", *IEEE IT Professional Magazine*, vol. 11, pp. 23-27, 2009.

[17] L. Richardson and S. Ruby, "RESTful Web Services", O' Reilly & Associates, ISBN 10: 0-596-52926-0, May 2007.

[18] W3C Member Submission, "Web Application Description Language", 31 August 2009.

[19] B. Hayes, “Cloud computing”, *Communications of the ACM – Web science*, vol. 51, pp. 9–11, July 2008.

[20] Citrix, Citrix Systems Inc. XenServer, available at: http://www.citrix.com/English/ps2/products/product.asp?contentID=683148&ntref=prod_top, Accessed February 30 2011.

[21] VMware ESX Server, available at: http://www.vmware.com/products/esx, Accessed May 10 2011.

[22] Amazon Elastic Computing Cloud, available at: http://aws.amazon.com/ec2, Accessed May 10 2011.

[23] Google, Google App Engine, available at: http://code.google.com/appengine, Accessed May 10 2011.

[24] R. Buyya, C. Shin Yeo, and S. Venugopal, “Market oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities”, *High Performance Computing and Communications, 10th IEEE International Conference*, 2008.

[25] J. Geelan, “Twenty-one Experts Define Cloud Computing”, *Virtualization Electronic Magazine*, August 2008.

[26] Cloud Hosting, Cloud Computing and Hybrid Infrastructure from GoGrid, available at: http://www.gogrid.com, Accessed May 20 2011.

[27] Dedicated Server, Managed Hosting, Web Hosting by Rackspace Hosting, available at: http://www.rackspace.com, Accessed May 20 2011.

[28] Salesforce CRM, available at: http://www.salesforce.com/platform, Accessed May 20 2011.

[29] M. Soujanya, and S. Kumar, "Personalized IVR system in Contact Center", *Electronics and Information Engineering (ICEIE)*, vol. 1, 2010.

[30] S. Xu, W. Gao, Z. Li, S. Zhang, and J. Zhao, "Design of hierarchical and Configurable IVR System", *Computational Intelligence and Natural Computing Proceedings (CINC)*, vol. 2, pp. 205-208, September 2010.

[31] C.-X. Qi, and Q.-D. Du, "A Smart IVR system based on application gateways", *Hybird Intelligence Systems*, vol. 3, pp. 110-115, August 2009.

[32] S. Sukaridhoto, N. Funabiki, T. Nakanishi, and D. Pramadihanto, "A Comparative Study of Open Source Softwares for Virtualization with Streaming Server Applications", *IEEE 13th International Symposium on Consumer Electronics*, pp. 577-581, May 2009.

[33] X-lite softphone, available at: http://www.counterpath.com/x-lite.html, Accessed March 9 2011.

[34] Glassfish Community, Glassfish server, available at: http://glassfish.java.net, Accessed April 25 2011.

[35] JSR 311: JAX-RS: The JavaTM API for RESTFul Web Services, available at: http://jersey.java.net/, Accessed April 25 2011.

[36] "RTP: A Transport Protocol for Real-Time Applications – IETF RFC1889". January 1996. Available at: http://www.ietf.org/rfc/rfc1889.txt, Accessed May 15 2011.

[37] "SIP: Session Initiation Protocol – IETF RFC 3261". June 2002. Available at: http://www.ietf.org/rfc/rfc3261.txt, Accessed May 15 2011.

[38] "Media Server Control Markup Language (MSCML) and Protocol – IETF RFC 5022". September 2007. Available at: http://tools.ietf.org/html/rfc5022, Accessed May 20 2011.

[39] "Sip Interface to VoiceXML Media Services – IETF RFC 5552". May 2009. Available at: http://tools.ietf.org/html/rfc552, Accessed May 21 2011.

[40] WIRESHARK network protocol analyzer, available at: http://wireshark.org, Accessed August 10 2011.