A Cloud Infrastructure for Multimedia Conferencing Applications

Flora Taheri

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

May 2014

© Flora Taheri, 2014

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis is prepared

By: Flora Taheri

Entitled: A Cloud Infrastructure for Multimedia Conferencing Applications

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

Master of Science (Computer Science)

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

Signed by the final Examining Committee:

_____ Chair
Dr. G. Grahne

_____ Examiner
Dr. J. Rilling

_____ Examiner
Dr. D. Goswami

_____ Supervisor
Dr. R. Glitho

Approved by: _____
Chair of Department or Graduate Program Director

_____2014 _____
Dean of Faculty

ABSTRACT

A Cloud Infrastructure for

Multimedia Conferencing Applications

Flora Taheri

Conferencing enables the conversational exchange of media between several parties. Conferencing applications are among important enterprise applications nowadays. However, fine grained scalability and elasticity remain quite elusive for multimedia conferencing applications, although they are key to efficiency in the resource usage.

Cloud computing is an emerging paradigm for provisioning network, storage, and computing resources on demand using a pay-per-use model. Cloud-based conferencing services can inherent several benefits such as resource usage efficiency, scalability and easy introduction of different types of conferences.

This thesis relies on a recently proposed business model for cloud-based conferencing. The model has the following roles: conferencing substrate provider, conferencing infrastructure provider, conferencing platform provider, conferencing service provider, and broker. Conferencing substrates are generally atomic and served as elementary building blocks (e.g. signalling, mixing) of conferencing applications. They can be virtualized and shared among several conferencing applications for resource efficiency purposes. Multiple conferencing substrates provided by different conferencing substrate

providers can be combined to build a conferencing service (e.g. a dial-out signalling substrate and an audio mixer substrate can be composed to build a dial-out audio conference service).

This thesis focuses on the conferencing infrastructure provider and conferencing substrate provider roles. It proposes a virtualized cloud infrastructure for multimedia conferencing applications. This infrastructure relies on fine grained conferencing substrates (e.g. dial-out signalling, dial-in signalling, audio mixer, video mixer, floor control, etc.) and offers several advantages in addition to fine grained scalability and elasticity (e.g. assembling substrates on the fly to build new conferencing applications). An architecture is proposed to realize the roles of conferencing infrastructure provider, conferencing substrate provider and their interactions. A resource allocation mechanism for conferencing substrates is also proposed. We have also built a prototype with Xen as the virtualization platform and validated the architecture. Performance has also been evaluated.

Acknowledgements

First I would like to express my deepest gratitude to my supervisor Dr. Roch Glitho for all his support, assistance and guidance throughout my thesis. This work would not have been possible without his encouragement, advices and countless drafts reviewed. It was a great opportunity working with him and learning from him during my research. I would also like to thank Dr. Belqasmi and Dr. Kara for their support, ideas and comments throughout my work.

I'm grateful to Dr. Rilling and Dr. Goswami for serving as members of my thesis committee and for their valuable feedbacks. I am also grateful to Dr. Grahne for chairing the thesis defense session.

I would like to take the opportunity to thank my colleagues from the Telecommunication Service Engineering lab for all their advices, ideas, and helps.

I am thankful to Dr. Roch Glitho and Concordia University for their financial support.

Lastly but not least, special thanks to my dearest family and friends who supported me during the completion of my degree.

Table of Contents

1 Introduction 1

1.1 Definitions 1

1.1.1 Conferencing 1

1.1.2 Cloud Computing 2

1.1.3 Cloud Conferencing Business Model 2

1.1.4 Virtualization 3

1.2 Research Domain 3

1.3 Motivations and Problem Statement 4

1.4 Thesis Contributions 5

1.5 Thesis Organization 5

2 Background 7

2.1 Multimedia Conferencing 7

2.1.1 An Introduction to Conferencing 7

2.1.2 Architectural Components of Conference 8

2.1.3 Different Types of Conferences 9

2.2 Cloud Computing 10

2.2.1 Definition of Cloud Computing 10

2.2.2 Key Benefits of Cloud Computing 10

2.2.3 Facets of Cloud Computing 12

2.2.3.1 Infrastructure as a Service (IaaS) 13

2.2.3.2 Platform as a Service (PaaS) 13

2.2.3.3 Software as a Service (SaaS) 13

2.3 Cloud-based Conferencing Business Model 14

2.4 Virtualization 15

2.4.2 Hypervisor 18

2.4.3 Virtualization Techniques 19

2.4.3.1 Full Virtualization 19

2.4.3.2 Paravirtualization (OS assisted virtualization) 20

2.4.3.3 Hardware Assisted Virtualization 21

2.4.5 Benefits of Virtualization 21

2.4.6 Critix Xen Server 22

2.5 Chapter Summary 23

3 Scenarios, Requirements and State of the Art Evaluation 25

3.1 Scenarios 25

3.2 Requirements 29

3.3 State of the Art Evaluation 31

3.3.1 Standard Conferencing Frameworks 31

3.3.2 Existing Cloud-based Conferencing Services 35
3.3.3 Description, Publication and Discovery of Cloud Conferencing Services . 42
3.4 Chapter Summary 46
4 Proposed Architecture 47
4.1 Overall Architecture 47
4.1.1 Architectural Components 49
4.1.1.1 Broker 49
4.1.1.2 IaaS Layer Components 51
4.1.1.3 SubaaS Layer Components 52
4.1.2 Communication Interfaces 53
4.2 Illustrative Scenario 61
4.2.1 Service Creation 61
4.2.2 Service Activation 62
4.2.3 Service Execution 63
4.2.3.1 Create Conference Execution Process 64
4.2.3.2 Add Participant to the Conference Execution Process 65
4.3 How the Requirements are Met by the Architecture 67
4.4 Chapter Summary 68
5 Resource Allocation Mechanism 70
5.1 Problem Statement 70

5.2 Existing Scaling Approaches 72

5.2.1 Static Resource Allocation 72

5.2.2 VM-level Elasticity 73

5.2.3 Fine Grained Resource-level Elasticity 74

5.3 Proposed Scaling Mechanism 79

5.3.1 Discussion on a Scaling Approach for Cloud Conferencing Applications 80

5.3.2 Description of the Proposed Scaling Mechanism 80

5.3.2.1 Assumptions 83

5.3.2.2 Scaling Mechanism Step by Step 85

5.4 Chapter Summary 89

6 Validation: Prototype and Evaluation 91

6.1 Software Architecture 91

6.1.1 Overall Software Architecture 91

6.1.1.1 IaaS Layer Software Components 92

6.1.1.2 SubaaS Layer Software Components 94

6.1.1.3 Operatinal Procedures 96

6.1.1.4 Communication Interfaces 97

6.2 Prototype Design and Architecture 98

6.2.1 Implemented Scenario 99

6.2.2 Prototype High Level Description 106

6.2.2.1 Scope 106
6.2.2.2 Assumptions 107
6.2.2.3 Design Decisions 108
6.2.3 Prototype Architecture 110
6.2.3.1 Implementation scope of prototype 110
6.2.3.2 Detailed Prototype Architecture 111
6.2.3.3 Software Tools 113
6.2.3.3.1 Java 114
6.2.3.3.2 NetBeans 114
6.2.3.3.3 Jersey 114
6.2.3.3.4 Xen Server 114
6.2.3.3.5 Medooze Multi Conference Server 115
6.2.3.3.6 Medooze Media Mixer 116
6.2.3.3.7 Glassfish Application Server 116
6.2.3.3.8 SailFin Application Server 116
6.2.3.3.9 MySQL Server 117
6.2.3.3.10 X-Lite 117
6.2.3.4 Procedures 117
6.2.3.5 Environmental Setting 120
6.3 Performance Evaluation 121
6.3.1 Performance Metrics 121
6.3.1.1 Substrate Activation Measurements 121
6.3.1.2 Create Conference Request Measurements 122
6.3.1.3 Add Participant Request Measurements 125

6.4 Chapter Summary 127

7 Conclusion and Future Work 128

7.1 Contribution Summary 128

7.2 Future Work 129

List of Figures

Figure2-1. Conference Architecture 8

Figure 2-2. Cloud Computing Service Levels 12

Figure 2-3. Cloud Conference Business Model 14

Figure 2-4. Different Server Models 17

Figure 2-5. Different types of Hypervisors 18

Figure 2-6. Virtualization Architectures 19

Figure 3-1. Illustrative Scenario Showing Usage of Cloud Conferencing IaaS 27

Figure 3-2. Conference Server 32

Figure 3-3. Conferencing System Logical Decomposition 33

Figure 3-4. High level Architecture of WebEx 36

Figure 3-5. Conference Manager Architecture 39

Figure 3-6. Cloud conferencing model 40

Figure 3-7. SOA Layers of Cloud Conferencing 42

Figure 3-8. General Broker Architecture for Publication and Discovery 43

Figure 4-1. Architecture of Cloud-based Conferencing IaaS 48

Figure 4-2. End-To-End Publication and Discovery Steps..... 51

Figure 4-3 .Service Activation Sequence..... 63

Figure 4-4. Create Conference Execution Sequence 65

Figure 4-5. Add Participant Execution Sequence..... 66

Figure 5-1. Resource Allocation Procedure for scaling..... 88

Figure 6-1. Software Architecture 91

Figure 6-2. Service Activation Procedure..... 96

Figure 6-3. Service ExecutionPprocedure (creating a conference) 97

Figure 6-4. Service Creation GUI by PaaS..... 100

Figure 6-5. Service Activation GUI by PaaS..... 101

Figure 6-6. Conference Application GUI- Initial Screen..... 103

Figure 6-7. Conference Application GUI- View Registered End Users..... 104

Figure 6-8. Conference Application GUI- Create Conference 105

Figure 6-9. Conference Application GUI- Add Participant..... 106

Figure 6-10. A SIP client (X-Lite softphone) 109

Figure 6-11. Prototype Architecture 113

Figure 6-12 . Procedure of Activation of Dial-out Audio Conference Application 118

Figure 6-13. Create Conference Execution Procedure 119

Figure 6-14. Add participant Execution Procedure 119

Figure 6-15. Non-cloud Distributed Conferencing Model 123

Figure 6-16. Create conference Delay in Cloud and Non-cloud Distributed Models 125

Figure 6-17. Add Participant Delay in Cloud and Non-cloud Distributed Models 126

List of Tables

Table 3-1.State of the Art Evaluation 45

Table 4-1. RESTful Pi interface for Dial-out Audio Conference 57

Table 4-2. RESTful Si interface of a Dial-out signaling (Substrate Provider A) 59

Table 4-3. RESTful Si interface of an Audio Mixer (Substrate Provider B)..... 60

Table 6-1. Substrate Activation Time Delay 122

Table 6-2. Measurement Result of Create Conference Request 124

List of Acronyms and Abbreviations

NIST	National Institute of Standards and Technology
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
SIP	Session Initiation Protocol
QoS	Quality of Service
SLA	Service Level Agreement
OS	Operating System
EC2	Elastic Compute Cloud
VMM	Virtual Machine Monitor
SIPPING	Session Initiation Proposal Investigation
APIs	Application Programming Interfaces
IMS	IP Multimedia Subsystem
MCU	Multipoint Conferencing Unit
SOA	Service Oriented Architecture
REST	Representational State Transfer
VM	Virtual Machine
PM	Physical Machine
OCDA	Open Data Center Alliance
SLO	Service Level Objectives
GUI	Graphical User Interface

RTP	Real-time Transport Protocol
SDP	Session Description Protocol
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
UDP	User Datagram Protocol
VOIP	Voice over IP
RPC	Remote Procedure Call
NIC	Network Interface Card
LSU	Lightweight Scaling Up
JSON	JavaScript Object Notation
LSD	Lightweight Scaling Down
XML	Extensible Markup Language

Chapter 1

1 Introduction

In this chapter, first we provide definitions for the key concepts related to our research and then we define the research domain. Then, we discuss the motivation, the problem statement, and the contributions of this thesis. Finally, we describe how this thesis is organized.

1.1 Definitions

There are four key concepts related to our research on a virtualized infrastructure for multimedia conferencing applications: conferencing, cloud computing, cloud conferencing business model and virtualization. We discuss these concepts in this section before explaining the research domain in the next section.

1.1.1 Conferencing

Conferencing is the real-time exchange of media (voice, video, and text) which enables collaboration between multiple conference participants [1]. Conferencing is the basis of a plethora of multimedia applications (e.g. audio/video conference, multiparty games and distance learning applications). These applications still face challenges such as scalability and elasticity, although they are ubiquitous nowadays.

1.1.2 Cloud Computing

Clouds are a large pool of virtualized resources which can be dynamically reconfigured to adjust to a variable load (scale), allowing optimum resource utilization [2]. According to National Institute of Standards and Technology (NIST) [3] cloud computing is an emerging and transformational paradigm for provisioning of network, storage, and computing on demand as a commodity using a pay-per-use model. According to one of the widely-accepted definitions of cloud computing [1], it encompasses three key facets: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS provides the infrastructure for computing, storage and networking using virtualized hardware resources. PaaS provides the software environments to design, develop and deploy applications. Service providers use platforms offered as PaaS by platform providers to develop and manage applications. The applications are provisioned to end-users (or other applications) as SaaS on a pay-per-use basis. Cloud platforms add levels of abstraction to the infrastructures offered as IaaS by infrastructure providers. PaaS eases the application development and management. IaaS is the actual dynamic pool of virtualized resources used by applications and ensures scalability, elasticity and efficiency in resource usage.

1.1.3 Cloud Conferencing Business Model

Despite the benefits that cloud computing may bring to the conferencing applications, to the best of our knowledge, there are currently no full-fledged environments that allow the development, deployment and management of cloud-based conferencing applications [1]. A business model was proposed for cloud-based conferencing [1].

The following are the roles in the proposed business model: connectivity provider, broker, conferencing substrate provider, conferencing infrastructure provider, conferencing platform provider, and conferencing service provider. Conference substrates (e.g. dial-out signaling, audio mixing) are sharable fine-grained building blocks of conferencing provided by the conference substrate provider that can be virtualized and shared between conferencing applications for resource efficiency purposes. These virtualized conference substrates can be composed to create full-fledged conferencing applications (e.g. dial-out audio conference).

1.1.4 Virtualization

Virtualization is the basic technology of clouds with characteristics such as dynamic assignment of resources and resources sharing [2, 4]. It brings key benefits such as resource efficiency through sharing of physical resources. Virtualization provides virtual resources instead of providing the actual ones [4]. By virtualization several operating systems can exist on the same hard ware server and several applications can coexist on the same operating system. Various virtualization platforms s are available such as Xen [5], VMware [6], etc. Virtalization technology can be used to virtualize conferencing substrates as building blocks of conferencing applications for resource efficiency, flexibility and scalability purposes.

1.2 Research Domain

Conferencing is the basis of a plethora of several enterprise applications (e.g. multimedia conferencing applications, multiparty games and distance learning). Various types of conferencing applications with different features can be offered.

Some examples are: dial-out audio signalling conference, dial-out video signalling conference, dial-in video conference service with floor control. These conferences can be built from composing several fine-grained conferencing substrates.

In this thesis we focus on a virtualized infrastructure which offers an on-demand dynamic provisioning of the conferencing substrates. The infrastructure relies on fined grained conferencing substrates. Conferencing substrate examples are dial-out signaling, dial-in signaling, audio mixer and video mixer, floor control, transcoder, etc. Infrastructure may offer atomic fine-grained substrates; also it may assemble several conferencing substrates on the fly to build conferencing services. Infrastructure provider relies on the conferencing substrates offered by third party substrate providers. Infrastructure provider may also offer its own conferencing substrates. Such an infrastructure will scale in fined grained manner to the demand of the users by generating multiple instances from conferencing substrates and running them simultaneously. Infrastructure will also be elastic in a fined grained manner.

1.3 Motivations and Problem Statement

Cloud-based conferencing services inherent significant benefits of cloud computing such as resource efficiency, scalability and easy introduction of different types of services. Several conferencing applications are offered today in cloud settings. However, to the best of our knowledge, there is still no cloud conferencing environment available that enables management and offering of a wide range of conferencing resources to deploy conferencing applications on this infrastructure [1].

Providing a virtualized conferencing infrastructure is critical for the realization of cloud-based conferencing. An architecture for this infrastructure needs to be designed

which offers fine grained conferencing substrates. The fine grained conferencing substrates can be hosted by infrastructure provider or offered by third party conference substrate providers. An infrastructure which has the capability to assemble these substrates on the fly to build a conferencing service, scale to the user demand and manage the conferencing resources in an elastic resource efficient way.

1.4 Thesis Contributions

The thesis contributions are as follows:

- Requirements for a virtualized infrastructure for multimedia conferencing applications based on the proposed cloud conferencing business model.
- Analysis of the state of the art with an evaluation summary based on our requirements.
- An overall architecture for a virtualized infrastructure for multimedia conferencing.
- Proposing a fine-grained scaling mechanism for conferencing substrates
- Implementation architecture, a proof of concept prototype, and performance evaluation.

1.5 Thesis Organization

The rest of this thesis is organized in 6 following chapters:

Chapter 2 presents the background concepts and definitions related to our research domain.

Chapter 3 presents the requirements for a cloud-based conferencing infrastructure and evaluates the state of the art based on the requirements.

Chapter 4 presents the proposed architecture for a virtualized infrastructure in cloud-based conferencing. It describes the architectural components, as well as the interactions between different cloud layers while enabling a conferencing service.

Chapter 5 reviews different scaling approaches and proposes a fine-grained scaling mechanism for conferencing substrates.

Chapter 6 describes the software architecture, the implemented prototype as the proof-of-concept. Also, it discusses the performance measurements collected to evaluate the architecture.

Chapter 7 concludes the thesis by giving a summary of the overall contributions, and future research directions.

Chapter 2

2 Background

This chapter describes background concepts related to the research domain. The following concepts are explained: multimedia conferencing, cloud computing, cloud-based conferencing business model, and also virtualization, which is the technology we use to provide the conferencing infrastructure and to ensure efficiency in resource usage. The cloud-based conferencing business business model is introduced because our proposed architecture relies on it. At the end, we summarise the chapter.

2.1 Multimedia Conferencing

In this section, we give an introduction to multimedia conferencing, architectural components of a conference, and different types of conferences.

2.1.1 An Introduction to Conferencing

Conferencing is the real-time multi-party exchange of media (voice, video, and text) which enables real-time collaboration among the participants of the conference. Conferencing plays a key role in several enterprise applications like conferencing application, gaming applications, distance learning applications, and social networking applications. Conference services are real-time resource-intensive services which may have huge number of users involved.

2.1.2 Architectural Components of Conference

Architectural components of a typical conference are shown in Figure2-1. Conference Architecture [1] : Signalling, media handling and conference control.

Figure2-1. Conference Architecture

- **Signaling:** This component handles signalling operations such as session set-up, session establishment, and session tear down using a signaling protocol (e.g. Session Initiation Protocol (SIP)).
- **Media Handling**: This component handles media aspects such as receiving, mixing, transmitting and transcoding of media. The mixer role is to receive and combine multiple input audio/video streams to output streams. The mixer should generate the output streams in a way that each participant in a conference will be receiving the mix of all streams from all participants excluding the participant itself. Transcoding is encoding and decoding the media between different media formats.
- **Conference Control:** Conference control gives additional functionalities to conferences such as floor control and policy control [1].

    - **Policy Control:** This component handles conference policy aspects such as access and admission control.

- **Floor Control:** This component allows users of multimedia applications to utilize and share resources such as continuous media (video and audio) without access conflicts. Floors are temporary permissions given to users dynamically in order to mitigate race conditions and guarantee mutually exclusive resource usage [18].

Signaling and media handling are necessary components for any conferencing application and conference control is an optional component for additional control and capabilities.

2.1.3 Different Types of Conferences

There are three major types of conferencing based on standard specifications [19]:

- **Tightly-coupled Conference**: A conference is an association of conference participants with a central point (i.e., a conference focus), where the focus has direct peer-wise relationships with all participants by maintaining a separate dialog with each of them [20].
- **Loosely-coupled Conference:** In the loosely coupled model, there is no coordinated signaling relationship amongst participants of the conference (no central point of control or a conference server).
- **Fully-distributed Conference:** In this model, each participant maintains a signaling relationship with other participants of the conference.

2.2 Cloud Computing

In this section we define what is cloud computing, what are the key benefits of cloud computing and what are its main facets.

2.2.1 Definition of Cloud Computing

Different definitions are available for clouds [7]. Vaquero et al. has proposed an integrative definition for cloud computing based on the study done on more than 20 previous definitions. The definition is that "clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized Service Level Agreements (SLAs)" [8: 51]. Although this definition includes main characteristics of cloud computing; resource pooling, rapid elasticity and measured services, but it fails to mention two other characteristic of cloud computing which are on-demand self-service (computing resources could be used at any time without any human interaction with infrastructure service provider) and broad network access network (having access to computing resources over network). NIST (US National Institute of Standards and Technology) definition of cloud computing covers all of these five characteristics [9].

2.2.2 Key Benefits of Cloud Computing

Cloud computing has several benefits. Some of its key benefits include:

- **Scalability**: Cloud provider's massive capacity offers an unlimited scalability [10]. “When you are tied into a cloud computing system, you have the power of the entire cloud at your disposal” [7:26]. Cloud computing manages your needs (storage, computing, networking) and scales on demand.
- **Elasticity:** is the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner [57].
- **Resource efficiency**: is scaling in such a way that at each point of time the available resources match the current demand as closely as possible [7].
- **Reliability**: when users move to cloud, they have a certain level of expectations about the Quality of Service (QoS) in clouds, including availability, performance and fault tolerance. To address user’s concerns, cloud service providers offer a warranty by SLAs with customers. In these agreements, details of the service to be provided and also the penalty for violations are specified [11].
- **Resource pooling**: is ability of cloud to serve multiple customers by assigning and reassigning the virtualized and physical resources dynamically according to demand. It is actually the sharing of resources which leads to optimum resource utilization and cost [7].
- **Easier management and maintenance:** as the service is hosted somewhere else, there is no need to maintenance or support [7]. Cloud services are web based and users do not need to upgrade, as the software is not locally saved on their systems.
- **Cost and pricing**: cloud customers pay on a pay-per-use model which decreases the total cost efficiently.

2.2.3 Facets of Cloud Computing

Cloud Computing consists of three main facets [4]:

- **Infrastructure as a Service (IaaS):** delivering cloud computing infrastructure: servers, storage, network and operating.
- **Platform as a Service (PaaS):** providing software environments and tools to create, develop and deploy applications and services.
- **Software as a Service (SaaS):** providing users the access to software applications and services, priced on a pay-per-use basis.

Figure 2-2 shows the cloud computing architecture with three facets.

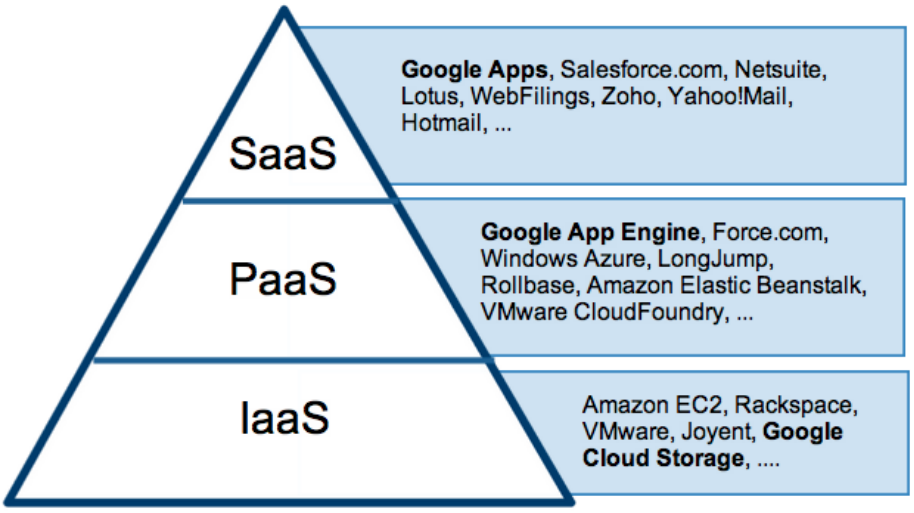

Figure 2-2. Cloud Computing Service Levels [12]

In the following sections we will describe each of these facets in more detail.

2.2.3.1Infrastructure as a Service (IaaS)

Infrastructure as a service is provisioning of computing recourses such as processors, storages and networks. Despite the traditional resource provisioning scenario, enterprises do not need to manage and maintain the resources which are located somewhere else on the cloud. Using virtualization technologies, IaaS enables co-existence of several heterogeneous resources on the same hardware system. By sharing resources, it provides resource and cost efficiency. Some known samples of cloud infrastructures are Amazon's Elastic Compute Cloud (EC2) and Google Compute Engine.

2.2.3.2 Platform as a Service (PaaS)

Platform as a Service adds a layer of abstraction to the cloud infrastructure and provides platforms and Application Programming Interfaces (APIs) in cloud settings for use of service providers and developers to implement and deploy the applications and services [13]. PaaS examples are Google App Engine and Windows Azure App Fabric.

2.2.3.3Software as a Service (SaaS)

By SaaS, service providers provide applications for the access of end users or other third party applications (via APIs). There is no need to buy, install and run the software on your computer. End users just need a browser and an internet connection to use the cloud based software on a pay-per-use basis [13]. Examples of SaaS are Salesforce.com and Google docs.

2.3 Cloud-based Conferencing Business Model

There is a cloud-based conferencing business model recently proposed for provisioning conference applications in the cloud setting [1]. This model brings benefit of cloud computing such as easy introduction of different types of conferences, resource efficiency and scalability to the conferencing. We use this model as the basis for our architecture. Figure 2-3 shows this business model.

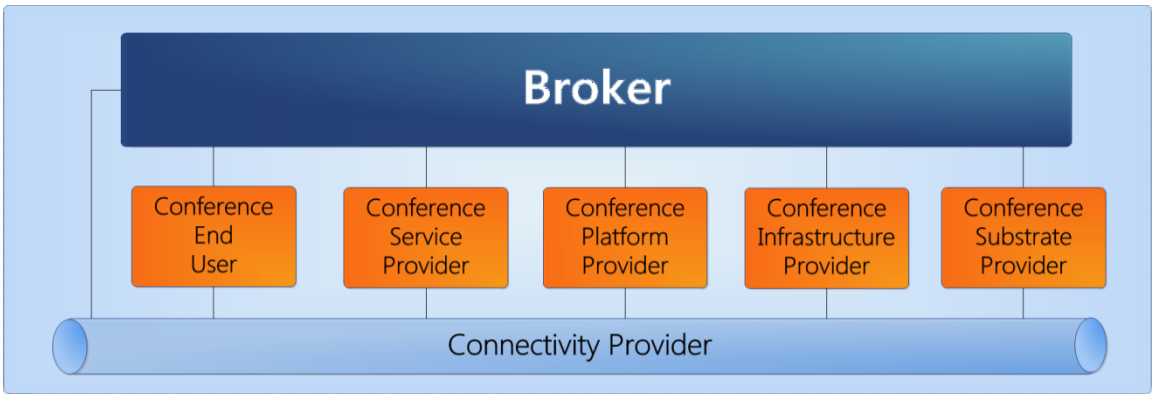

Figure 2-3. Cloud Conference Business Model

The key actors in this business model are Broker, Connectivity Provider, Conference End User, Conference Service Provider, Conference Platform Provider, Conference Infrastructure Provider and Conference Substrate Provider.

- **Connectivity Provider:** The connectivity provider is the communication channel for interactions between the different actors.
- **Broker:** The broker provides a publication and discovery mechanism for substrates and conference applications in different levels of cloud.
- **Conference Substrate Provider**: The substrate provider provides virtualized fine-grained conference substrates which are considered the main building blocks of conference applications.

- **Conference Infrastructure Provider**: The infrastructure provider may also provide conference substrates like conference substrate provider. Infrastructure provider discovers the substrates published by the substrate provider from the broker. And then publishes the substrates again to be discovered by the conference platform provider. It may publish the atomic conference substrates as they are or it may alternatively, compose them and then publish the composed substrate to the broker.
- **Conference Platform Provider**: The platform provider provides a set of tools for service providers to create conference applications. They are software frameworks for creation, composition and execution of conference applications. Conference platform provider discovers the substrates published by the infrastructure provider from the broker.
- **Conference Service Provider:** The service provider uses the platform tools provided by the platform provider to create conference applications. After creation of the conference application, publishes it to the broker to be discoverable by conference end users.
- **Conference End User:** The conference end user is who discovers the conference application from the broker and uses it on a pay-per-use model.

2.4 Virtualization

Virtualization in the main enabling technology for cloud computing. In this section, first we give a brief introduction of virtualization, and then we explain what a hypervisor in a virtualized environment is. After that, we discuss different virtualization techniques. Then, we mention the benefits of the virtualization and

finally we discuss one of the virtualization technologies (Xen) which is used in our prototype.

2.4.1 Definition of Virtualization

The term virtualization describes the separation of a resource from the underlying physical delivery of that service [26]. For example, By hardware virtualization, you can create a virtual machine that acts like a real computer with an Operating System (OS) which is separated from the underlying hardware resources, or with virtual memory computer software gains access to more memory than is physically installed [26]. The actual machine that virtualization takes place on, is the host and the virtual machine is the guest. The software which creates a virtual machine on the host machine is called a hypervisor or Virtual Machine Monitor. Several guest machines may co-exist on the same hardware server [21]. The concept of virtualization is very broad and can be applied to other IT infrastructure layers including devices, servers, networks, storage, hardware, operating systems and applications [23].

In a traditional physical computer, one instance of the operating system supports one or more application programs. In a virtualization environment, a single physical computer runs multiple virtual computers. Each of virtual machines may be running a different operating system from all of the other virtual machines on the physical machine (Figure 2-4).

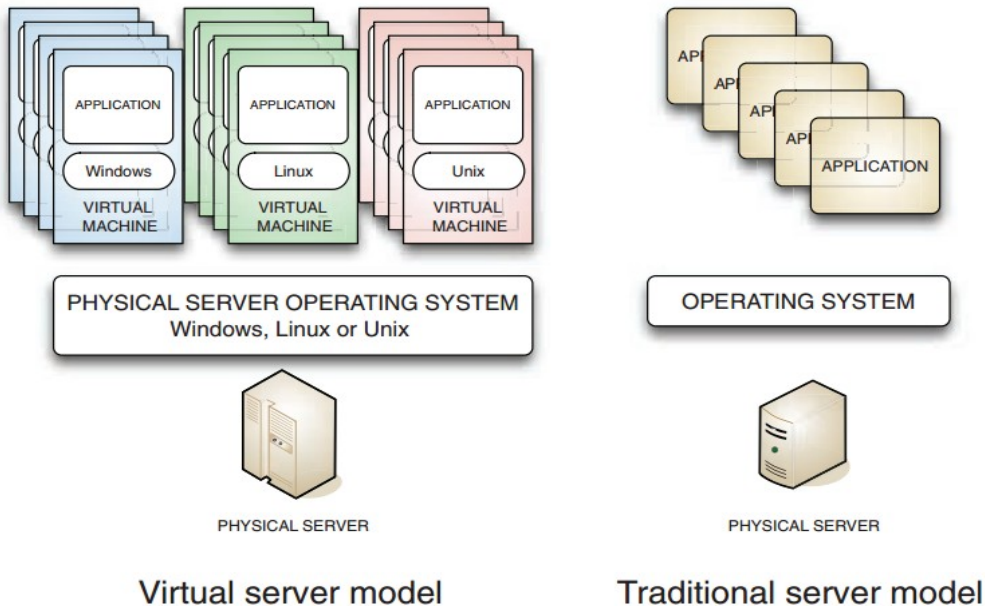

Figure 2-4. Different Server Models [23]

There are key differences between virtual and traditional server models [26]. In the traditional server model:

- There is single OS image per machine.
- Software and hardware are tightly coupled.
- Running multiple applications on same machine often creates conflict.
- Resources are underutilized.
- The infrastructure is inflexible and costly.

And in the virtual server model:

- Hardware is independent of operating system and the applications.
- Virtual machines can be provisioned to any system.
- It can manage OS and application as a single unit by encapsulating them into virtual machines.

Examples of virtualization technologies are Critix Xen Server, VMware Workstation, VirtualBox, etc.

2.4.2 Hypervisor

Hypervisor or Virtual Machine Monitor (VMM) is a piece of software, firmware or hardware that creates, runs and controls the virtual machines. Hypervisor runs on the host machine and manages the execution of guest operating systems. Guest machines share the virtualized hardware resources.

There are two types of hypervisors [22] which are depicted in Figure 2-5:

- **Type 1 hypervisors or native (bare metal) hypervisors**: they run directly on the host's hardware to control the hardware and guest operating systems. Guest operating systems run above the hypervisor in another level. Examples for this type of hypervisors are Citrix Xen Server and VMware ESX.

- **Type 2 hypervisors or hosted hypervisors**: They are within a distinct second software level, and the guest operating systems run at the third level above the hardware. Examples for this type of hypervisors are VMware Workstation (Figure 2-6) and VirtualBox.

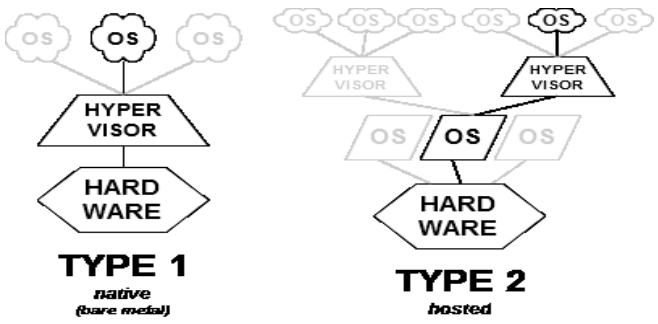

Figure 2-5. Different types of Hypervisors [32]

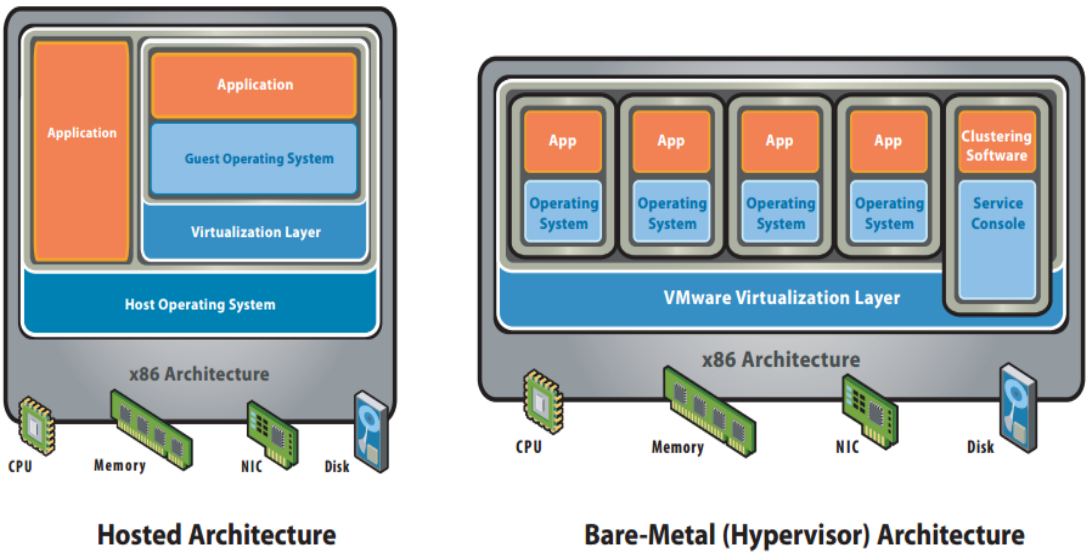

Figure 2-6. Virtualization Architectures [26]

2.4.3 Virtualization Techniques

There are three alternatives to provide a certain kind of virtual machine environment [27]. They are as follows:

- Full virtualization
- OS assisted virtualization or paravirtualization
- Hardware assisted virtualization

We explain each of these techniques in the following sub-sections.

2.4.3.1 Full Virtualization

In this technique, each virtual machine is provided with all the service of the physical system and the complete underlying hardware is simulated. In a full virtualized environment, any software capable of execution on the raw hardware can be run in the

virtual machine or the operating system. In full Virtualization the guest OS is fully abstracted (completely decoupled) from the underlying hardware by the virtualization layer [27]. Operating systems and the applications run on the same architecture as the underlying physical machine. It is also called native virtualization [33]. The guest OS is not aware of being virtualized and does not require any modification. It requires no hardware assist to virtualize the instructions [27].

Full virtualization has the best isolation and security for virtual machines. It simplifies the migration and portability because the same guest OS instance can run virtualized or on native hardware. Examples of full virtualization products are VMware's virtualization products and Microsoft Virtual Server [27].

2.4.3.2 Paravirtualization (OS assisted virtualization)

"Para-" is an English affix of Greek origin that means "beside". Paravirtualization means "alongside virtualization" which refers to communication between the guest OS and the hypervisor to improve performance and efficiency. In paravirtualization the OS kernel is modified to replace nonvirtualizable instructions with hypercalls. It also provides hypercall interfaces for other critical kernel operations such as memory management and interrupt handling [27].

In paravirtualization, the hypervisor exports a modified version of the underlying hardware and usually small changes are required in the guest operating system [33].

The performance advantage of paravirtualization over full virtualization varies depending on the workload. The compatibility and portability of paravirtualization is poor because it cannot support unmodified operating systems (e.g. Windows XP).

Paravirtualization may cause maintainability issues as it needs deep OS kernel modifications. The open source Xen project (using a modified Linux kernel) is an example for paravirtualization [27].

Major advantages of paravirtualization are improved performance, scalability and manageability [33].

2.4.3.3 Hardware Assisted Virtualization

In hardware-assisted virtualization, the virtual machine simulates the hardware needed to run unmodified guest operating systems for completely different hardware architectures and the operating systems are executed unmodified [33]. It enables efficient full virtualization. It is also known as accelerated virtualization. By using hardware capabilities, privileged and sensitive calls are set to automatically trap to the hypervisor which removes the need for either binary translation or paravirtualization. The guest state is stored in Virtual Machine Control Structures or Virtual Machine Control Blocks. Processors with these hardware assist features (Intel VT and AMD-V) became available in 2006 and only newer systems contain these features [27].

2.4.5 Benefits of Virtualization

Virtualization has many common uses and benefits. They include:

- By virtualization, it is possible to run multiple virtual machines on a single physical hardware simultaneously. It maximizes the utilization by decoupling the physical hardware from the operating system [23].

- It promises flexible resource provisioning and management, and also cost effective outsourcing of applications, platforms and hardware resources [23]. Hardware is fully utilized when multiple operating systems coexist on a single physical machine [33].
- It enables creation of dynamic virtual infrastructures on demand, performance isolation and fast deployment of services (developing a standard virtual server that can be easily duplicated to speed up server deployment).
- It has the ability to perform a recovery of virtual machines by taking snapshot from a stable guest image [33]. In case any virtual machine crashes, all of the other virtual machines will be left unaffected [23]. To recover a virtual machine from an attack, it can be simply rolled back to the trusted saved point [33].
- Virtual machine monitors provide a uniform interface to hardware which shields the guest systems from the lower level physical computing resources. So it provides portability and guest systems can be moved from one physical machine to another without interruption [33].
- Hypervisors ease the work for developers. They no longer need to restart physical machines to switch between different operating systems [33].
- It has the benefit of decreased power consumption and cooling infrastructure by making more efficient use of power [33].

2.4.6 Critix Xen Server

Xen Server is an open source virtualization solution for managing cloud server and desktop virtual infrastructures. Xen Server includes a Xen Hypervisor and APIs for

the control and integrations. It provides built-in support and templates for Windows and Linux guest operating systems. It provides management functionalities for control of Virtual Machines (VMs): creation, starting, taking snapshot, shutting down, resizing, networking setting, deletion, etc. during their lifecycle [24]. Xen Server automates management processes, increases IT flexibility and agility and lowers the costs [24].

Xen hypervisor is between the guest domains and the physical hardware. It allocates the resources to the guest domains and provides protection and isolation among them. It is above the physical hardware and gives a virtual hardware interface to each guest domain. It gives a portion of underlying physical hardware to each guest domain and shares the resources. It allocates a limited amount of memory and a share of CPU [24].

A privileged domain called Domain0 serves the administrative interface role for the Xen. It actually implements some logical functions of the hypervisor to administrate the guest domains. When the system boots, this is the first domain that gets lunched and is used to create and configure the other guest domains. If there are domains that are previously created, domain0 can start them automatically by consulting the configuration files [24].

2.5 Chapter Summary

In this chapter we discussed the background concepts that are related to this thesis. First, we introduced the concept of multimedia conferencing, its key technical components and typical types of conference applications. Then we explained cloud

computing, its benefits and its key facets. Then we explained a cloud-based conferencing business model recently proposed for cloud-based multimedia conferencing which we have used as the basis of our architecture.

We followed by discussing the virtualization concept explaining Virtual Machine Monitor, different virtualization techniques and the benefits of the virtualization. At the end, we discussed Xen which is the virtualization technology that we have used in our prototype.

Chapter 3

3 Scenarios, Requirements and State of the Art Evaluation

This chapter includes three sections. First, we present the scenarios that illustrate the use of conferencing infrastructure in a cloud-based conference setting. These scenarios are based on the cloud conferencing business model [1] we discussed in chapter 2. Then, based on the scenarios we derive the requirements for such cloud-based conferencing infrastructure. Next, we evaluate the state of the art based on the requirements and finally, we summarize the chapter.

3.1 Scenarios

In this section we present the scenarios based on the interacting roles in the cloud-based conferencing business model. We describe a scenario which illustrates the usage of cloud conferencing infrastructure based on cloud conferencing business model. There are different actors in the cloud-based conferencing scenario: Conference Application /Service Provider, Conference End Users, Conference Platform Provider, Conference Infrastructure Provider and Conference Substrate Providers.

Let us assume that there is a cloud conferencing infrastructure provided by a conferencing infrastructure provider and there are three conferencing application/service providers which use the same cloud conferencing infrastructure:

- An audio/video conference application/service provider A,
- A multiparty game application/service provider B,
- A distance learning application/service provider C

Let us also assume that there is a conferencing platform provider which offers Conferencing PaaS.

The application/service providers use the conferencing PaaS to create their services/ applications (i.e. audio/video conferencing applications, multiparty games, distance learning applications) or to include conferencing capabilities in their applications/services. Let us now assume that application/service provider A has created and offers a dial-out audio conference application, application/service provider B has created and offers a multiparty game application with dial-in audio conference with floor control capability, and application/service provider C has created and offers a distance learning application with dial-in video conferencing capability. These applications may have some constraints (e.g. QoS requirements such as performance and availability, number of conference end-users supported, cost of service, etc.). These service providers may also offer other type of applications/services. The number of these applications/services may also vary over time.

In our envisioned architecture all these applications rely on sharable substrates (fined grained building blocks) offered by the conferencing IaaS provider. This IaaS may also rely on third party substrate providers as stipulated in reference [1]. Figure 3-1 depicts the scenario described and illustrates the usage of cloud conferencing infrastructure.

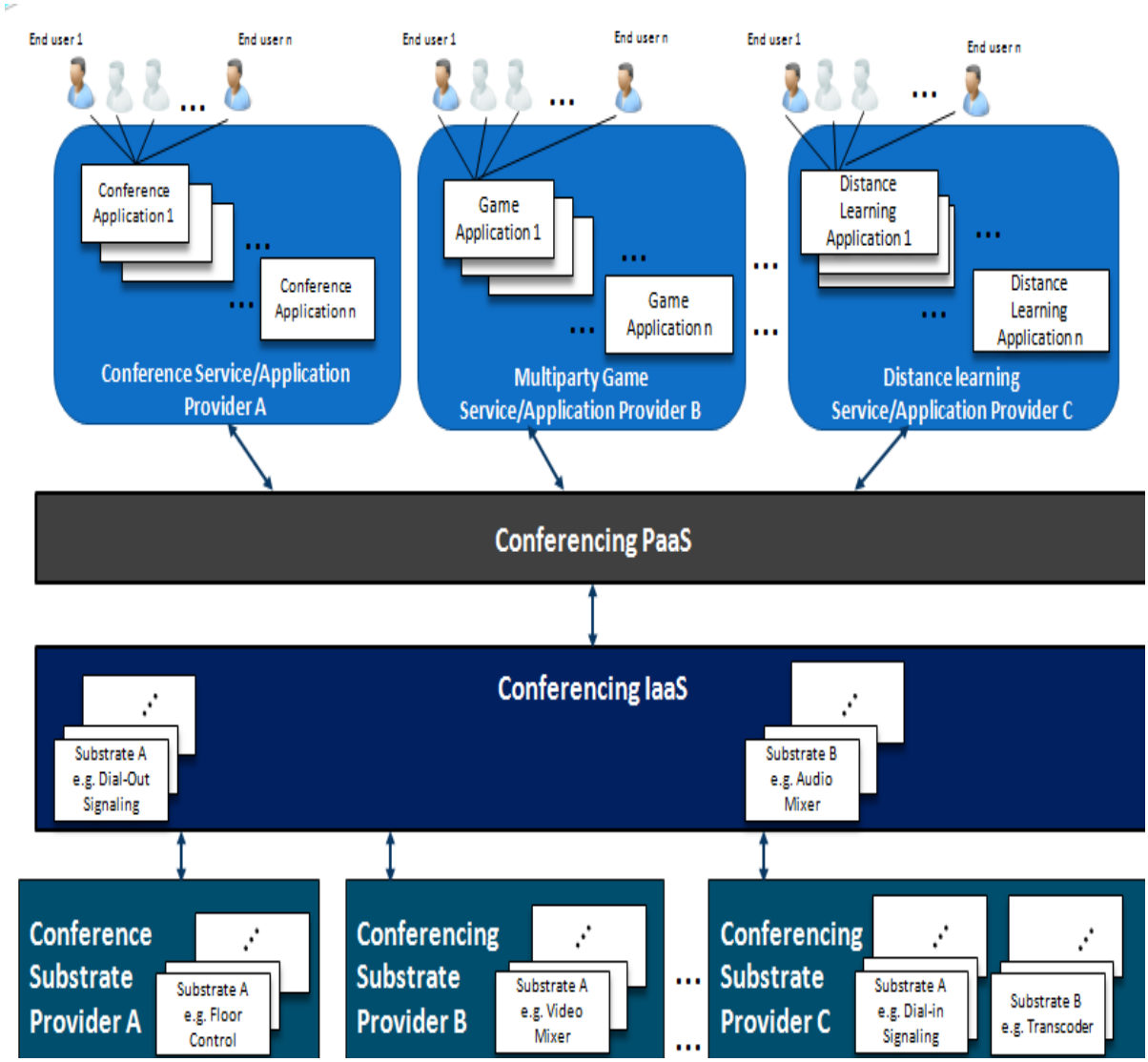

Figure 3-1. Illustrative Scenario Showing Usage of Cloud Conferencing IaaS

Service Providers use conferencing PaaS to create conference applications/services. They specify the type of conference application to be created (e.g. dial-out audio signaling conference) along with the constraints that need to be taken into account (e.g. QoS requirements, number of users supported, cost of service, etc.).

Later, service providers use the conferencing PaaS to activate the conference application/ service created (creation and activation of a service may have different timings). Conferencing PaaS has access to the conferencing IaaS and asks

conferencing IaaS (which has access to the actual pool of conferencing resources) for the activation of the conferencing substrates involved in the conference application/service created, giving the information of the substrates required and the constraints. For example, to have a dial-out audio signaling conference application which supports up to 100 users, a dial-out signaling substrate and an audio mixer substrate with the capacity of minimum 100 users are required. Or to have a multiparty game application which supports up to 200 users, a dial-in signaling substrate, an audio mixer substrate and a floor control substrate with the capacity of minimum 200 users are required.

Now assume that for activation of the dial-out audio conference application requested by service provider A, IaaS searches into a substrate broker (which has the substrate information) and discovers that a substrate provider A (a third party substrate provider) is offering a dial-out signaling substrate and a substrate provider B is offering an audio mixer substrate which are capable to meet the constraints specified. Thus, infrastructure provider will ask these substrate providers to instantiate and activate the required substrates for the conference application. Assume that substrate providers activate the substrates and send back the activation result to the conferencing infrastructure provider and then conferencing infrastructure provider sends back the activation result to the conferencing platform provider. After a successful service activation process, conferencing platform provider deploys the conference application. The conference application offered by service provider A is now ready to be used by conference end users. Number of end users may change dynamically over time. New end users may join to the conferences or current end users may leave the conferences.

3.2 Requirements

In this section, we draw out the requirements for provisioning a cloud-based conferencing infrastructure based on the scenario we described in the previous section and the cloud-based conferencing business model.

There are 5 different actors involved in the scenario described in the previous section. The 5 roles include Conferencing Application/Service Provider, Conferencing End User, Conferencing Platform Provider, Conferencing Infrastructure Provider and Conferencing Substrate Provider.

Requirement #1 is that the interfaces between conferencing infrastructure provider and other actors should rely on standard technologies in order to ease interoperability and usage of available standard tools and technologies.

Requirement #2 is that third party substrate providers should be able to make their substrates known to the conferencing infrastructure provider by publishing them to a broker. Conferencing substrate providers publish the conferencing substrate to a broker, providing the information of substrate (e.g. functional features of substrate, QOS parameters, substrate constraints like the number of users supported, cost of substrate usage, etc.). This requirement later enables discovery of published substrates by conferencing infrastructure provider. On the other hand, conference infrastructure provider will publish the discovered substrates to another broker (a different broker level from the level that substrate providers publish their substrates) in order to make them known to conferencing platform provider. This enables their discovery by the platform provider. Infrastructure provider might alter some of the information provided in the substrate descriptions (e.g. provider information, cost) before

publishing them. If infrastructure provider itself has also any substrates to offer, they will be published to this broker as well. Conferencing infrastructure provider may publish the conferencing substrates as they are as atomic substrates or it may compose several substrates and publish the resulted composite substrate, depending on the capabilities the substrates provide. Conferencing infrastructure provider edits the information of substrates which are from third party substrate providers, deletes the substrate provider information and replaces them with the information of infrastructure provider (information like the address of infrastructure provider or the cost model).

Requirement #3 is that conferencing infrastructure provider should be able to perform an expressive discovery and search for the substrates by criteria. E.g. an advances search option by functional features of substrate, QoS requirements, cost, etc. This way infrastructure provider will be able to discover the substrates which fit the best to capabilities and features desired.

Requirement #4 is scalability of conferencing IaaS in terms of number of conferencing end users which use a conferencing applications/service provided by an application/service provider and in terms of the number of applications that can be created by application/service providers. Number of end users may change dynamically over time and conferencing infrastructure should scale to the demand while the number of users are increasing and be responsive to the requests (it is assumed that the scalability requirement is discussed in the SLAs between providers and the requesters of the service).

Requirement #5 is elasticity of conferencing IaaS based on the user demand for resource efficiency purposes. For example, a substrate provider may decide to release some of the resources allocated to a conferencing substrate instance that has a

decreasing demand (e.g. when some end users leave the conference) and allocate the released resources to another substrate instance which has an increasing demand (e.g. when new end users are joining to the conference).

3.3 State of the Art Evaluation

In this section, we discuss the state of the art for conferencing frameworks. We categorize the state of art and review them in 3 sections: existing standard conferencing frameworks which are not cloud-based, existing cloud-based conferencing services and description, publication and discovery of cloud conferencing services.

3.3.1 Standard Conferencing Frameworks

Conferencing has been extensively studied by standard bodies. SIPPING (Session Initiation Proposal Investigation), XCON (centralized conferencing), and DCON (Distributed conferencing) Internet Engineering Task Force (IETF) working groups research on multimedia conferencing frameworks. IETF also defines a floor control protocol [15], and a data model for conferencing [16].

SIPPING defines a centralized coupled model for centralized conferencing. It defines a centralized conferencing framework with Session Initiation Protocol (SIP) [19]. The Session Initiation Protocol (SIP) supports the initiation, modification, and termination of media sessions between user agents by SIP dialogs (which represent a SIP relationship between a pair of user agents). There is a central point of control in which each participant connects to this central point. It follows a tightly coupled

model for centralized conferencing which all the functional parts are integrated in a single conference server.

This conferencing framework is not based on a cloud conferencing model with several different actors of different cloud layers. It is not based on the concept of sharable virtualized substrates as elementary building blocks for conferencing applications.

- As our 2nd and 3rd requirements, it does not provide any publication and discovery mechanism for the conferencing service.
- There is only a single physical conference server which implements the focus, the conference policy server, and the mixer (Figure 3-2). It does not satisfy the scalability and the elasticity requirements which are our 4th and 5th requirements and cannot scale when the number of conference participants keeps increasing [25].

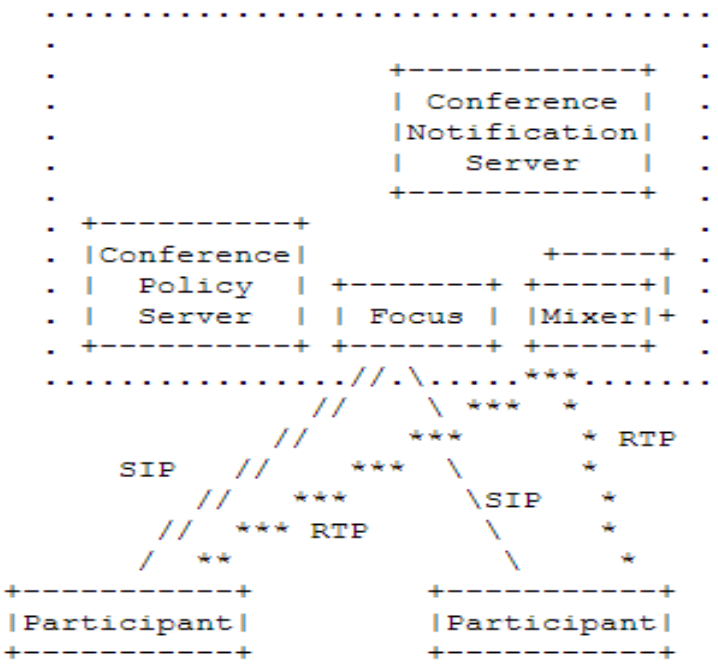

Figure 3-2. Conference Server [19]

XCON defines a complete architecture for centralized conferencing framework [14]. The framework allows participants using various call signaling protocols (e.g. SIP, H.323, Jabber) and exchange media in a centralized unicast conference. The state of a conference is represented by a conference object which has the conference information. The framework also outlines a set of conferencing protocols as complement to the call signaling protocols, for building advanced conferencing applications. This framework binds all the components together for the benefit of builders of conferencing systems in one central focus [14]. Figure 3-3 shows the logical decomposition of this conferencing system.

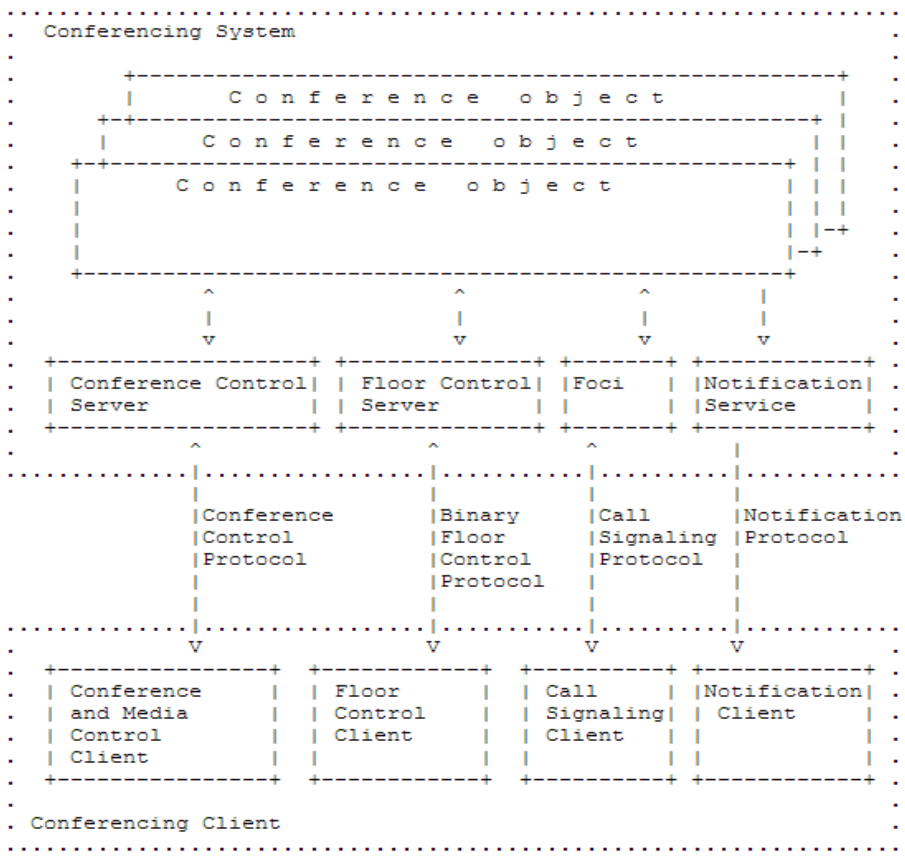

Figure 3-3. Conferencing System Logical Decomposition [14]

Like SIIPING, this conferencing framework is not based on a cloud conferencing model with several different actors as providers of different cloud layers. It is not based on the concept of sharable virtualized substrates as elementary building blocks for conferencing applications.

- As our 2nd and 3rd requirements, it does not provide any publication and discovery mechanism for the conferencing service.
- As our 4th requirement, there is only a single physical conference server and it is not scalable.
- As our 5th requirement, there is also no elasticity mechanism provided.

DCON is a distributed conferencing framework defined by IETF which is inspired from XCON conferencing framework [28]. It implements an overly network from centralized conferencing nodes. A DCON constructs a distributed conferencing system as a federation of centralized conferencing components. Again this conferencing framework is not a cloud based conferencing model with several different actors of different cloud layers. It is not based on the concept of sharable virtualized substrates as elementary building blocks for conferencing applications.

- As our 2nd and 3rd requirements, it does not provide any publication and discovery mechanism for the conferencing service.
- As our 4th requirement, it has improved the scalability in comparison with XCON by orchestrating of a number of XCON frameworks but still it is not totally scalable [28].
- It does not satisfy our elasticity requirement (5th requirement).

3GPP defines a specification for **IP Multimedia Subsystem (IMS)**, providing multimedia services to end users in 3G Networks. The specification provides a centralized architecture [17] and APIs [29] for multimedia conferencing. Centralized architecture uses Multipoint Conferencing Unit (MCU), so as the number of participants increases there is a lack of scalability as MCU acts as a single point of recipient for each participant.

IMS conferencing framework is not also a cloud-based conferencing model with several different actors. It is not based on the concept of sharable virtualized substrates as elementary building blocks for conferencing applications. It does not provide any publication and discovery mechanism for the conferencing service. The resource allocation is static and there is no elastic scaling mechanism provided.

3.3.2 Existing Cloud-based Conferencing Services

There are also some cloud based distributed conferencing products in the market, remarkably Cisco's WebEx and BlueJeans. A few embryonic architectures also have been put forward for conferencing in the cloud [25, 36, 37, 38].

WebEx provides a conferencing service as SaaS [30] which is not based on the cloud conferencing model with different actors e.g. Infrastructure provider. WebEx is only a service provider which offers conferencing as SaaS. It follows a coarse-grained approach and do not provide an infrastructure with fine-grained substrates that can be published, discovered and shared with other applications. It does not meet our 2nd and 3rd requirements.

- As there is no service as Conferencing IaaS to check whether if the interfaces toward IaaS are standard-based or not, our 1st requirement is not applicable.
- It supports a great number of users (up to 3000 participants which maximum of 500 participants only can see live, shared video of presenters or panelists, in video-enabled events [30]). WebEx is not totally scalable [25] and does not meet the scalability requirement which is the 4th requirement. It only supports up to 100 users per conference.
- Information about Webex elasticity features not available..

High level architecture of Cisco's webEx is shown in Figure 3-4.

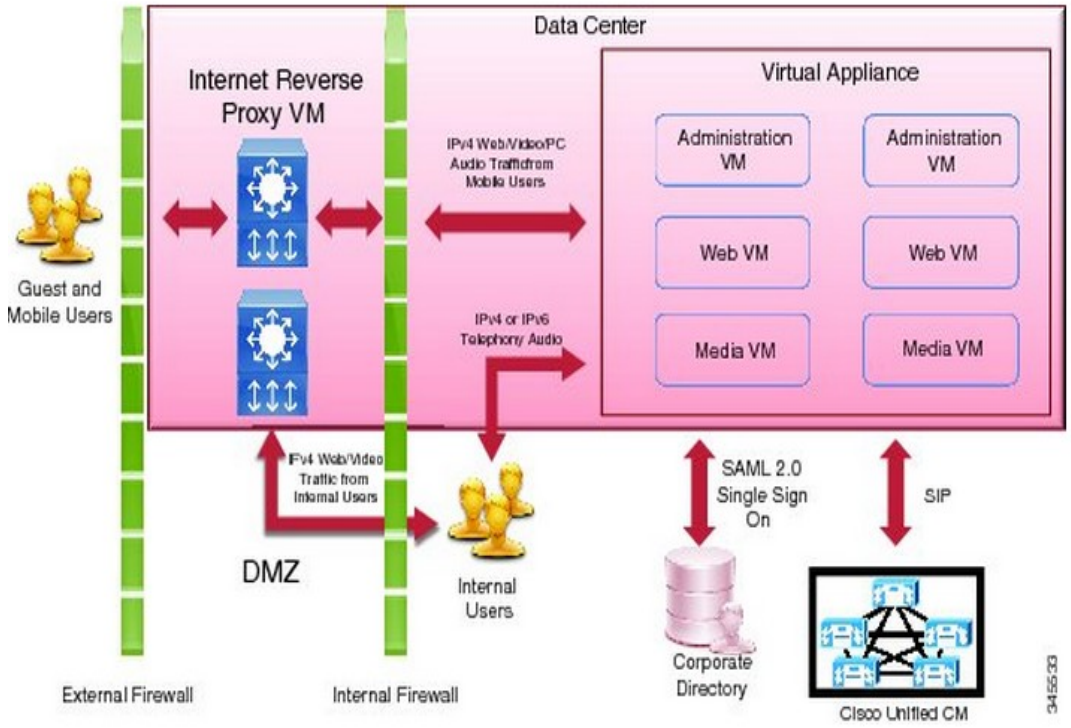

Figure 3-4. High level Architecture of WebEx [30]

BlueJeans is another cloud based conferencing service which supports limited number of participants per meeting.

- As there is no service as Conferencing IaaS offered, our 1st requirement is not applicable.
- BlueJeans is a conferencing service given as SaaS and it is not based on our cloud model. It follows a coarse-grained approach and do not provide an infrastructure with fine-grained substrates that can be published, discovered and shared with other applications. It does not meet our 2nd and 3rd requirements.
- BlueJeans supports up to 25 users per conference meeting [31] and it is not totally scalable and does not satisfy our 4th requirement.
- Blue Jeans provides elasticity and satisfies our 5th requirement. It allows customers to set a right size and then grow, rather than over size and waste.

Reference [37] proposes an audio/video conference application as SaaS called Nuve. It comes along with the virtualized infrastructure. It also allows third party applications take advantage of the conferencing resources. It provides a clouds service to access to a collaborative environment including audio, video and shared applications. It offers videoconference as a Representational State Transfer (REST) web service over an interface which can be used by third parties to enrich their application with the standard HTTP operations.

- It follows a coarse-grained approach which depends on one virtual full-blown conferencing server, and does not propose an infrastructure with fine-grained substrates that can be published, discovered and also shared with other applications. It does not satisfy our 2nd and 3rd requirements.

- The architecture is flexible and scales depending on the number of users, it meets our 4th requirement.
- Elasticity feature is not discussed in the paper [37].

Reference [36] proposes deploying video conferencing applications as SaaS to a hybrid cloud (including OpenNebula, VMWare, Amazon EC2) instead of a single cloud to benefit from resource and cost efficiency in the cloud platform. They reuse an existing video conferencing framework (Isabel) for their validation. It is based on monolithic conferencing servers with tightly coupled structure (they only specify logical boundaries between functional components).

- It does not offer any conferencing IaaS and our 1st requirement on standard interface toward IaaS is irrelevant and not applicable.
- It is not based on fine-grained conferencing substrates which can be published and discovered, so it is not satisfying our 2nd and 3rd requirement.
- The conference applications are only deployed and scaled horizontally as a single unit. It allows a great number of users, but is not totally scalable. It does not satisfy our 4th requirement on scalability.
- Elasticity feature as our 5th requirement is not discussed in the paper [36].

Figure 3-5 shows the general architecture of this proposal. 3 components of the architecture are [36]:

1) API: This is the interface between the scheduler and third party services.
2) Scheduler: responsible for scheduling all the conference events so they are executed in order. It also decides which cloud provider will execute the

conference. Before responding to a new create conference request, it checks if there are enough resources available throughout the duration of conference.

3) Executor: This component executes the jobs which are scheduled (it is the core video conference server of the system).

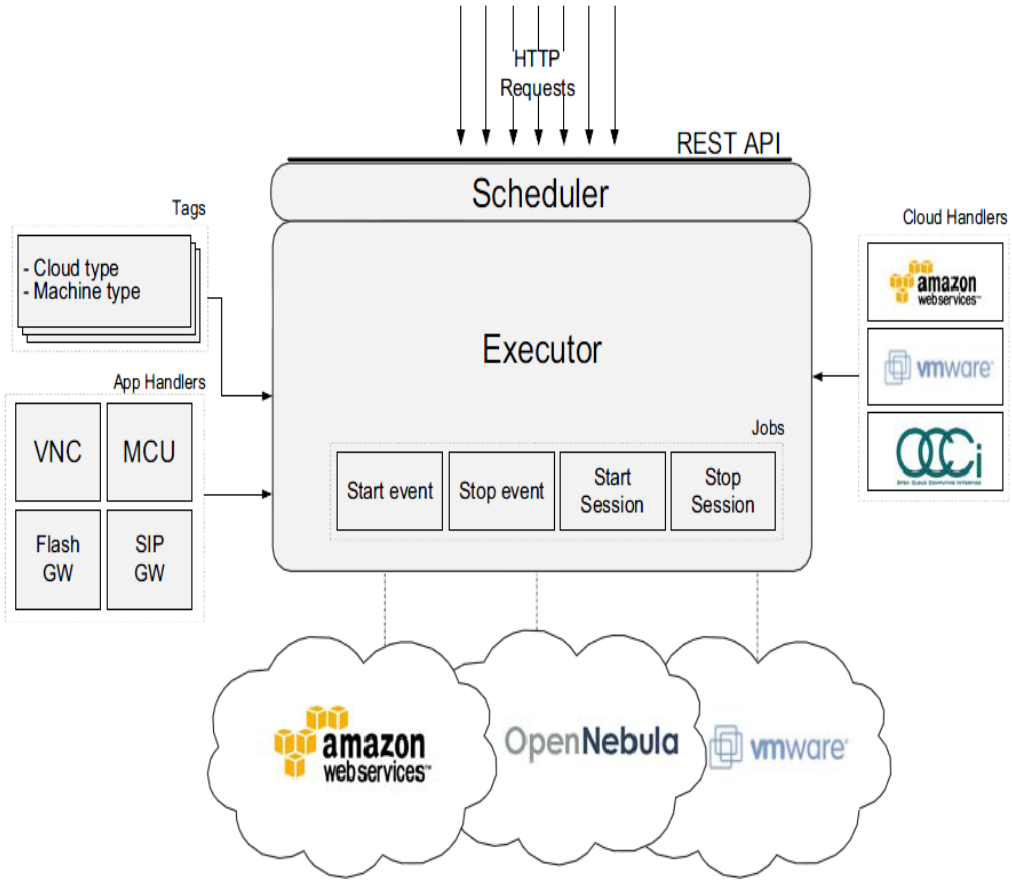

Figure 3-5. Conference Manager Architecture [36]

Reference [25] also offers audio/video conferencing as a service on cloud. This cloud conferencing relies on the Service Oriented Architecture (SOA) and has a loosely coupled structure. They proposed cloud conferencing system is depicted in Figure 3-6. They have divided the architecture into 4 layers from bottom to top; physics layer, virtualization layer, platform layer and application layer.

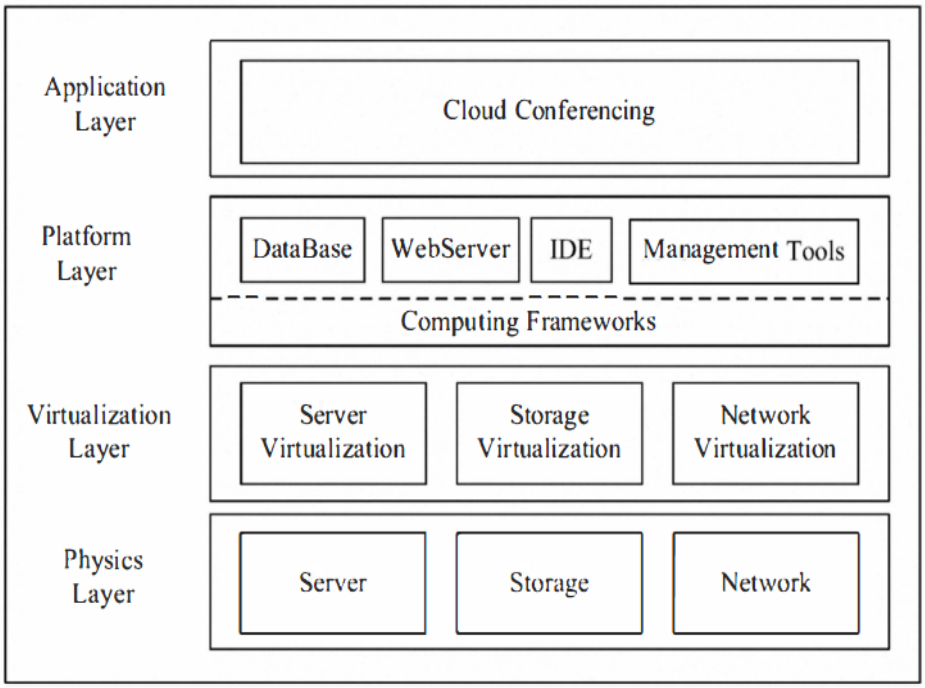

Figure 3-6. Cloud conferencing model [25]

Infrastructure includes both physics and virtualization layer which delivers storage, computing and networking resources. The platform layer is divided into two sub layers. One is the computing frameworks for managing the transactions and task scheduling. The other sub layer is application capability layer has the tools for building applications. This layer delivers a computing platform which consumes cloud infrastructure to facilitate deployment of applications. The application layer is the top layer of this layer (SaaS layer) which delivers the conferencing application as a service over the Internet.

They develop cloud conferencing at the application layer. This layer includes the atomic and composite conferencing services and their business process model. Atomic services include services like audio service, video service, and floor control service (it is eextensible and new service components may be added). The composite services are composed from atomic services. At the service creation time, customers

can get the information of a conference model which describes a conference setting supported by the system. They are actually templates which specify the service process to build a conference service [25]. Figure 3-7 shows how the cloud layers defined in this architecture. Their infrastructure just delivers the physical recourses (storage, computing and networking resources) and virtualization. This architecture does not provide an infrastructure with fine-grained conferencing substrates.

- As our first requirement interfaces toward infrastructure are not discussed in particular but it is mentioned that this system is compatible with IETF standards.
- As our 2nd and 3rd requirements, they mention the architecture is based on SOA with publication and discovery mechanisms, but detailed information is not provided about if there is any expressive discovery enable or not.
- It provides scalability characteristic as our 4th and 5th requirements. New conferencing resources can be added into cloud when required. When participant number increases it gets resources from the resource pool and when participants leave, the resources will be returned to the pool.
- As our 5th requirement, is not discussed if any elasticity feature is provided.

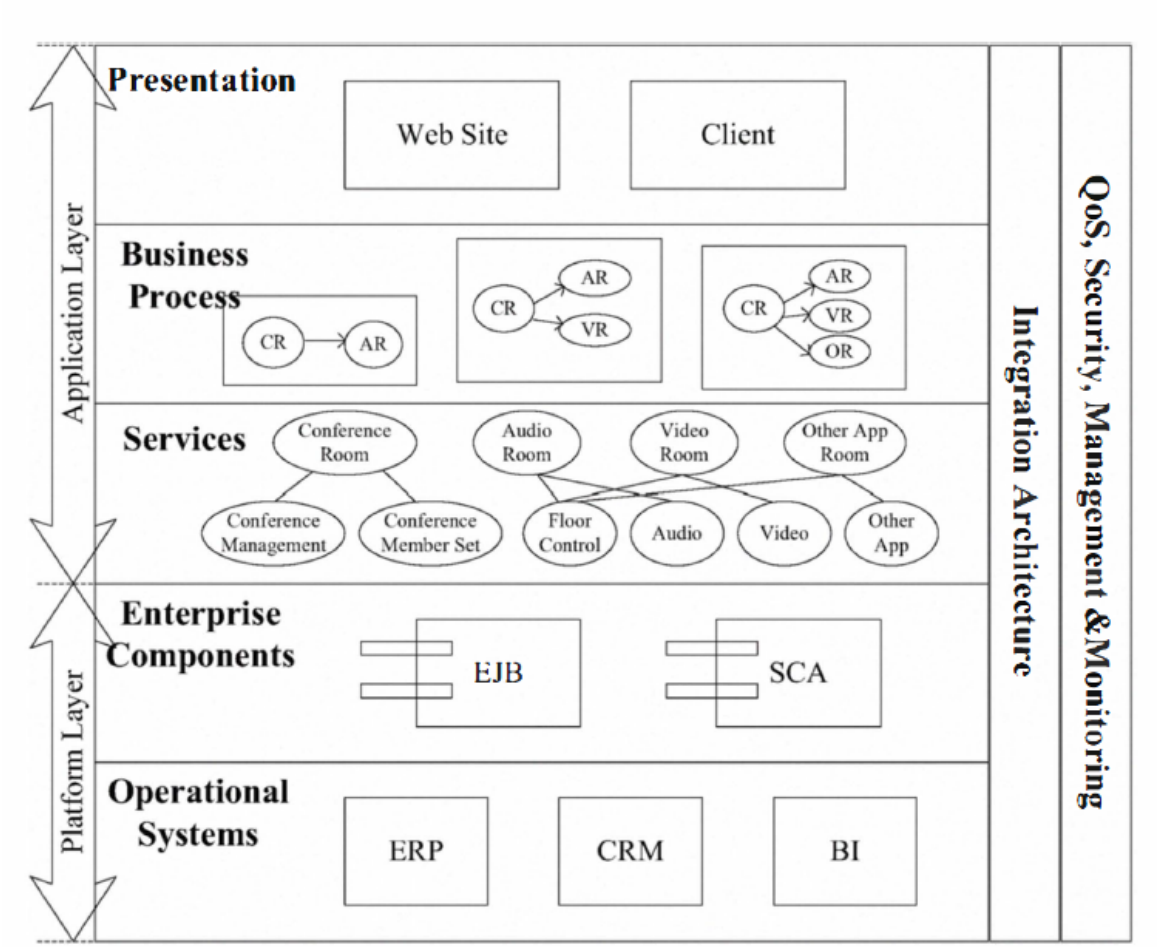

Figure 3-7. SOA Layers of Cloud Conferencing [25]

3.3.3 Description, Publication and Discovery of Cloud Conferencing Services

Regard the challenges related to description, publication and discovery of conferencing services, a student working in our lab (TSE lab) has recently proposed a semantic-oriented description framework and a broker architecture for publication and discovery of cloud based conferencing [38]. This architecture enables the description of virtualized conferencing substrates which captures both technical and business aspects of the substrates. It enables publication and discovery of conferencing substrate to a broker using rich and expressive criteria. A ranking and selection mechanism is also provided for an efficient discovery of conferencing substrates to

enable finding the substrates which fit the best to the user requirements. Figure 3-8 depicts this architecture. The providers interact with the broker via a REST APIs. Broker uses a semantic data store to keep the descriptions of conferencing substrates and the cloud conference ontology (The cloud conference ontology is as reference ontology designed and used for the validation of substrate description documents at the substrate publication). Lower level providers (e.g. conferencing substrate provider) publish to the broker and the higher level providers (e.g. conferencing infrastructure provider) discover from the broker.

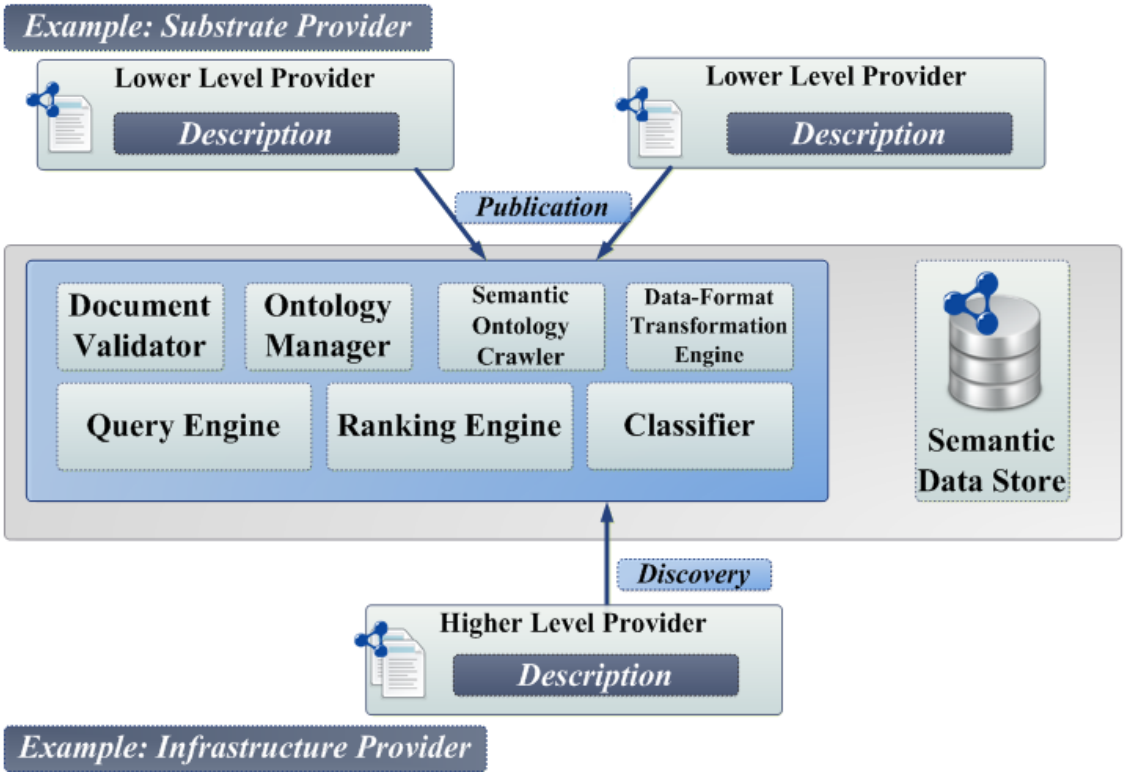

Figure 3-8. General Broker Architecture for Publication and Discovery [38]

- This work satisfies our 2nd and 3rd requirements by providing a mechanism for publication and discovery of conferencing substrates.

- Our 1st requirement on stand-based interfaces toward conferencing infrastructure can be satisfied by this work as the publication and discovery interfaces are RESTful standard interfaces.
- Our 4th and 5th are irrelevant to this work.

This work is the most relevant work in the state of the art to what we need and can be integrated to our proposed architecture to provide the broker and the substrate publication and discovery functionalities.

Table 1 summarises our evaluation on the state of the art.

Related Work \ Requirements		Req. 1	Req. 2	Req. 3	Req. 4	Req. 5
Standard non-cloud conferencing frameworks	IETF SIPPING	Not applicable	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied
	IETF XCON	Not applicable	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied
	IETF DCON	Not applicable	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied
	3GPP IMS	Not applicable	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied
Cloud based conferencing frameworks	WebEx	Not applicable	Not Satisfied	Not Satisfied	Not satisfied	Information not available
	BlueJeans	Not applicable	Not Satisfied	Not Satisfied	Not Satisfied	Satisfied
	Isabel [36]	Not applicable	Not Satisfied	Not Satisfied	Not Satisfied	Not discussed
	Nuve [37]	Not applicable	Not discussed	Not discussed	Satisfied	Not discussed
	Service oriented cloud conferencing [25]	Satisfied	Satisfied	Information not available	Satisfied	Not discussed
Discover and publication of cloud conferencing services	Broker for publication and discovery of conferencing applications [38]	Satisfied	Satisfied	Satisfied	Not applicable	Not applicable

Table 3-1.State of the Art Evaluation

3.4 Chapter Summary

In this chapter we presented a scenario that illustrates the usage of cloud-based conferencing infrastructure in the cloud conferencing business model. Then we extracted the requirements based on the scenario and explained what is expected from the cloud-based conferencing infrastructure.

Finally, we presented the state of the art. We categorized the related work based on the link they had with our work and reviewed them in 3 different sections.

Based on our studies, there is no full-fledge cloud conferencing infrastructure which can meet all of our requirements completely.

We specified a previous work on publication and discovery of conferencing substrates which satisfies our requirements for publication and discovery and can be reused in our proposed architecture which we will present in the next chapter.

Chapter 4

4 Proposed Architecture

In the previous chapter, we derived our requirements for a virtualized infrastructure for cloud-based multimedia conferencing applications. Accordingly, in this chapter we discuss the proposed architecture for a virtualized cloud infrastructure for multimedia conferencing application which is based on the requirements.

We start by explaining the overall architecture. We explain the functionalities of each architectural component and the communication interfaces. Then, we describe an illustrative scenario for a specific type of conferencing applications, showing the sequences which take place during service creation, activation and execution. Then we explain how the requirements are met by the architecture. And finally, we summarize this chapter.

4.1 Overall Architecture

The focus of this thesis is on the conferencing infrastructure provider and substrates provider roles. Figure 4-1 depicts the overall architecture. There are two layers shown in the figure: Conferencing IaaS (Infrastructure as a Service) and Conferencing SubaaS (Substrate as a Service) layers. Conferencing SubaaS is a term we use for the conferencing substrates that can be offered by substrate providers as service. A substrate provider may offer one or more conferencing substrates. Based on the cloud conferencing business model [1], an Infrastructure provider can interact with several substrate providers. The SubaaS layer will encompass all the Conferencing SubaaSes given by different substrate providers.

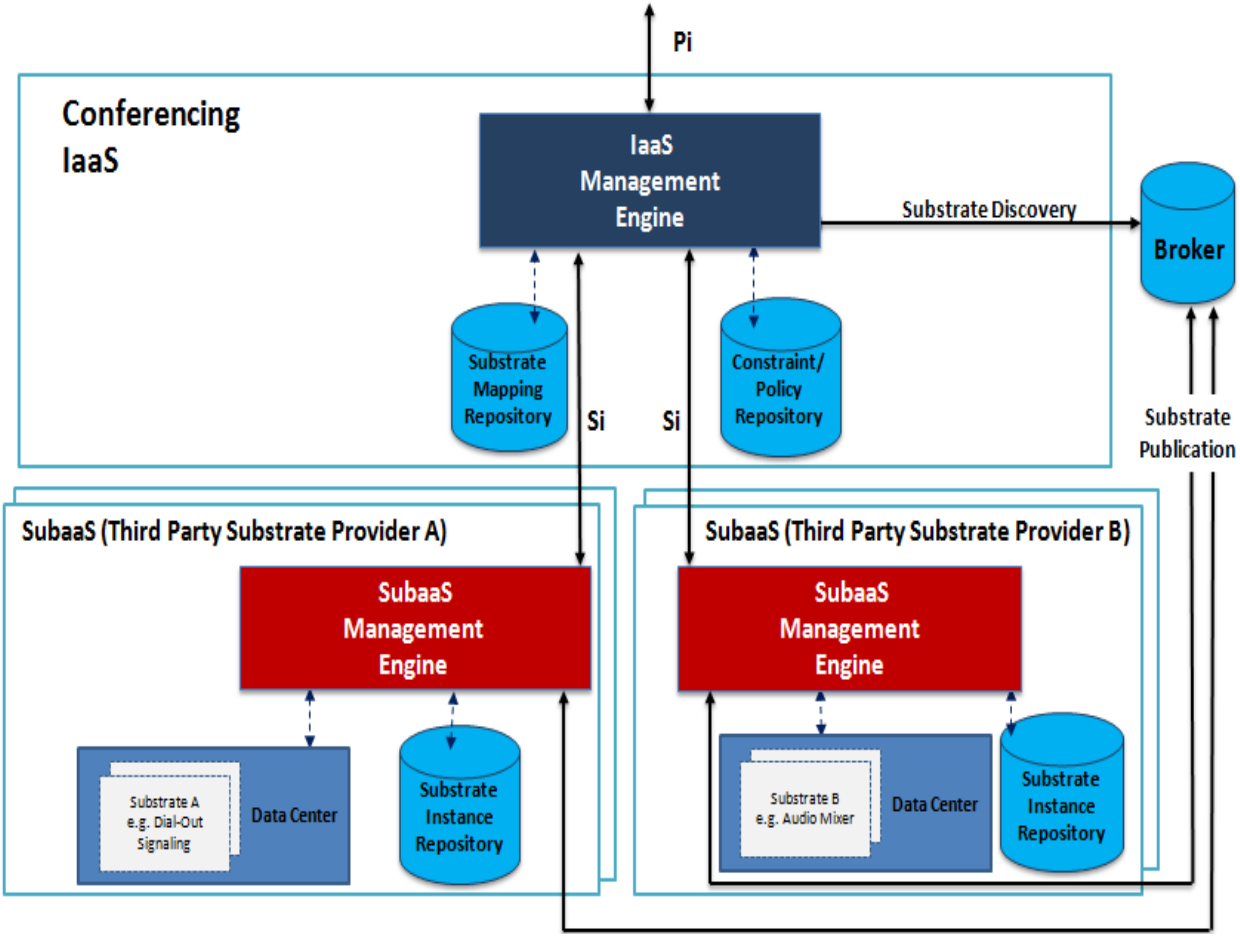

Figure 4-1. Architecture of Cloud-based Conferencing IaaS

The Conferencing IaaS layer contains the functional components that realize the conferencing infrastructure provider role and the SubaaS layer contains the functional components that realize the conferencing substrate provider role. There is also a broker that enables the interactions between infrastructure provider and the substrate providers. A conferencing infrastructure provider may interact with several conferencing substrate providers. Also, a conferencing substrate provider may interact with several conferencing infrastructure providers. For the sake of simplicity, we assume that there is only one conferencing infrastructure provider. Later the architecture may extend to include more than one infrastructure provider.

Conferencing substrate providers offer different type of conferencing substrates (e.g. dial-out signaling, dial-in signaling, audio mixer, video mixer, floor control server, etc.). Each conferencing substrate provider may offer one or several conferencing substrates. For simplicity, we just have shown 2 third party conferencing substrate providers in figure 4-1 (Substrate provider A and substrate provider B). However, the set of substrate providers will not be limited to them and any other substrate providers will belong to the SubaaS layer as well. We assume that substrate provider A is offering dial-out signaling substrate and substrate provider B is offering audio mixer substrate.

In the following subsections, we will explain the architectural components and the communication interfaces shown in the general architecture (figure 4-1).

4.1.1 Architectural Components

We divide the architectural components of the general architecture into three categories of broker, components of the IaaS Layer and components of the SubaaS layer, and will explain them in 3 different sub-sections.

4.1.1.1 Broker

The broker contains the information of substrates and supports the interactions between the infrastructure and substrate providers. Substrate providers publish the substrates through this broker and infrastructure provider discovers substrates from this broker.

The broker used in this architecture, is rooted in a previous work done by another student from our lab [38] which proposes a semantic-oriented description framework and a general architecture for broker for publication and discovery of cloud-based

conferencing substrates and services [38]. In chapter 3, we reviewed this work and stated that this broker satisfies our requirements for publication and discovery of conferencing substrates. So we integrate this broker in our architecture. It captures both technical and business aspects of conferencing substrates. Provisioning of business aspects in the substrate descriptions will enable a richer and more expressive discovery. The technical aspects include the functional features of a conferencing substrate, e.g. Create Conference, Call participant, Delete participant, Delete Conference functions in a dial-out signalling substrate. The business aspects include the non-functional information of the substrate such as the provider's information, cost of using substrate and the pricing model, QoS the substrate offers, etc. The broker provides RESTful based interfaces for publication and discovery of substrates.

This broker considers 3 different levels based on the publishers and the requesters of a broker level. The information provided in the substrate descriptions in different levels may differ. There are 3 broker levels with different publishers/requesters. We describe an end to end publication and discovery operations based on cloud conferencing business model. We explain it in seven steps which are shown in figure 3. Steps 1, 3, and 6 denote publication by the substrate providers, infrastructure providers, and service providers, respectively. Steps 2, 5, and 7 denote discovery by the infrastructure provider, platform provider, and end-users, respectively. During publication at each broker level, the providers will alter information in the description before publishing the description to the higher level broker. This publication flow provides flexibility for the providers to add or manipulate functionality or constraints. For example, after discovery from the level 1 broker, the infrastructure provider will alter the substrate provider related information with infrastructure provider information or it may change the cost model of substrate usage in the substrate

description before publishing it to the level 2 broker. For more details please refer to [38].

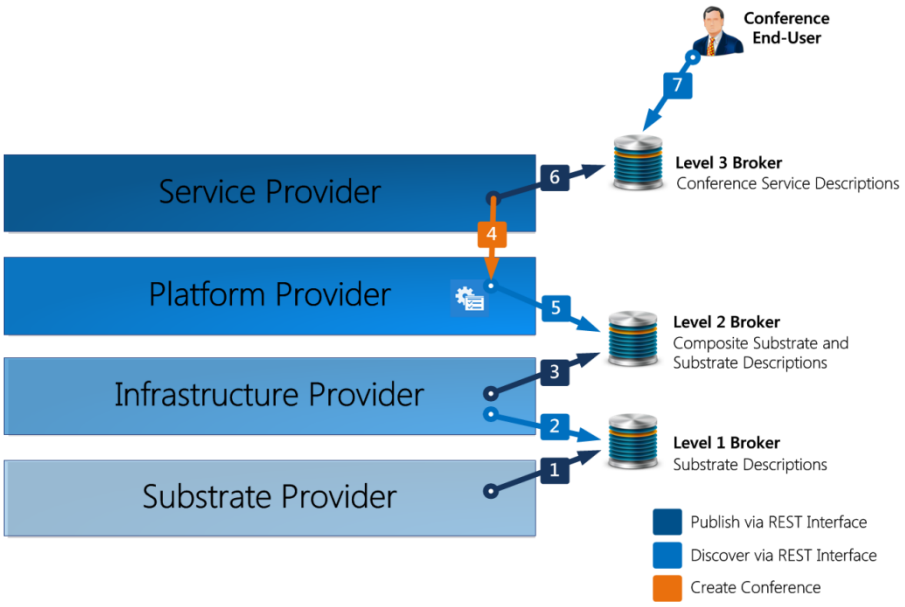

Figure 4-2. End-To-End Publication and Discovery Steps [38]

4.1.1.2 IaaS Layer Components

Based on the cloud conferencing business model [1], conferencing IaaS is the layer between conferencing PaaS and conferencing SubaaS layers Conferencing IaaS has two main responsibilities of: 1) provisioning conferencing resources for conferencing PaaS by identifying appropriate substrate resources which can be provided by different substrate providers, and 2) coordinating the execution of conference services by redirecting the execution request from PaaS to the substrates which serve the conference service. In this section we explain each one of the architectural components of conferencing IaaS layer shown in figure 4-1.

- **IaaS Management Engine:** The key functional component of the conferencing IaaS layer is the IaaS Management Engine. This component accepts the substrate activation requests for a conferencing service from

Conferencing PaaS, searches the broker to find the proper substrates to be activated and then sends activation requests to the target SubaaS for provisioning the required substrates. Later, it may also request for deactivation of the substrates upon a deactivation request from conferencing PaaS. It also redirects execution related requests coming from PaaS to the target SubaaS.

- **Substrate Mapping Repository**: Conferencing IaaS uses this internal repository to save the substrate mapping information. It has the information of substrate instances provided by different substrate providers for different conference services during the service activation phase. This information later will be used during the service execution phase to address a certain substrate instance.
- **Constraint/ Policy Repository:** Conferencing IaaS has another internal repository which saves the constraints of the substrates requested by the conferencing PaaS (e.g. QoS requirements, cost, etc.) and also any internal policies (e.g. cost justifications) which IaaS may require to take into account while discovering and selecting a substrate.

4.1.1.3 SubaaS Layer Components

In this section we explain each of architectural components of conferencing IaaS layer shown in figure 4-1.

- **SubaaS Management Engine:** The key component of the SubaaS layer is the SubaaS Management Engine. This component handles several functionalities: publication of the substrates to the broker, virtualization and instantiation of substrates, monitoring the substrate instances and managing the resource

allocation to the substrate instances for scaling purposes. Details on the resource allocation mechanism that substrate provider uses to realize the scaling support is discussed in chapter 5.

- **Data center:** the data center at substrate provider's cloud which houses the Physical Machines (PM) hosting the VMs which run the substrates. Substrate provider may also deploy the services on VM instances they get from other cloud providers e.g. Amazon EC2 services.
- **Substrate Instance Repository:** Each Substrate provider has an internal repository to save the information of the substrate instances created (e.g. substrate type, PM's address hosting the substrate, VM's name, IP address of VM, etc.).

4.1.2 Communication Interfaces

In this section we explain each of communication interfaces shown in the general architecture. They are namely Publication interface, Discovery interface, Pi interface and Si interface.

- ***Publication interface:*** is the interface between SubaaS Management Engines and the broker for the publication of conferencing substrates. It is a RESTful interface provided by broker that substrate providers can publish their substrate descriptions through it.
- ***Discovery interface:*** is the interface between IaaS Management Engine and the broker which enables IaaS to discover the conferencing substrates previously published by substrate providers. It is a RESTful interface which provides an advanced search by criteria function for an expressive discovery. It also has the capability to rank the search result based on the importance

given to each criterion by the requester using a ranking algorithm. For example if the most important criterion is the cost, the discovery result for a specific conferencing substrate will be sorted in a way which the less costly substrate is on the top of the list.

- ***Pi interface*:** is an interface between Conferencing PaaS and IaaS Management Engine to forward service activation and execution related requests from PaaS to IaaS.
- ***Si interface:*** is the interface between Conferencing IaaS and SubaaS Management Engine to forward activation and execution related requests from IaaS to SubaaS.

We have chosen REST for the implementation of Pi and Si interfaces. RESTful web services provide several advantages: REST is lightweight, standards-based, offers a uniform interface (REST resources can be accessed and manipulated in a standard way), is flexible in terms of the supported data formats (e.g., plain text, JavaScript Object Notation (JSON), and Extensible Markup Language (XML)), and is lightweight comparing to other web service technologies. REST has also the addressability advantage, which means that the REST models the information as resources, where a resource is any information that is important to be named and referenced.

In this section we present Pi and Si interfaces which we have designed for activation and execution of substrates at IaaS and SubaaS levels. They contain two categories of APIs: activation related APIs and execution related APIs. The reader should note that the activation related APIs are generic, but execution related APIs are specific to the

type of target conferencing substrate. Based on the architectural principle to using RESTful web services for communications, all substrates should provide a REST interface to The interfaces we present in this section are designed for dial-out signaling and audio mixer substrates. Similarly, for the other types of conferencing substrates the REST interfaces can be designed and then substrate providers can use them as standard templates for designing the REST interfaces of conferencing substrates.

Table 4-1 shows the REST APIs provided in the Pi interface at conferencing IaaS level to be called by conferencing PaaS. It has some APIs for activation and deactivation of a conference service. It has some other some other APIs for the coordination of execution of a dial-out audio conference application which is based on two dial-out signaling and audio mixer substrates.

Operation	Method name	Path	Request body parameters	Response	Direction
ActivateService	POST	/Activation/Services/	Set of SubstrateInfo	ServiceId, Set of SubstrateIds	PaaS→ IaaS
DeactivateService	PUT	/Activation/Services/ {ServiceId}/status	ServiceId	200 OK	PaaS→ IaaS

CreateConference	POST	/Execution/Services/{ServiceId}/DialOutSignaling/Conferences	ServiceId, ConfInfo	ConfId	PaaS→ IaaS
DeleteConference	DELETE	/Execution/Services/{ServiceId}/DialOutSignaling/Conferences/{ConfId}	ServiceId, ConfId	200 OK	PaaS→ IaaS
CallParticipant	POST	/Execution/Services/{ServiceId}/DialOutSignaling/Conferences/{ConfId}/Participants	ServiceId, ConfId, UserInfo, CallBackInfo	ParticipantId	PaaS→ IaaS
SendAkhnowledgmentToParticipant	POST	/Execution/Services/{ServiceId}/DialOutSignaling/Conferences/{ConfId}/Participants/{ParticipantId}	ServiceId, ParticipantId, MixerInfo	200 OK	PaaS→ IaaS
DeleteParticipant	DELETE	/Execution/Services/{ServiceId}/DialOutSignaling/conferences/{ConfID}/Participants/{ParticipantId}	ServiceId, ParticipantId	200 OK	PaaS→ IaaS

CreateConference	POST	/Execution/ Services/{ServiceId} /AudioMixer/ Conferences	ServiceId, ConfInfo	ConfId	PaaS→ IaaS
DeleteConference	DELETE	/Execution/ Services/{ServiceId}/ AudioMixer/ Conferences/{ConfId}	ServiceId, ConfId	200 OK	PaaS→ IaaS
StartReceivingAudio	POST	/Execution/ Services/{ServiceId}/ AudioMixer/ Conferences/ {ConfID}/Participants	ServiceId, ParticipantId	200 OK	PaaS→ IaaS
SendSendingAudio	POST	/Execution/ Services/{ServiceId}/ AudioMixer\/ Conferences/ {ConfID}/Participants/ {ParticipantId}/ Connect	ServiceId, ParticipantInfo	200 OK	PaaS→ IaaS
DeleteParticipant	DELETE	/Execution/ Services/{ServiceId}/ AudioMixer/ Conferences /{ConfID}/Participants {ParticipantId}	ServiceId, ParticipantId	200 OK	PaaS→ IaaS

Table 4-1. RESTful Pi interface for Dial-out Audio Conference (Conferencing Infrastructure Provider)

Table 4-2 shows the APIs provided in a Si interface by a substrate provider offering a dial-out signaling substrate as SubaaS to be called by conferencing IaaS. It contains some APIs for substrate activation and deactivation, and some execution APIs for different operations of a dial-out signaling substrate to be called by conferencing IaaS.

Operation	Method name	Path	Request body parameters	Response	Direction
ActivateSubstrate	POST	/Activation/Substrates/	SubstrateInfo, ReferenceId	SubstrateId	IaaS→ SubaaS
DeactivateSubstrate	PUT	/Activation/Substrates/ {substrateId}/status	SubstrateId	200 OK	IaaS→ SubaaS
NotifySubstrate Activation	POST	Notification/Substrates/ {Referenced}	ReferenceId , SubstrateId	200 OK	SubaaS→ IaaS
CreateConference	POST	/Execution/ DialOutSignaling/ Conferences	ConfInfo	ConfId	IaaS→ SubaaS
DeleteConference	DELETE	/Execution/DialOutSignaling/Conferences/	ConfId	200 OK	IaaS→ SubaaS

		{ConfId}			
CallParticipant	POST	/Execution/ DialOutSignaling/ Conferences/{ConfId}/ Participants	ConfId, UserInfo, CallBackInfo	ParticipantId	IaaS→ SubaaS
SendAckTo **Participant**	POST	/Execution/ DialOutSignaling/ Conferences/{ConfId}/ Participants/ {ParticipantId}	ParticipantId, MixerInfo	200 OK	IaaS→ SubaaS
DeleteParticipant	DELETE	/Execution/ DialOutSignaling/ Conferences/{ConfID}/ Participants/ {ParticipantId}	ConfId ParticipantId	200 OK	IaaS→ SubaaS

Table 4-2. RESTful Si interface of a Dial-out signaling (Substrate Provider A)

Table 4-3 shows the APIs in a Si interface provided by a substrate provider for an audio mixer substrate offered as SubaaS. It has some APIs for substrate activation and deactivation, and some execution APIs for different operations of an audio mixer substrate to be called by conferencing IaaS.

Operation	Method name	Path	Request body parameters	Response	Direction
ActivateSubstrate	POST	/Activation/Substrates	SubstrateInfo	SubstrateId	IaaS→ SubaaS
DeactivateSubstrate	PUT	/Activation/Substrates/ {substrateId}/status	SubstrateId	200 OK	IaaS→ SubaaS
NotifySubstrate Activation	POST	Notification/Substrates/ {Referenced}	ReferenceId , SubstrateId	200 OK	SubaaS→ IaaS
CreateConference	POST	/Execution/AudioMixer / Conferences	ConfInfo	ConfId	IaaS→ SubaaS
DeleteConference	DELETE	/Execution/AudioMixer /conferences/{ConfId}	ConfId	200 OK	IaaS→ SubaaS
StartReceivingAudio	POST	/Execution/AudioMixer /conferences/{ConfId}/ Participants	confId ParticipantInfo	200 OK	IaaS→ SubaaS
StartSendingAudio	POST	/Execution/AudioMixer /Conferences/{ConfId}/ Participants/ {ParticipantId}/ Connect	confId, ParticipantId	200 OK	IaaS→ SubaaS
DeleteParticipant	DELETE	/Execution/AudioMixer /Conferences/{ConfId}/ Participants/ {ParticipantId}	confId, ParticipantId	200 OK	IaaS→ SubaaS

Table 4-3. RESTful Si interface of an Audio Mixer (Substrate Provider B)

4.2 Illustrative Scenario

In this section we provide an illustrative scenario to demonstrate the interactions between different architectural components during service creation, service activation and service execution. First briefly we explain the service creation phase, and then we explain the interactions during service activation phase and after that the interactions during service execution phase. Conference application used as example is a dial-out audio conference service.

4.2.1 Service Creation

Service creation is the first step toward building a conferencing application. After a conference service is created, it can be activated and then executed. The details of service creation phase and how a conferencing PaaS composes several conferencing substrates to develop a conferencing application are out of scope of this work and should be investigated in future.

In this thesis we assume that the conferencing service provider has used a service creation tool provided by the conferencing PaaS and has created a dial-out audio conference service. We assume that as the result of service creation, the workflow of the conference application is created from composition of two substrates: a dial-out audio signaling substrate and an audio mixer substrate. The workflow is the logic of application which defines the flow of execution for different requests. Based on this assumption we focus only on the conferencing infrastructure and the interactions which take place in next steps during service activation and execution.

4.2.2 Service Activation

Conferencing service provider initiates the service activation process. Service provider uses a service activation tool provided by Conferencing PaaS and requests for the activation of the dial-out audio conference application created during service creation phase.

Conferencing PaaS sends a service activation request to IaaS Management Engine to activate the conferencing substrates involved in this conferencing service. They are dial-out signaling and audio mixer substrates in this example. Conferencing PaaS sends information of required substrates along with the activation request (e.g. type of substrate, substrate features, QoS requirements, etc.). IaaS Management Engine upon receiving the activation request will continue the activation process by searching into the broker to find the requested substrates. The criteria for required substrate are specified in the request (e.g. substrate type is an audio mixer and it should support up to 100 users). The broker finds the substrate that matches to the criteria given and returns back the information of substrate and also its provider the IaaS Management Engine. Assume that discovered substrates are a dial-out signalling substrate given by substrate provider A and an audio mixer substrate given by substrate provider B.

Now IaaS Management Engine will contact each of these substrate providers to ask for instantiation of substrates. Substrate activation requests will be sent to corresponding SubaaS Management Engines of these two SubaaSes. Each SubaaS Management Engines checks the resource availability and then instantiates a substrate, save the instance information in the internal substrate instance repository and returns back the instantiation result to the IaaS Management Engine. In case of a successful activation for both substrates, IaaS Management Engine sends back the

positive activation response to the conferencing PaaS. The sequence of service activation is shown in figure 4-3. The substrate mapping information will be saved in the substrate mapping repository of IaaS for later use. Now that the back end for the dial-out audio conference application SaaS is provided, and end users can start using this application.

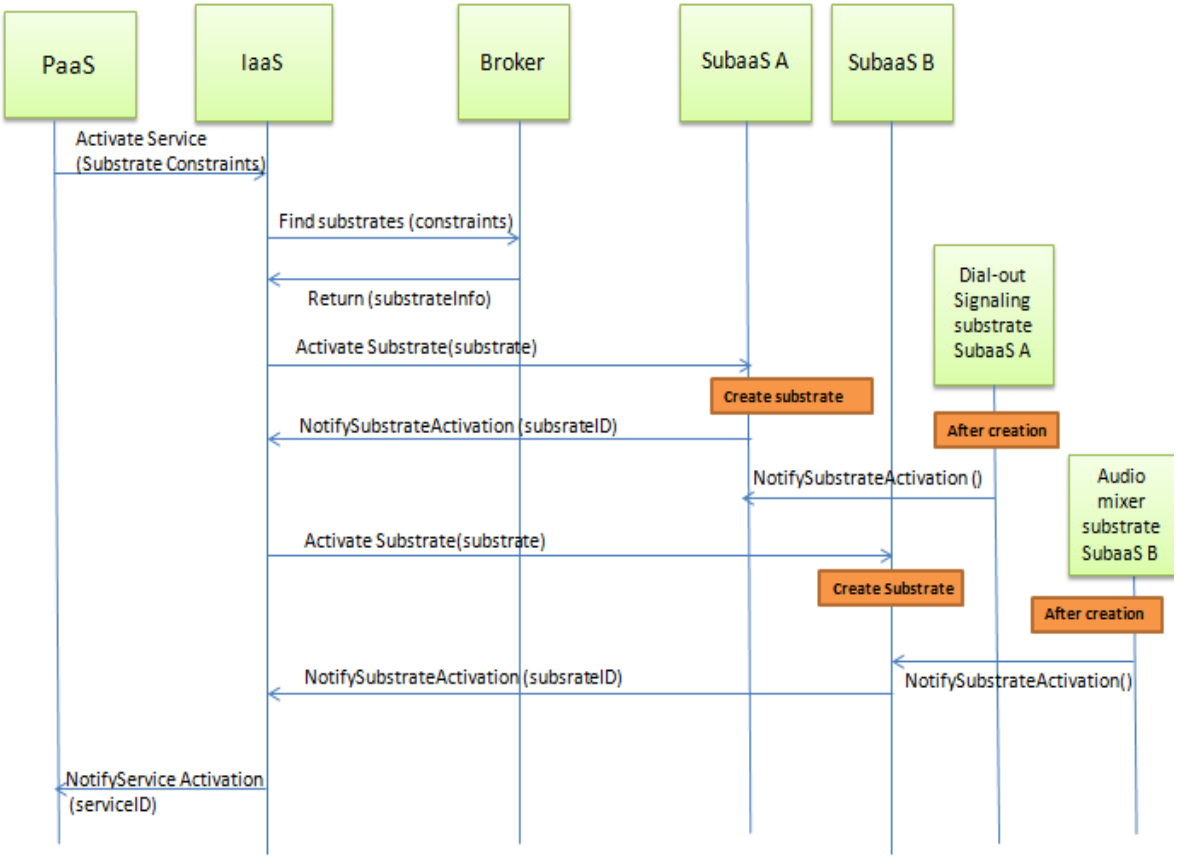

Figure 4-3 .Service Activation Sequence

4.2.3 Service Execution

Administrator of the conference application now can use the conference application and perform different functions (e.g. see the list of registered users, asks for creation of a conference and then add users to a conference). The main functions to be requested by conference administrator in a dial-out audio conference application are *Create*

Conference, Add Participant to conference, Delete Participant from conference and Delete Conference. Execution requests pass through the workflow of the service hosted at conferencing PaaS (the workflow was produced at the time of service creation). In order to process the request, conferencing PaaS sends substrate execution requests to the IaaS Management Engine and IaaS Management Engine sends redirects the requests to the target SubaaS Management Engines. Eventually when request is received by the actual substrate instance, it will be executed there.

Process of execution of a request may have several substrates involved. So depending on the request, the workflow of a request may contain several sub processes which will be executed one after another until the initial request is completely fulfilled. In the next two sub-sections we explain the execution sequence for two main types of requests in a dial-out audio conference service: *Create Conference* and *Add Participant to conference* requests.

4.2.3.1 Create Conference Execution Process

In the dial out audio conference application, when an end user asks for creation of a conference, this request will have both dial-out signaling substrate and also the audio mixer substrate involved. The workflow for a *create conference* consists of two sub processes. The first one asks dial-out signaling substrate to create a new conference and the second asks audio mixer substrate to reserve the resources for the new conference. Only after both these two sub processes get executed successfully, the initial request which was a *Create Conference* request can be considered as successfully processed. The sequence of *Create Conference* execution is shown in figure 4-4.

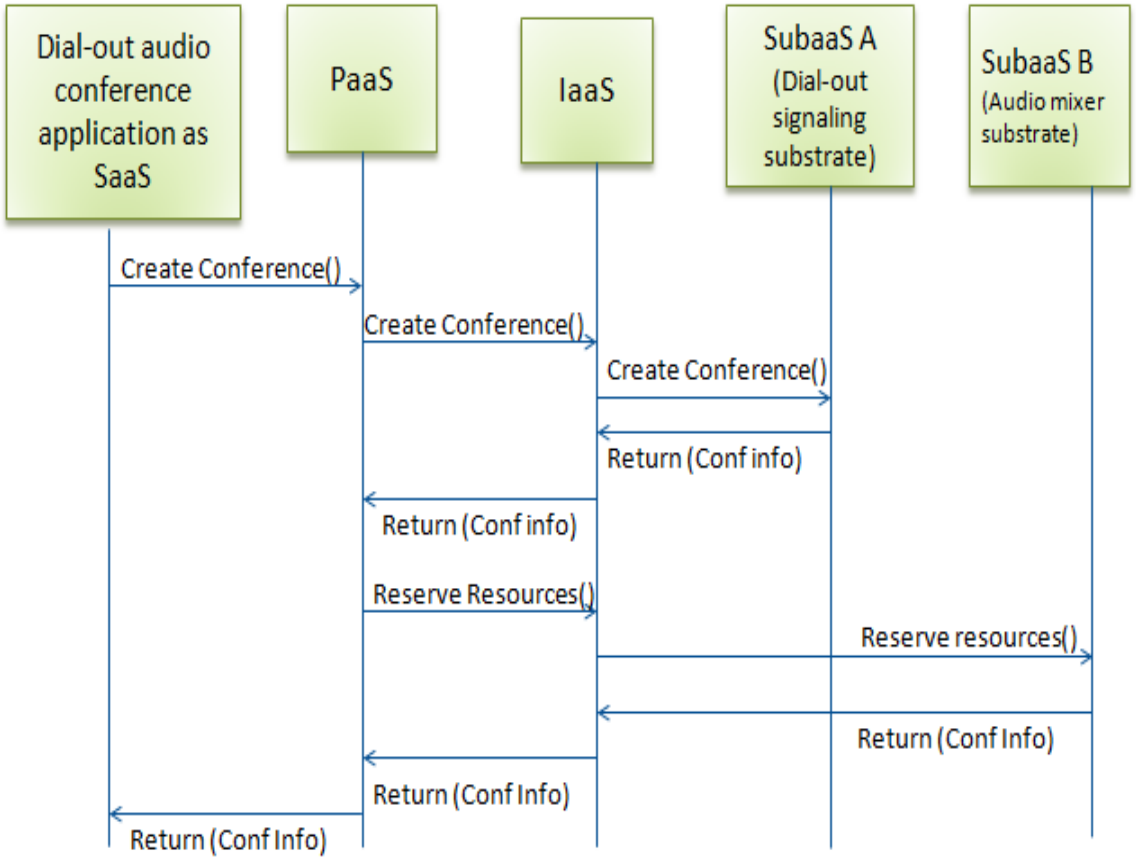

Figure 4-4. Create Conference Execution Sequence

4.2.3.2 Add Participant to the Conference Execution Process

In the dial out audio conference application, after creation of a conference, the admin of the conference application can check the registered users, select them and ask en an end user asks for adding them to the conference as conference participants. The *Add Participant* request will have both dial-out signaling substrate and also the audio mixer substrate involved. The workflow for an *Add Participant* request consists of three sub processes. The first one is to ask audio mixer to reserve the resources required for the end user as new participant (E.g. reserving a port and setting a Real-time Transport Protocol (RTP) endpoint to receive the audio from participant from that end point). The second one is to ask dial-out signaling substrate to call the end user and give the

audio mixer information to let end user know where to send the media and then send acknowledgment to end user after receiving answer of call. The third one is to ask audio mixer substrate to start sending audio to the new participant giving the information of participant (E.g. participant's IP number and the receiving port number). From now on a bidirectional media session is established between participant and the audio mixer. The sequence of *Add Participant* execution is shown in figure 4-5.

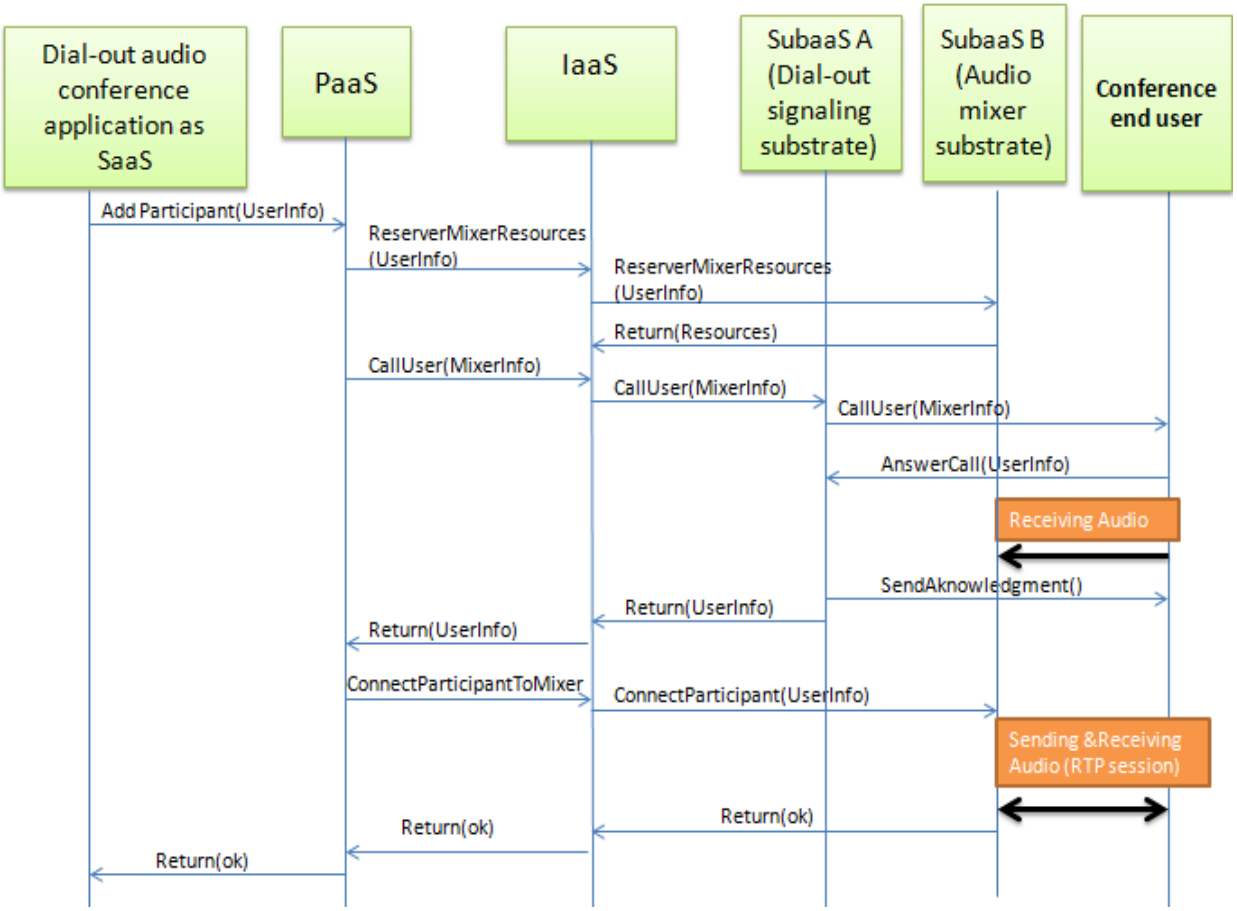

Figure 4-5. Add Participant Execution Sequence

4.3 How the Requirements are Met by the Architecture

In this section we explain that how the proposed architecture satisfies our requirements.

Requirement #1 was that the IaaS provider should interact with all the other actors via standard based interfaces to ease interoperability. In our architecture we adopt REST for the design and implementation of interfaces which is standards-based.

Requirement #2 was that third party substrate providers should be able to make their substrates known to the IaaS provider by publishing them, providing the characteristics of substrates in their description (e.g. substrate provider information, functional features of substrate, the QoS they offer, their cost). We have integrated a broker in our architecture which enables publication of conferencing substrates to the broker which stores the semantic description of substrates in which they can be discovered by conferencing infrastructure provider. The broker was developed by another student of our lab as previously stated.

Requirement #3 was that conferencing infrastructure should be able to search for the substrates by criteria (e.g. functional features of substrate, QoS requirements, cost, etc.) in order to discover the substrates which fit the best to the requirements. The integrated broker in our architecture enables an expressive discovery of conferencing substrates for infrastructure provider and provides an advanced search option based on criteria.

Requirement # 4 was scalability of conferencing IaaS in terms of number of end users which use the conferencing applications/services and in terms of number of applications/services that can be offered by application/service providers as SaaS.

Scalability is improved in several ways in this architecture. First, this architecture uses atomic conferencing components (we called them substrates) as elements to build a conferencing service, instead of coarse grained monolithic conference servers. This improves scalability significantly. Second, in this cloud conferencing business model, a conferencing application can be built using conferencing substrates from different substrate providers and is not limited to a single provider. This improves the scalability in terms of the number of conference applications that can be created. And third, each substrate provider uses a scaling mechanism to take care of scaling the substrates based on demand. We explain this scaling mechanism in the next chapter.

Requirement #5 was that IaaS should scale in an elastic way for resource efficiency purposes. The scaling mechanism that we propose to be used for scaling conference substrates is elastic and performs scaling at the granularity of underlying hardware resources. Chapter 5 provides information on the scaling mechanism.

4.4 Chapter Summary

In this chapter we presented the overall architecture for a virtualized infrastructure for cloud-based conferencing applications. In this architecture we integrated a broker from a previous work into our architecture. This broker satisfies our requirements for the publication and discovery of conferencing substrates.

We explained all the architectural components and the communication interfaces in the proposed architecture in detail.

We presented an illustrative scenario to give a clear view of the events that take place during activation and execution of conferencing applications in respect to the

architecture and the cloud based business model. We also showed the sequence of execution of different requests in a dial-out audio conference application as an example to support the illustrative scenario described.

At the end of chapter, we discussed how our proposed architecture meets the requirements that we specified in chapter 3.

Chapter 5

5 Resource Allocation Mechanism

In chapter 4, we explained the functionality of each architectural component in the proposed architecture. We mentioned that monitoring and resource allocation management are among the tasks assigned to SubaaS Management Engine component of a SubaaS. In this chapter first we state the problem and discuss the related challenges. Then we review existing scaling approaches for resource allocation management and show that they are not adequate for the specific case of conferencing substrate. After that, we propose our scaling mechanism. At the end of chapter, we conclude.

5.1 Problem Statement

In chapter 3 we mentioned that elastic scalability based on demand as a requirement for a virtualized cloud conferencing infrastructure. In order to meet this requirement, conferencing substrate providers as the actual providers of conferencing resources for a conferencing application have to implement an elastic scaling mechanism. First we define salability and elasticity terms, as they are important prerequisites for stating of the problem.

The two terms of **scalability** and **elasticity** are broadly used in the context of cloud computing. In [57], authors define scalability as the ability of a system to sustain increasing workloads with adequate performance by making use of additional resources.

Several organizations have defined elasticity. Open Data Center Alliance (OCDA) defines elasticity as the ability to scale up and scale down capacity based on subscriber's workload [57].

NIST defines rapid elasticity as: "Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited which can be appropriated at any time [57]".

Scalability is a prerequisite for elasticity, but it does not consider how fast, how often, and at what granularity scaling actions can be performed. Elasticity aims at matching the amount of resources allocated to a service with the amount of resources it actually requires, avoiding over- or under-provisioning. Over-provisioning, i.e., allocating more resources than required, should be avoided as the service provider often has to pay for the resources that are allocated to the service" [57].

The problem is that conferencing substrate providers need to scale and be responsive to a growing demand. In other hand substrate provider needs to scale to the demand in an elastic way to be resource efficient and to realize the elastic provisioning support as a key feature for cloud applications. Thus, substrate providers need to follow an elastic scaling approach appropriate for scaling the conferencing substrates which support a conferencing application. They need adequate algorithms.

5.2 Existing Scaling Approaches

In this section we briefly review different existing scaling approaches for non-cloud and clouds applications. They are namely: static resource allocation mechanism for non-cloud applications, and VM-level elasticity and fine grained resource-level elasticity for cloud applications.

5.2.1 Static Resource Allocation

Scaling for non-cloud applications has long been studied. These kind of scaling techniques focus on transforming performance targets into underlying resources and statically allocate enough resources to the application's servers to meet the application's peak workload [39]. The drawback of this static resource allocation approach is that dynamic load changes and various capacity demands of the applications in their lifetime results in low resource utilization [40].

This resource provisioning approach has crucial relevance for the industry of mobile multimedia services. For example the services which are based on IP Multimedia Core Network Subsystem (IMS). These services require a non-negligible amount of resources and are shared by a large number of users (e.g. voice over IP, conferencing and messaging services). The usage of the resources in these services may significantly vary with current user load during different times. These services may have different loads during the day, during the night, during the week or special peak times in the year such as New-Year's-Eve. Thus, in order to meet the QoS levels promised at all times, the resources are over-provisioned statically and that leads to non-efficient resource utilization [41].

5.2.2 VM-level Elasticity

In contrast to non-cloud environments which use the static resource allocation approach, cloud applications have more focus on providing resources on demand. Cloud applications in shared virtual computing environments have dynamic load changes and apply dynamic on-demand approaches for resource scaling. They scale up and down when the user demand changes. This kind of scaling approach introduces a high elasticity requirement [39] as well. The reason is that such scaling, needs to decide how quick and in what granularity level the scaling up and down actions should be performed.

Most dynamic on-demand resource scaling approaches perform scaling by increasing or decreasing the number of VM instances that serve an application [41, 42, 43, 44]. According to [39], this kind of scaling is called VM-level elasticity. Most infrastructure cloud providers follow a policy-based mechanism to control the VM-level elasticity approach. This mechanism employs pre-defined policies (rules) to guide the application scaling. E.g. Amazon EC2 auto scaling [42] enables application owners to manually specify scaling rules giving the upper and lower bounds of the number of servers, the conditions to trigger scaling and the number of changed servers in each scaling [39].

The drawback of VM-level elasticity approach is that when the application is not using new created VMs efficiently, considerable computing resources will be wasted and this will incurs extra cost [39]. Also, creating, shutting down and removing the VMs dynamically at run time will increase the overhead.

This scaling approach is the most used approach for on-demand resource scaling in practice and is appealing for a wide range of applications for which scaling up of an application involves adding a new application server and hence an extra VM in the cloud context (each VM hosts one application server), and scaling down of an application involves removing a VM, e.g. applications which are based on multitier architectures, or server-side software platforms. But typically this is not the case for the applications that have a fluctuating demand where the load changes rapidly over time [39].

5.2.3 Fine Grained Resource-level Elasticity

There are other researches [39, 34, 35, 40] that focus on a fine grained scaling for cloud applications and VM-based data centers rather than VM-level elasticity. They focus on changing the VM capacity at the level of underlying resources (e.g. CPU, memory) instead of using VMs as basic units for dynamic resource provisioning. This is called a resource-level elasticity which resource allocation is at the granularity of underlying resource components (e.g. CPU, memory, I/O) [39]. They notice that VM-level elasticity is not always required and in some real word scenarios lightweight resource-level elasticity can be sufficient [39]. Also, resource-level scaling can be completed quicker than VM-level elasticity and has less overhead [39]. Fine grained resource-level elasticity by dynamic allocation of fine grained resources solves the problem of non-efficient resource utilization of the static resource allocation and the VM-level elasticity approaches.

There are different scaling algorithms proposed based on fine grained resource-level elasticity approach. Gong et al. [34] proposes a predictive fine grained resource-level elasticity approach that conducts accurate resource allocation using two techniques:

patterns-driven and state-driven demand prediction techniques. First it acquires knowledge about the application demand after the application starts. After getting a few resource demand samples, it starts making predictions. Resource allocation for scaling starts once it is confident in those predictions. If it finds a signature (they call repeating patterns as signatures), it uses the signature-based technique to make the predictions. It derives a signature from the pattern of historic resource usage, and uses that signature for predictions. Repeating resource usage patterns are often caused by repeating requests or iterative computations. If a signature is not found, which is often the case in applications without cyclic workloads, it will use the state-based technique to make predictions. Statistical state-driven technique is used to capture short term patterns in resource demand, and a discrete-time Markov chain is used to build a short-term prediction of demand for the near future. As resource consumption patterns change, the resource prediction models will be repeatedly updated.

Scaling based on demand predictions will introduce different errors namely under estimation and over estimation errors. Such scaling is not elastic because of over-provisioning and under provisioning problems as the result of under estimation and over estimation errors. Under estimation error happens when the predicted demand is less than real demand and this error may lead to Service Level Objectives (SLO) violation. Over estimation error happens when the predicted demand is more than the real demand and this error leads to non-efficient resource utilization. Shen et al. [35] proposes a proactive resource-level scaling approach based on demand predictions and addresses the set of under estimation and over estimations problems. Over-estimations waste the resources and are corrected by updating the online resource demand prediction model with true application resource demand data. They further improve the prediction mechanism by introducing two handling schema for the under

estimation errors in predictions, namely online adaptive padding and reactive error correction. The *prediction error correction schema* performs *online adaptive padding* that adds a dynamically determined cushion value to the predicted resource demand to avoid under-estimation errors. The *reactive error correction* schema detects under-estimation errors that are not prevented by the padding scheme and corrects them by considering an initial resource cap for each application VM. In online adaptive padding, under estimation errors are corrected by adding a small extra value to the predicted resource demand. The padding value is chosen based on the recent burstiness of application resource usage and recent prediction errors. In the error correction schema, the aim is to correct under estimation errors as soon as possible to prevent the application violating SLOs during resource under-provisioning. To do so, they raise the resource cap by multiplying the current resource cap by a ratio parameter until the under-estimation error is corrected. The value of this scale-up ratio is a value greater than 1 and is calculated by mapping the resource pressure and application SLO feedback to a value, considering the severity of the underestimation error. They mention that this scaling approach is application-agnostic and can achieve good prediction accuracy for a range of real world applications.

Han et al. [39] proposes a fine grained resource-level scaling algorithm which performs scaling based on the feedback collected from the application execution. They assume that QoS requirements are specified as the required response time in the SLAs. This response time includes a lower bound and an upper bound which is the range of response times acceptable by service users. They target multi-tier applications mainly where each tier's server is installed on separate dedicated VMs. They collect end-to-end response time of the application and each VM's resource utilization by monitors. The algorithm runs for each PM in the data center. The

algorithm triggers a Lightweight Scaling Up (LSU) when the observed response time is larger than the upper bound of the required response time and a Lightweight Scaling Down (LSD) if the observed response time is smaller than the lower bound of the required response time. The scaling process will be completed when the detected response time meets the required response time. No prior knowledge about the application is required and the algorithm applies an automatic reactive scaling mechanism based on measurements collected during application's run time. LSU algorithm conducts different levels of scaling with different priorities. The resource-level scaling first reduces the application's response time by increasing VMs' capacity using the available resources from the VM's hosting PM. The server's underlying resources scaled in this algorithm are namely CPU, memory and bandwidth. Each VM's resource utilization is collected periodically by monitors. In LSU, first the usage of resource r is compared with the threshold of utilization defined for scaling up that resource. If the usage has passed the threshold, one unit of that resource will be added to the VM in each scaling and the configuration of VM resources will be updated. The resource-level scaling has some constraints. For example, the available resources on a PM that can be allocated to a VM instance are limited. The LSU can be accomplished if constrained fine grained resource-level scaling can meet the response time target. Otherwise, a VM-level scaling is required to be performed too. This is done by adding a new VM instance for the application in a new PM. The LSD algorithm first tries to remove as many as VMs possible from an application when there are several instances given to an application. The algorithm first performs the VM-level scaling down until it is infeasible and removing a VM would violate response time target. This feasibility is checked using application profiling and workload predication techniques. After removing redundant VMs, LSD

algorithm will conduct the resource-level scaling down and removes units of resources from VM's resources which their utilization is lower than the lower bound threshold for resource utilization. When remaining resources allocated to a VM reach to the default minimum amount of resources allocated, resource-level scaling down will be stopped and no more resources will be removed from the VM. Based on the result of their emulation test they claim that this lightweight scaling approach can adapt to different fluctuating demands and meet response time expected from the application by performing the fine granularity scaling up (down).

Song et al. [40] proposes a two-tiered fine-grained resource allocation mechanism for on-demand resource provisioning which improves the efficiency of data centers. This work states that the existing techniques of turning on or off servers with the help of VM migration are not enough for the efficiency of data centers and optimized dynamic resource allocation methods are required to solve the problem. In contrast to other existing dynamic resource allocation mechanisms which only focus on local optimization within a server, this mechanism proposes a two-tiered on-demand resource allocation mechanism consisting of the local and global resource allocations. The reason for introduction of a global optimization is that local optimization cannot always lead to global optimization. This is true when a number of VMs distributed onto various PMs host the same application and resources are independently allocated to VMs by each PM's local optimizer, and this may result in unbalanced resource allocation among applications. However, as there is no technological support provided yet for the resource allocation from a PM to a VM residing in another PM, they only evaluate the local optimization algorithms proposed. They propose a set of algorithms to control the dynamic resource allocation among VMs according to the demand and quality goals of the hosted applications. The local on-demand resource allocation on

each PM optimizes the resource allocation to VMs within a PM. In order to avoid the huge interaction among applications hosted on VMs, there is an allocation threshold for each VM. They have implemented a Xen-based prototype, on a workload scenario reflecting the resource demands in an enterprise environment where a local resource scheduler controls resource allocation in each PM. It works in small intervals (e.g., 1 second) for each PM to quickly respond to the sudden changes of resource demand of applications in a timely manner. It dynamically adjusts the allocated resource to VMs according to their static priorities, resource utilizations, and the demand of the hosted applications. Resource demand of a VM at a time t is proportional to the request arrival rate to and denotes the activity of the hosted application on the VM. They point that resources demanded for and the resources allocated to a VM determine the quality of the application hosted on the VM. They use the resource utilization of VMs and the number of arrival requests during the last interval as a predictor for the next interval and the resources allocated to the VM will be adjusted based on that. As this algorithm triggers scaling based on resource utilization and not based on quality of application (like the algorithm proposed in [39]), it removes the need of continuously checking the quality of application and measuring the performance metrics. The quality of application indirectly comes to play in this algorithm by mapping the quality of application to an amount of resources that should be allocated to a VM for a specific demand.

5.3 Proposed Scaling Mechanism

In this section, we first provide details on which type of scaling approach can be used to fulfill the elastic scaling requirement based on the review we did in the previous

section. Then, we propose a scaling mechanism to be used by substrate providers for resource provisioning.

5.3.1 Discussion on a Scaling Approach for Cloud Conferencing Applications

The load of a cloud conferencing application dynamically changes over time. Thus, for cloud applications an on-demand resource allocation mechanism is required to solve the low resource utilization problem of static resource allocation in non-cloud applications. Among on demand elasticity approaches, the ones that are based on addition/ deletion of the VM instances to an application typically cause overhead and extra cost for applications which have fluctuating demands [39]. For this type of applications, a fine grained resource-level scaling approach will result to more efficient resource utilization. Cloud conferencing applications with rapidly changing demands are among those applications. Hence, an on-demand resource allocation mechanism with a fine grained resource-level scaling approach can improve resource utilization of cloud conferencing applications. In our cloud model, a conferencing application is composed from several conferencing substrates and to scaling to the demand of the application, each of underlying conferencing substrates should scale to the demand.

5.3.2 Description of the Proposed Scaling Mechanism

In the previous section, we concluded that a resource-level fine grained scaling approach can be appropriate to be used for scaling of cloud conferencing substrates, because of their fluctuating demand.

We reviewed different fine grained resource-level scaling approaches. Some of them [34, 35] predict the future demand for an application and scale based on the predictions. Predictive scaling approaches do not seem appropriate for scaling conferencing substrates. Since, finding a usage pattern or a signature to be used for prediction of the future demand, may not be the case for conferencing substrates because of their dynamic load changes. Using predictive scaling approach for conferencing substrates may lead to high severity of under estimation and over estimation errors, and consequently high under provisioning and over provisioning problems. Even in the predictive scaling approach which corrects the under estimation and over estimation errors [35], a lot of corrections can be required and the quality of application may get affected during correction of under estimation errors.

Algorithm proposed in [39] triggers scaling based on quality of application. Response time of the application is measured periodically and when it gets violated, scaling procedure will be triggered. Now if the utilization of any VM resources has passed the resource utilization thresholds, scaling up/down will be performed accordingly. Using such scaling approach for conferencing substrates introduces several complexities. One problem is that for each conferencing substrate, the appropriate performance metric which can specify the quality of application should be defined. The performance metric measured in [39] is the response time of application' main request which as the only performance metric defined. This is not the always the case for conferencing substrates. Different conferencing substrates have their own specific quality metrics. Some of them may have several quality metrics. e.g. for an audio mixer substrate with several QoS metrics of average packet loss, bit error rate, and average jitter. Second problem is that this approach requires checking the QoS of application continuously by measuring the performance metric. So using such

approach for scaling conferencing substrates introduces the need of monitoring systems for QoS metrics of different conference substrates. Third problem is that if the time duration that violation of QoS lasts is acceptable from the users or not. The forth problem is that, a conferencing substrate may be shared between several conferencing applications and using an algorithm which triggers scaling based on QoS of application may have its own complexities as there are multiple terminals (conference applications) that use the same conferencing substrate which may experience different qualities.

In [40], the scaling is triggered based on resource utilization. The resource utilization of the VM (CPU, Memory) and the demand (number of coming requests in period of time) of application are monitored. If the utilization of any resources passes the resource utilization threshold, scaling up/down based on the demand will be performed to meet the quality requirements of application.

Using a scaling mechanism for conferencing substrates which triggers scaling based on resource utilization, removes all the problems we mentioned an approach which triggers scaling based on the quality the application may introduce. The scaling of conferencing substrates may also need a custom support in some scaling cases. For example when scaling down is triggered, if the substrates still holds any ongoing conferences or has any active users, the substrate should not be totally deleted. To determine this, the application level information should be checked before taking a scaling action. Also we mentioned that a substrate can be shared among several applications. Each individual tenant expects the application to be scalable and actions of other tenants should not affect the performance of the application.

As none of existing scaling algorithms satisfies all of our requirements, we propose a new mechanism. This mechanism is derived from existing works. What we propose to be used as scaling mechanism for conferencing substrates is a scaling mechanism which is triggered based on the resource utilization of VMs. Resource utilization denotes the activity of application [40]. In this mechanism, we propose to scale unit by unit in terms of resources in the scaling iterations. A similar approach to work done in [39] which adds/ removes only one unit of resources in each scaling action. This way, the cost will be minimum because it just adds the minimum units of resources required [39]. This gives the flexibility to scale in the smallest scale possible with the smaller cost. Also, resources are typically charged based on their basic units (e.g. 1 GB memory or one unit vCPU) and is easier to design a pricing model based on the cost of the number of resource units used. The scaling mechanism proposed is a generic mechanism and can be applied to different type of conferencing substrates.

Before delving into explaining the proposed scaling mechanism, in the next sub section first we explain the assumptions that we have taken into account. After that, we explain the scaling mechanism in detail in the sub section after.

5.3.2.1Assumptions

In this section we explain the assumptions that are taken into account in this scaling mechanism. They are as follows:

- Only one conferencing substrate is deployed on each VM, e.g. a dial-out signaling, an audio mixer substrate, etc.). This way, the changes in resource

utilization can be reasoned to the activity of the only conferencing substrate running on that VM.

- Each T period of time (e.g. 1 minute) VM metrics (CPU, memory and I/O) are monitored. CPU, memory and bandwidth are the resources that their utilization significantly varies in a conferencing application with different user loads.
- For each type of resources (CPU, memory and I/O), two thresholds of utilization are defined which trigger scaling up and down. The upper bound threshold indicates high resource utilization (e.g. 80% CPU usage) which indicates additional units of resources should be added. The lower bound threshold indicates that utilization rate of the resource is low and redundant resource can be removed (e.g. 20% CPU usage).
- This scaling mechanism is applied to each PM in a datacenter. As a result, all the VMs hosted on a PM are monitored periodically.
- When performing a VM-level scaling up, the new VM will be added on a new PM and not to the same PM. This is to avoid having several instances of the same substrate given to same application on the same PM.
- Each PM reserves some resources in advance in order to have the required resources for performing a resource-level scaling up [39]. These resources can be used for scaling up without affecting the other VMs hosted on the PM. Please note that resource-level scaling up will be limited to the resources reserved on a PM. Excessive resource reservation on a PM is costly as well. So there is trade-offs between reservation cost and the cost incurred by VM-level scaling which should be further investigated [39].

- In order to avoid huge negative effects of the VMs hosted on the same PM to each other when competing for available resources, we assume that the resources allocated to a VM should be a value between minimum and maximum available resource thresholds of the VM [40]. (e.g. when the minimum available memory resource threshold of a VM is defined 1 MB and the maximum 4 MB, it means that 1) if a VM has only 1 MB memory, it will not scale down in terms of memory resources anymore and 2) the VM will not get a share of memory more than 4 MB from memory resources).
- Each VM starts with its basic resource configuration which is the minimum available resource threshold of the VM (the least resources that should be allocated to the VM all time).

5.3.2.2 Scaling Mechanism Step by Step

The procedure of resource allocation in the proposed scaling mechanism is shown in Figure 5-1. We explain the steps of this procedure one by one.

The mechanism starts by checking the utilization of VM resources on a PM. All the VMs on a PM are monitored and the resource utilization information is collected each T period of time. Dynamic resource provisioning of applications enables real-time acquisition/release of computing resources to adjust the applications performance [39]. Hence, choosing a value for T (e.g. 10 seconds) depends on how quick you need to respond to the sudden changes of resource demand (this can be decide by experimental evaluations). We name the set of monitored VM resources R and R includes CPU, Memory and network I/O. In the next step, the resource utilization of a resource R is compared with upper bound threshold and lower bound threshold for

resource R. The resource utilization of a resource R can be either less than the lower bound threshold, greater than the upper bound threshold or between this two values. If it is between these two values, no scaling action is required and the VM configuration will not change. If it is greater than the upper bound threshold, a scaling up operation on resource R should be performed. If the resource R utilization was less than the lower bound threshold for resource R, a scaling down operation will be performed. We explain scaling up and down procedures below:

Scaling up:

We mentioned that if the resource utilization of a resource R is greater than the upper bound threshold, a scaling up operation on resource R should be performed. By resource R we mean the type of resource which has experienced high utilization and triggered the scaling process. It can be either CPU, memory or I/O or a couple of them. E.g. if the utilization of CPU is greater than 0.8 (the upper bound threshold defined for CPU utilization), a scaling up operation on VM's CPU will be performed. Each time in a resource-level scaling up , one unit of resources will be added to VM. Now if the PM has any resource R available and if the VM has not yet reached to the maximum available resource threshold for resource R, a resource-level scaling up can be performed by adding one unit of resource R to the VM. Otherwise a new VM will be created on another PM. This new VM starts with its basic resource configuration. As a result, when resource-level scaling up cannot be responsive enough to the demand, a VM-level scaling up will be performed. This happens when we already have allocated the maximum resources available to the VM, but the resource utilization is still higher than the upper bound threshold. In this case we cannot

perform any more resource-level scaling and a VM-level scaling up will be performed to add a new VM to serve the application.

Scaling down:

If the utilization of resource R in a VM is less than the lower bound threshold for resource R, a scaling down operation will be performed. To decide for the level of scaling down, either a resource-level or a VM-level scaling down, first some conditions should be checked. If this VM belongs to a group of VMs serving the same conferencing application and if there is no ongoing execution processes in on the VM, this VM is considered redundant and it will be removed by performing a VM-level scaling down. Else, if the VM is the only VM serving a conferencing application, or if the VM has any ongoing processes (e.g. active conference, active user, etc.) which should not be interrupted, a resource-level scaling down will be performed by releasing one unit from the resource R. For example, assume that there is a VM running an audio mixer substrate to serve a dial-out audio conference application and this VM is one of the 2 audio mixer VMs given to the same conference application. If this VM is serving any active conference end user, it cannot be removed even if it belongs to a group of VMs serving the same application. Only a resource-level scaling down can be performed on this VM. This can be checked by using application profile and saving application level information [39].

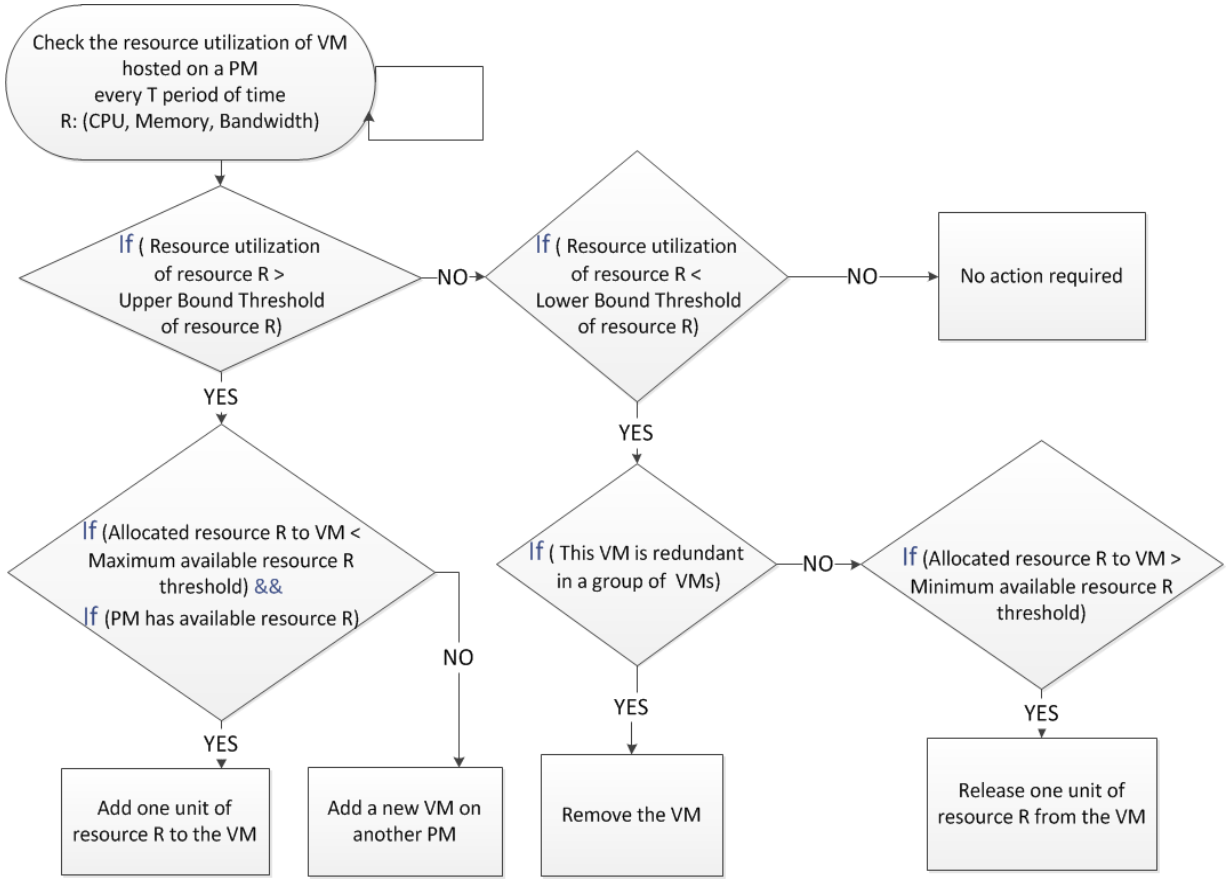

Figure 5-1. Resource Allocation Procedure for scaling

The Pseudo code of the scaling mechanism is also provided below:

Scaling Mechanism

Input: Lower bound utilization thresholds of resources, Upper bound utilization thresholds of resources

1. **Begin**
2. **while** (VM is running)
3. **Monitor** utilization of VM resources R (CPU, memory, I/O) once every T period of time;
4. **if (**utilization of resource R **>** Upper bound utilization threshold of resource R), **then** Scale UP
6. **else if (**utilization of resource R **<** Lower bound utilization threshold of resource R), **then** Scale Down
7. **Else** No action is required
8. **End**

Scale Up

Input: Maximum resources available threshold, Resources available on the PM

1. **Begin**
2. **if** ((Resources allocated to VM**<** Maximum resources available threshold) **and (**PM. has available resource R **=** true)), **then** add one unit of resource R to VM.
3. **Else** create a new VM on another PM
4. **End**

Scale Down

Input: Minimum resources available threshold**,** Application profile, application level information

1. **Begin**
2. Check Application profile and application level information
3**. if** (VM is redundant **=** true), **then** Remove the VM
3. **Else if** (Resources allocated to VM**>** Minimum resources available threshold), **then** remove one unit of resource R from the VM
4. **End**

5.4 Chapter Summary

In this chapter we discussed how the elastic scalability requirement can be satisfied by substrate providers.

We started by explaining the problem that substrate provider need to use a proper scaling approach for scaling of conferencing substrates to meets the elastic scaling requirement. We reviewed different scaling approaches for cloud and non-cloud environments. We explained existing scaling approaches for cloud environments in two different categories of VM-level scaling approaches and fine grained resource-level scaling approaches. Based on the review, we concluded that the fine grained resource-level scaling approach is the most resource efficient scaling approach because scaling is in the granularity of underlying resources. Then we reviewed

different existing scaling algorithms which are based the fine grained resource-level scaling approach. Based on the reviewed algorithms, we proposed a fine grained resource allocation mechanism to be used by substrate providers for scaling. This mechanism is still in a preliminary level to be used as a skeleton for designing a scaling algorithm. It should be further tested and evaluated.

Chapter 6

6 Validation: Prototype and Evaluation

In chapter 4, we proposed a general architecture for a virtualized infrastructure for cloud based multimedia conferencing applications. In this chapter we focus on designing the software architecture, validating the architecture with an implemented prototype and evaluating the measurements collected from prototype.

This chapter starts by presenting the overall software architecture. Then we describe the prototype we implemented as a proof of concept to validate architecture. After that we present the performance measurements collected from the prototype and evaluate them. And finally, we summarize this chapter.

6.1 Software Architecture

In this section, first we present the overall software architecture proposed for virtualized infrastructure for cloud-based conferencing applications.

6.1.1 Overall Software Architecture

Figure 6-1 shows the software architecture proposed for the virtalized infrastructure for cloud-based conferencing applications. Architectural components shown in the general architecture are broken to smaller software modules which carry different functionalities.

In the next four following sub-sections we explain the software components of IaaS layer, software components of SubaaS layer, the operational procedures and the communication

interfaces.

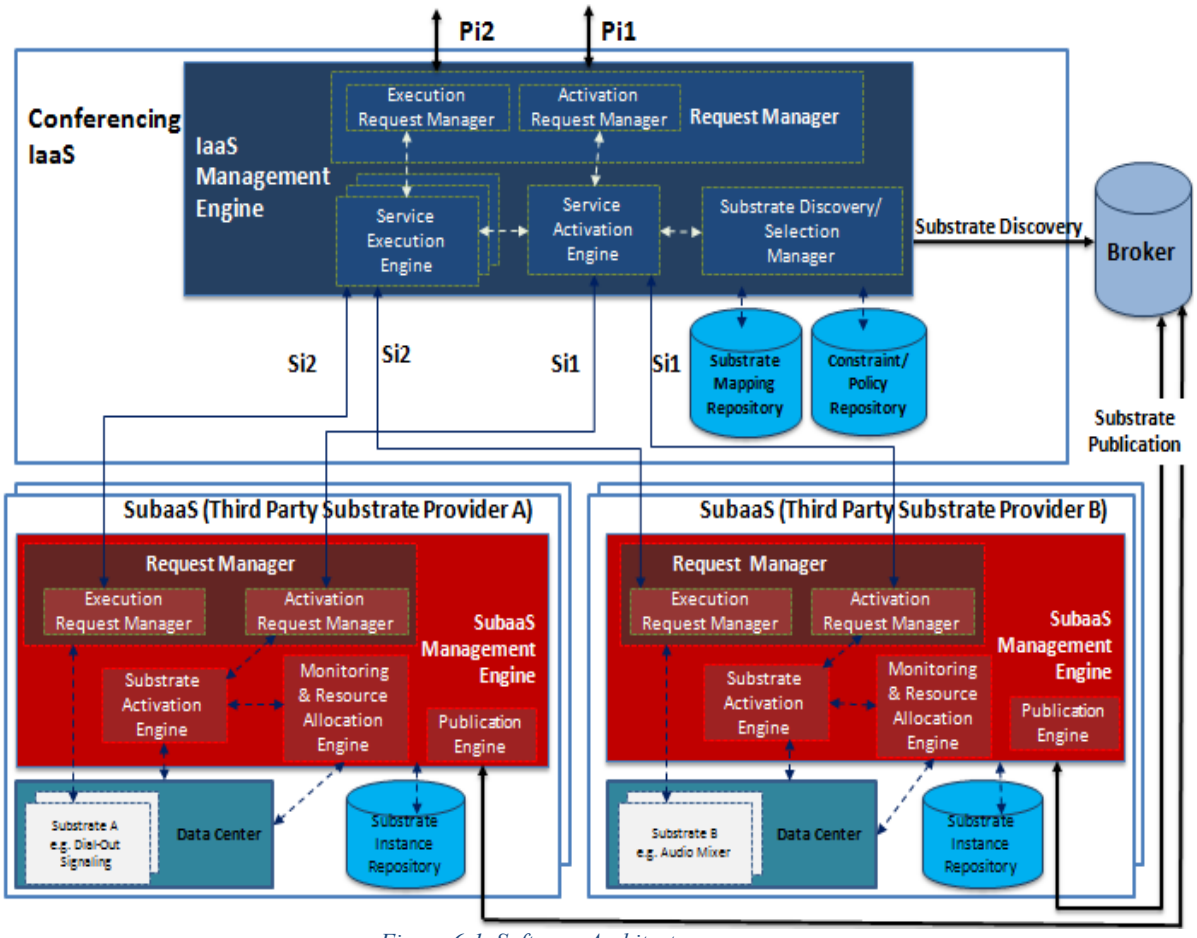

Figure 6-1. Software Architecture

6.1.1.1 IaaS Layer Software Components

In the Conferencing IaaS, the components are: IaaS Management Engine. IaaS Management Engine includes Request Manager, Service Execution Engine, Service Activation Engine and Substrate Discovery/Selection Engine.**Request Manager:** this module handles the requests coming from the PaaS layer. It includes two request manager entities:

- **Activation Request Manager**: realizes the activation interface provided to the cloud platform provider. It receives service activation requests from PaaS and sends them to substrate activation engine to be further processed.
- **Execution Request Manager**: receives execution requests from PaaS and sends them to the corresponding service execution engine instance to be processed (e.g. a using a content based routing with service identifier to proxy the requests to the corresponding service execution engine instance).

- **Service Activation Engine**: each service activation request may ask for activation of one or several substrates. Upon receiving a service activation request, service activation engine searches for the substarates that match to the criteria given along the request. Service activation engine cooperates with Substrate Discovery/Selection Engine to find the substrates from the broker and after discovering them sends substrate activation requests to each of substrate activation engines of the corresponding SubaaSes. After receiving the successful substrate activation notifications from all of SubaaSes, service activation engine of IaaS creates a Service Excution Engine instance for coordination of execution of this service and notifies the PaaS that the service is activated, giving the endpoint address for service execution. If any of the subsrate providers was not able to provide the requested substrate and returned a negative response to the request, substrate activation engine will try to find an alternative substrate and repeat the substrate activation procedure. PaaS may also request for deactivation of a service. In this case substrate activation engine contacts the corresponding substrate providers to dectivate the substrates.

- **Service Execution Engine:** the service execution engine is responsible for managing and coordinating the execution of a service. It realizes the execution interface provided to the platform provider. It translates the execution requests received to the requests that will be sent to the target substrates. It uses the substrate mapping repository to find out the information of target substrates since these substrates can be provided by different substrate providers and can have several instances. Substrate Execution Engine of IaaS is like a mediator between PaaS and SubaaS for redirecting the execution requests to target substrates during the execution of a service.
- **Substrate Discovery/ Selection Engine**: interacts with the broker to find the substrates through the RESTful discovery interface provided by the broker. After finding the proper substrate which match to the criteria given in the discovery request, it returns the information of substrates and their substrate providers to the service activation engine.

6.1.1.2 SubaaS Layer Software Components

In SubaaS layer, the software components of SubaaS Mangemnet Engine are Publication Engine, Request Manager, Substrate Activation Engine, and Monitoring and Resource Allocation Engine.

- **Publication Engine:** this component is used for publication of substrates to the broker through the RESTful interface provided by broker. The description of the substrate will be sent along publication request and will be saved in the broker.

- **Request Manger:** this module handles requests from the IaaS layer. It includes two request manager entities:
  - **Activation Request Manager**: realizes the activation interface provided to the IaaS provider. It receives substrate activation requests from IaaS and sends them to substrate activation engine of SubaaS to be further processed.
  - **Execution Request Manager**: It receives execution requests from IaaS and sends them to a substrate instance to be processed since there can be several instances of a substrate given to a service. The execution request manager dispatches the execution requests to destined substrate instances. It may use a load balancing algorithm for distributing the workload across multiple substrate instances which serve the same conference service.
- **Substrate Activation Engine:** upon receiving a substrate activation request, substrate activation engine checks the availbility of resourses for provisioning the reqiuered substrate and then creates an instantance from substrate. Substrate activation engine has access to the datacenter of SubaaS and uses a virtualization client to connect to a server/ physical machine in data center to communicate for creation of substrate instances (more precisely, creating the VMs which host the substrates).
- **Monitoring and Resource Allocation Engine:** realizes on demand scaling support and implements the scaling mechanism proposed for scaling conferencing substrates in chapter 5. It monitors the resource usgae of substrate instances and uses the management APIs provided by the virtualization framworks used in data center to manage the resource allocation.

6.1.1.3 Operatinal Procedures

Here we illustrate the service activation and service execution procedures based on the proposed software architecture for a dial-out audio conference application.

Figure 6-2 shows the sequence diagram of service activation. The service activation process is initiated by PaaS to activate the conference application.

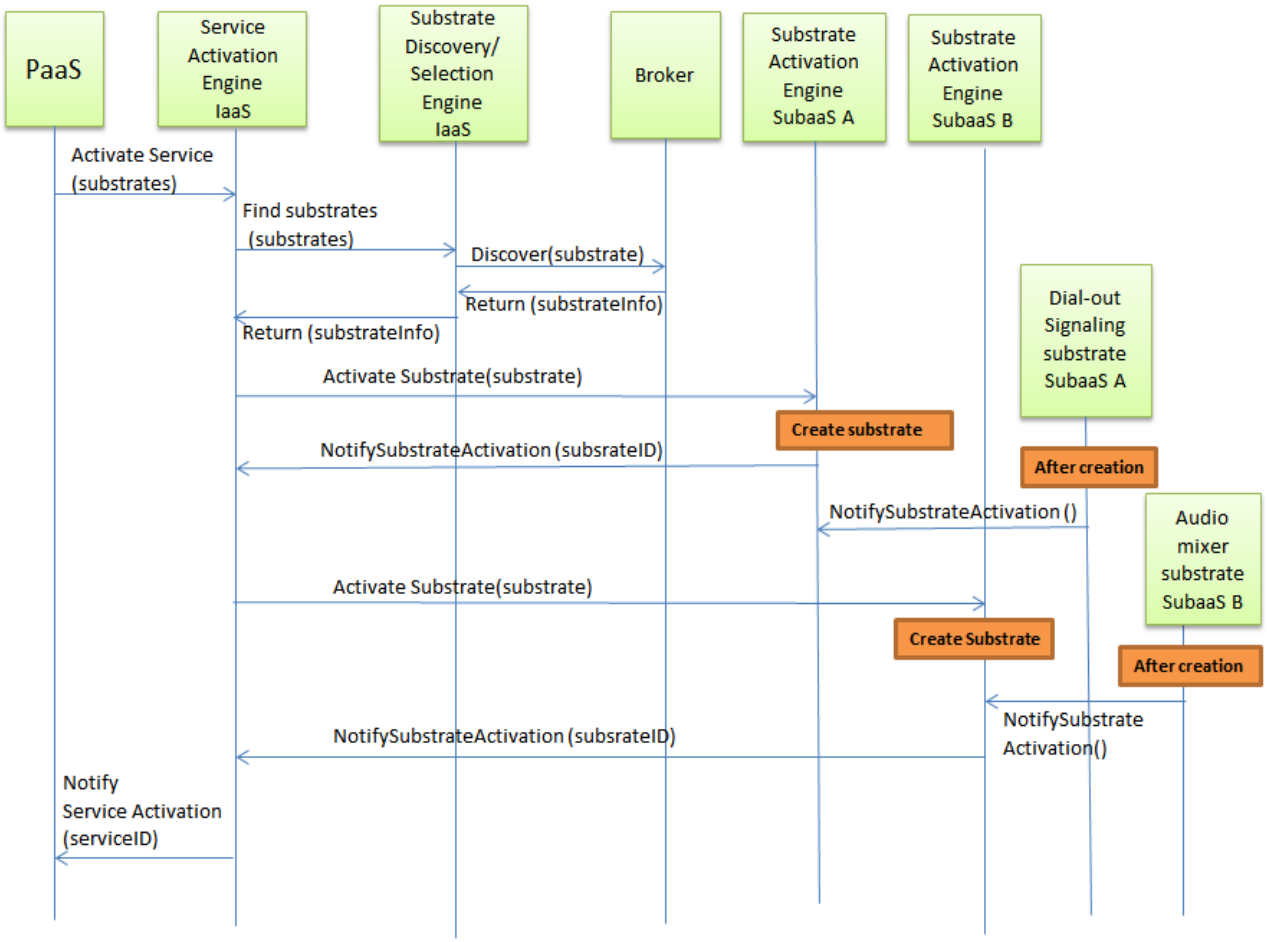

Figure 6-2. Service Activation Procedure

Figure 6-3 shows the sequence diagram of service execution for a *Create Conference* request. The request is initiated by PaaS when a conference administrator asks for creation of a conference.

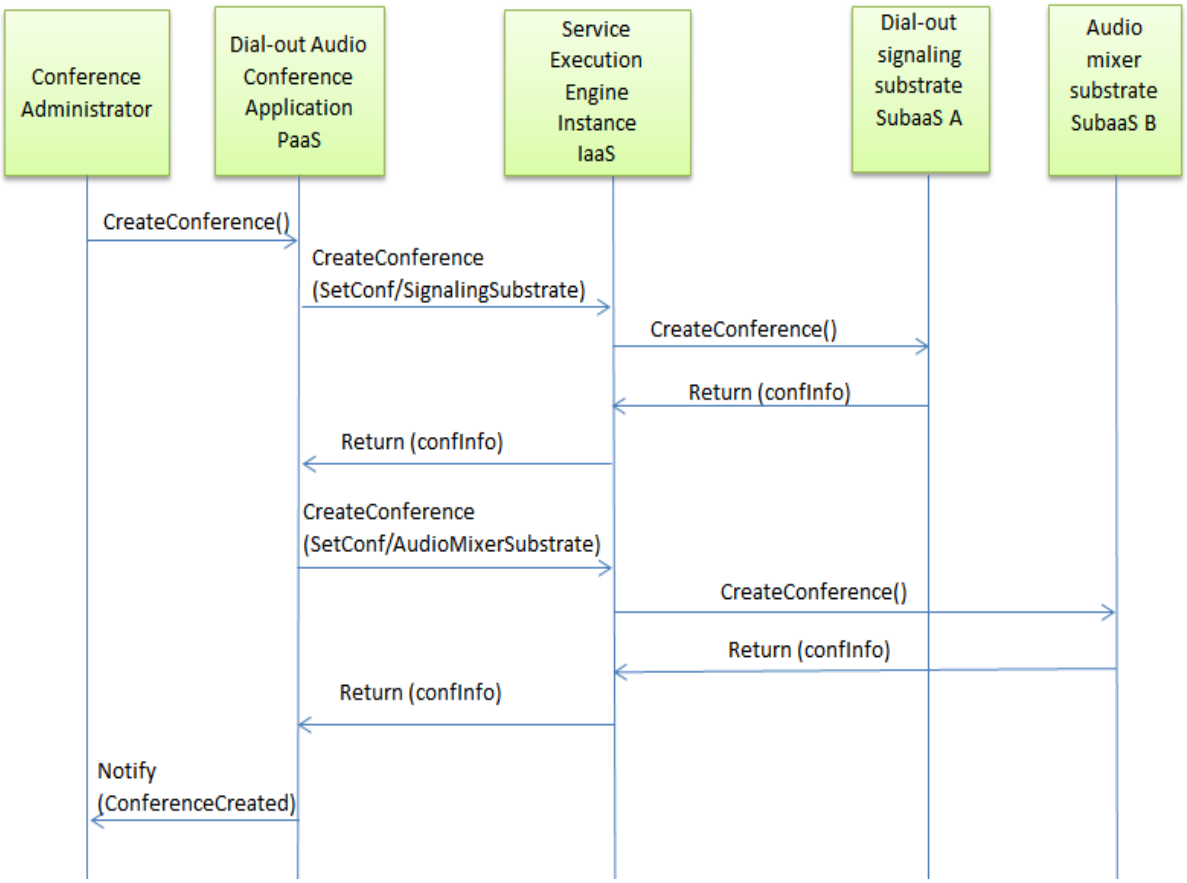

Figure 6-3. Service ExecutionPprocedure (creating a conference)

6.1.1.4 Communication Interfaces

The communication interfaces shown in the software architecture between cloud layers are:

- **Pi1 interface:** this is the REST based interface between conferencing PaaS and conefrencing IaaS provided for service activation. Conferencing PaaS is a REST

client to the REST server provided by IaaS (more precisely the REST server provided by substrate activation enigine of IaaS).

- **Si1 interface:** this is the REST based interface between conferencing IaaS and conefrencing SubaaS provided for service activation. Service activation engine of IaaS is a REST client to the REST server provided by substrate activation engine of SubaaS to send substrate activation requests. And, substrate activation engine of SubaaS is a REST client to the REST server provided by service activation engine of IaaS to send substrate activation notifications.
- **Pi2 interface:** this is the REST based interface between conferencing PaaS and conefrencing IaaS provided for substrate execution coordination. Conferencing PaaS is a REST client to the REST server provided by IaaS for substrate execution (more precisely the REST server provided by a service execution engine instance in IaaS).
- **Si2 interface:** this is the REST based interface between conferencing IaaS and conefrencing SubaaS provided for execution of a substrate. Conferencing IaaS is a REST client to the REST server provided by a substrate instance.

6.2 Prototype Design and Architecture

As a proof of concept we have implemented a prototype to validate the proposed architecture. In the following subsections, first we describe the implemented scenario, and then we present a high level description of prototype. After that we present the prototype architecture.

6.2.1 Implemented Scenario

As for the prototype, we have implemented the dial-out audio conference application scenario where we have a conferencing service provider that uses the conferencing PaaS to create and activate a dial-out audio conference application and offer it as a conferencing SaaS. Service provider uses a service creation tool of PaaS and choosing two substrates of dial out signaling and an audio mixer creates a dial-out audio conference application. Figure 6-4 shows a simple Graphical User Interface (GUI) for service creation. At this step, the conference application is not activated yet. More precisely, the back end substrate resources required for the execution of the application are not activated yet and only the composition of substrates is done. Later, service provider uses a service activation tool provided by PaaS to activate the conferencing application created. Figure 6-5 shows a simple activation GUI where service provider request for activation of application.

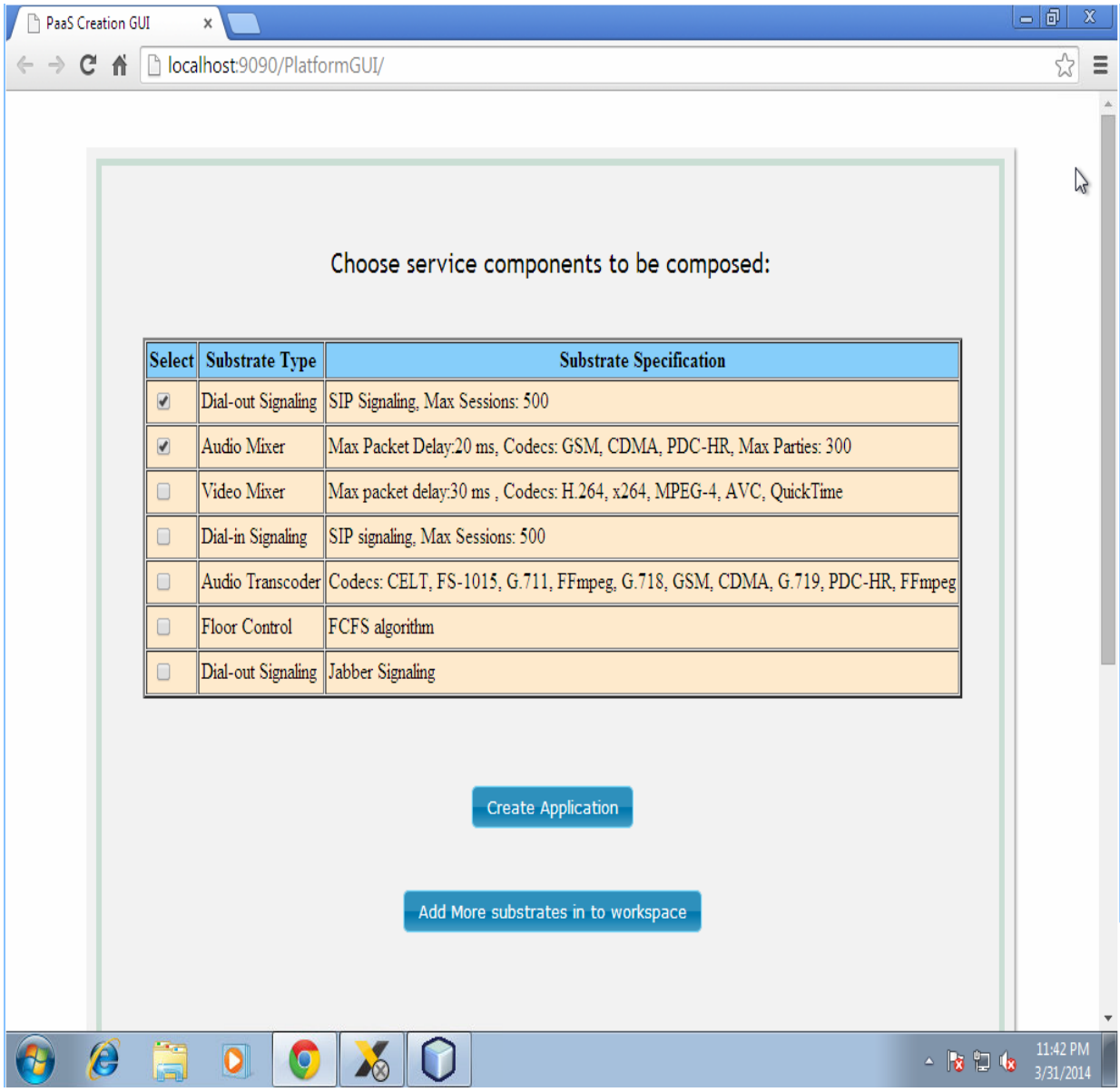

Figure 6-4. Service Creation GUI by PaaS

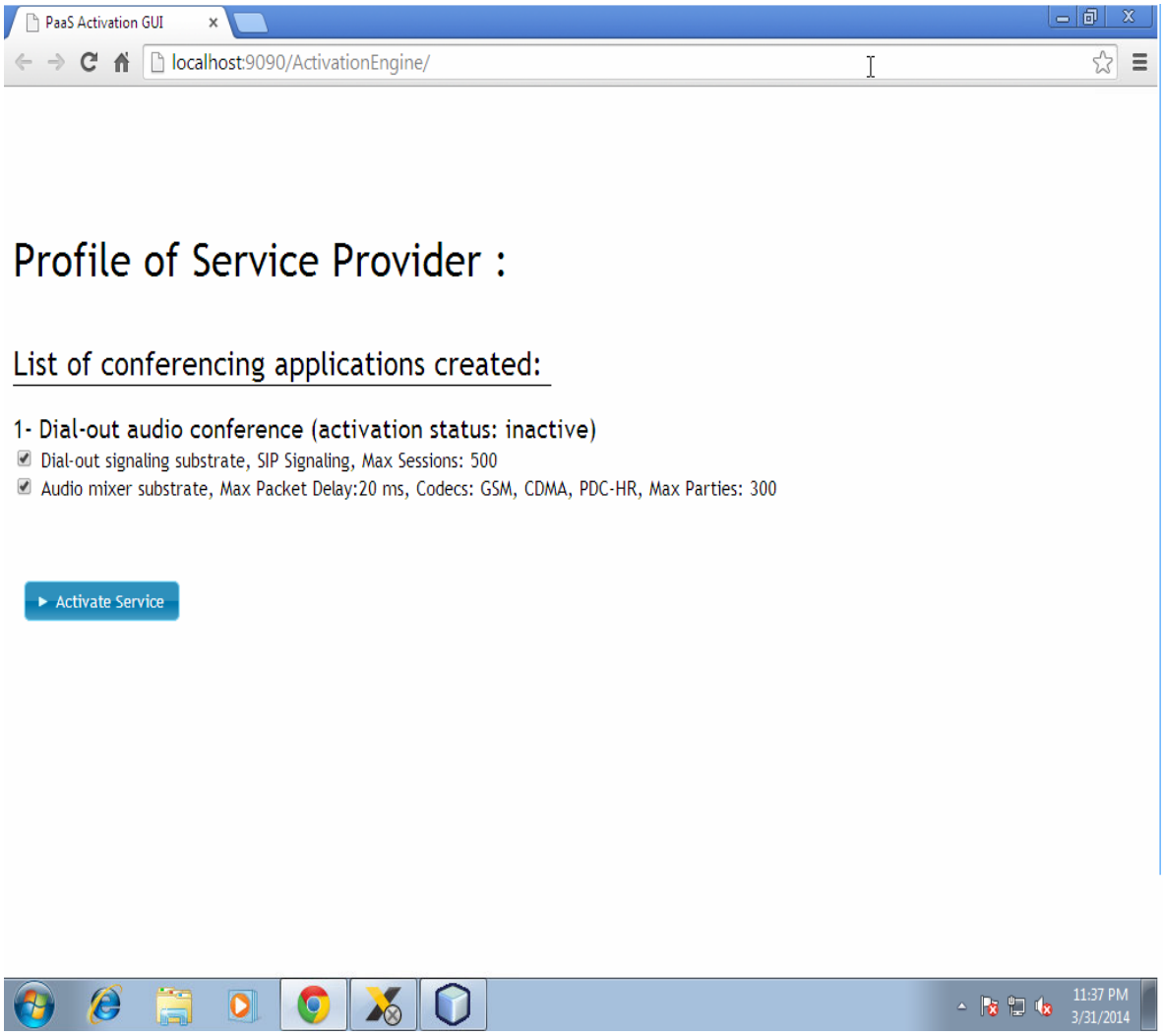

Figure 6-5. Service Activation GUI by PaaS

IaaS receives this request and interacts with corresponding substrate providers to ask for substrates. The substrates providers create instances from substrates and send back the required information for execution of substrates (e.g. the endpoint address for substrate execution, a signaling domain address that conference end users can register themselves with it, etc.). After receiving successful substrate activation notifications from substrate providers, infrastructure provider creates the execution interface required at the infrastructure level, using the execution interfaces of substrates and the substrates'

execution endpoint addresses. This interface will be called by PaaS during the execution of conference application. Then infrastructure provider notifies the platform provider that substrates are activated, giving the required information for service execution (e.g. endpoint address for service execution). Platform provider receives the successful service activation notification from infrastructure provider, configures the created application and deploys the conference application.

After activation of application the conference application is ready to be used. The conference application provides a GUI on the web (figure 6-6) to be used by a conference administrator to manage the conference application through it, e.g. to view the list of users (figure 6-7), to create conferences (figure 6-8), add participants (figure 6-9) to the conferences, etc.

Figure 6-6. Conference Application GUI- Initial Screen

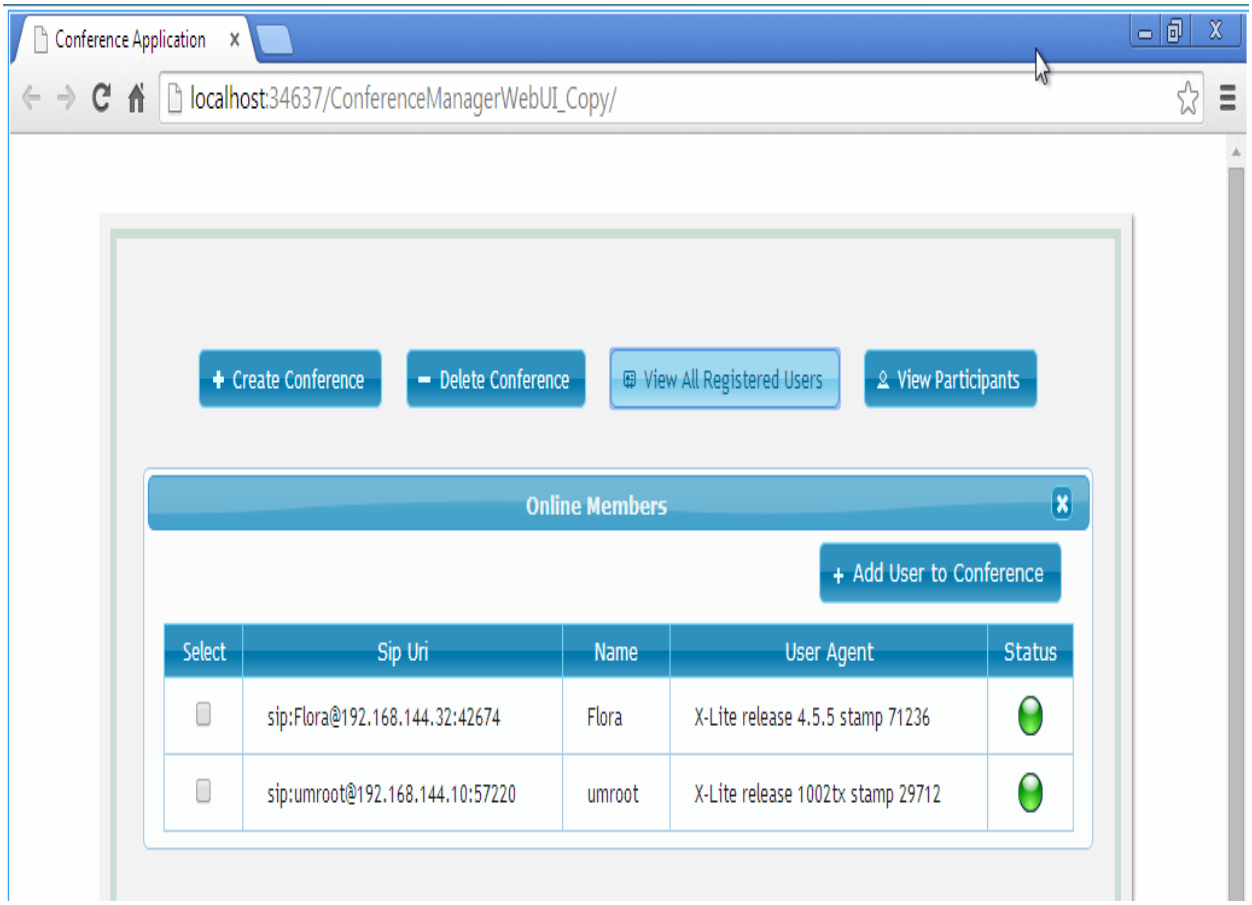

Figure 6-7. Conference Application GUI- View Registered End Users

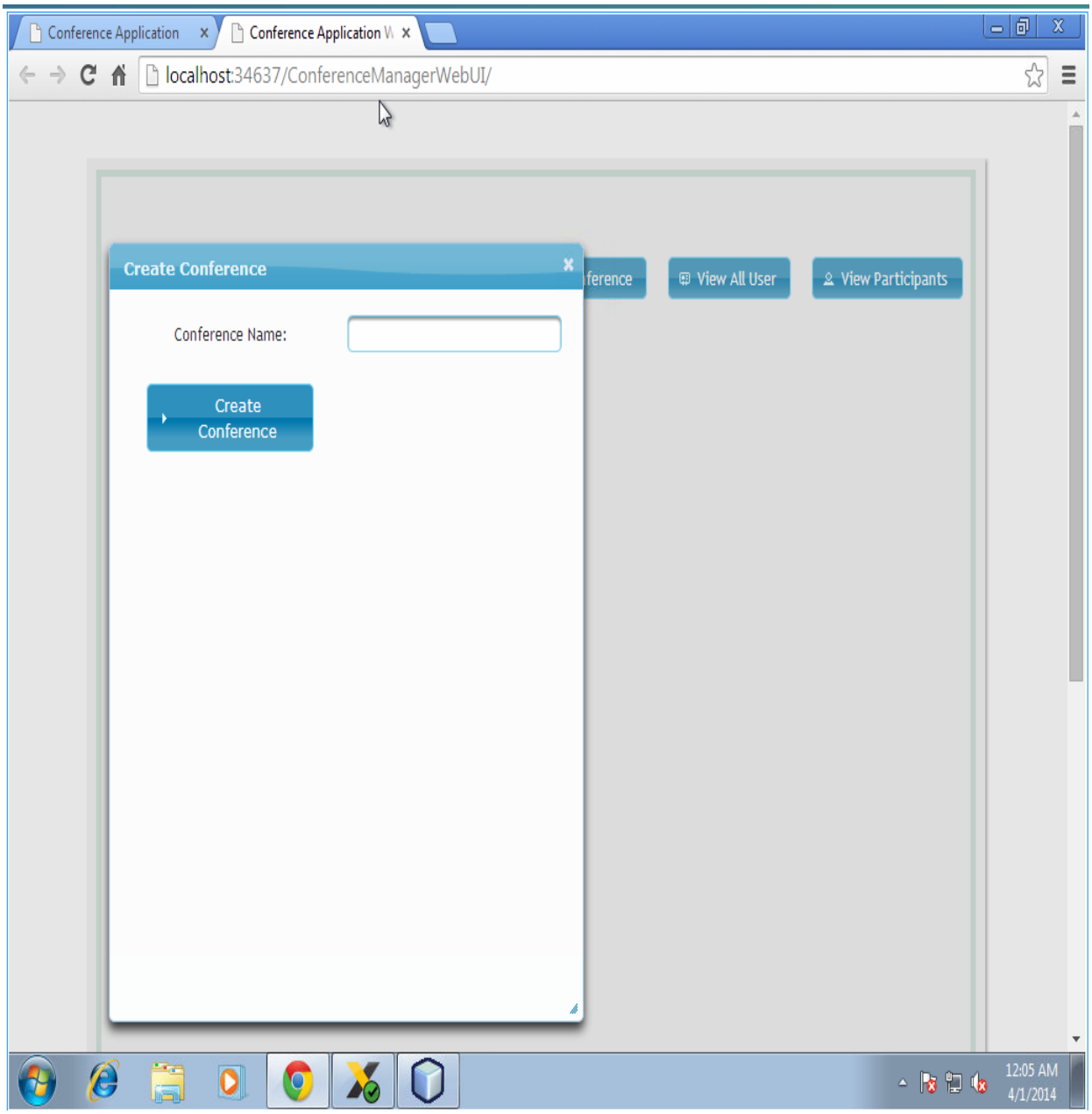

Figure 6-8. Conference Application GUI- Create Conference

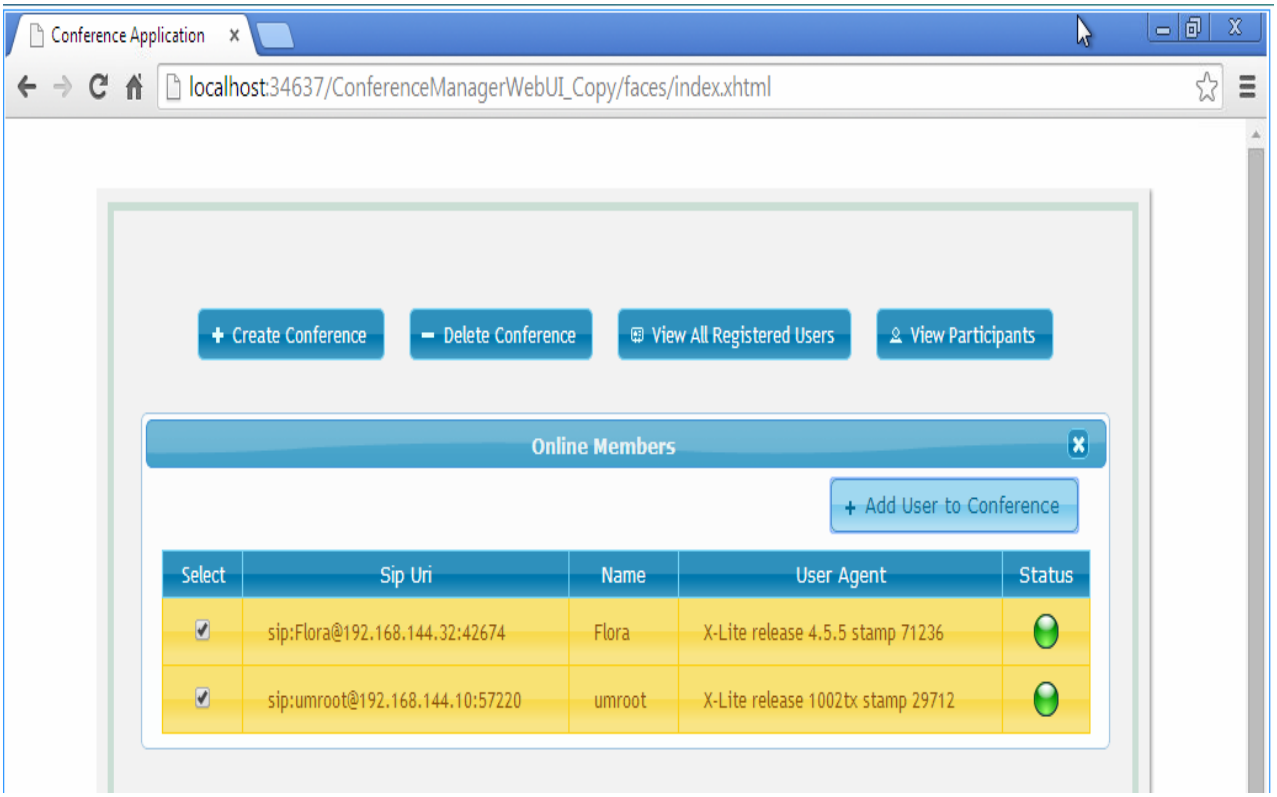

Figure 6-9. Conference Application GUI- Add Participant

6.2.2 Prototype High Level Description

In this section first we explain the scope of prototype, the assumptions and design decisions that we have made for the prototype.

6.2.2.1 Scope

The roles involved in this prototype are: a conferencing service provider, a conferencing platform provider providing the conferencing PaaS, a conferencing infrastructure provider providing the conferencing IaaS, and two substrate providers A and B providing dial-out signaling and audio mixer substrates.

Service creation/composition is related to the PaaS layer and is out of scope of this thesis. Just for the prototype we provide a simplified web GUI to illustrate the initiation of creating a conferencing service. We also provide a simple web GUI to allow the service provider initiate the activation process through it.

Substrate providers statically allocate resources to the substrates and there is no dynamic resource allocation management mechanism used in the prototype (this will be introduced as a future work).

6.2.2.2 Assumptions

The assumptions that we have made for the implemented prototype are as follows:

- Substrate provider A offers dial-out signaling substrate as SubaaS and Substrate provider B offering audio mixer substrate as SubaaS. The signaling substrate performs SIP signaling.

- The two substrates given by substrate provider A and substrate provider B correspond to the substrates composed for creation of the dial-out audio conference application.

- Infrastructure provider knows that substrate provider A and substrate provider B are offering two substrates which match to the requested substrates by the PaaS and therefore infrastructure doesn't require to check the broker. We assume that infrastructure provider has the required information for the activation and execution of these substrates (e.g. the invocation interface for substrate activation,

the end point addresses to send substrate activation request and also the invocation interfaces for execution of substrates).

- We have several conference end users as participants and a conference administrator to manage and control the conference application.

6.2.2.3 Design Decisions

The signaling communication protocol used to perform signaling actions for the prototype is SIP. SIP is a signaling protocol used to establish and manage multimedia IP sessions [52]. It is an application-layer control protocol for creating, modifying, and terminating sessions with one or more participants (unicast or multicast sessions) in internet telephone calls, multimedia conferences, and instant messaging applications. SIP allows participants to agree on a set of compatible media types [53]. For several reasons we have decided to use SIP:

- SIP is an IETF standard and can be used with other IETF protocols to build a complete multimedia architecture. Protocols such as RTP for transporting real-time data and providing QoS feedback, and the Session Description Protocol (SDP) for describing multimedia sessions [53].
- SIP is the major signaling protocols used in Voice over IP (VoIP). It is accepted as a 3GPP signaling protocol and creates a more robust standard compared to the H.323 [56].

- It is an open standard and there are several implementations of SIP signaling servers available. There are also many free sip clients available compatible with different operating systems [56]
- SIP is less complex than other signaling protocols (e.g. H.323), and offers text-based protocol encoding which is human readable [54]. Establishing a connection using H.323 takes about three times the data and turnarounds compared to when using SIP. H.323 uses several protocols and more ports need to be opened in a firewall to enable H.323 traffic through, while SIP is a single protocol and only one port has to be opened for SIP traffic [55].

Conference end users in the prototype use SIP softphones to get the calls and participate in the conferences. They need to be first registered with a SIP signaling server. Figure 6-10 shows an end user using a SIP softphone registered with the dial-out signaling substrate.

Figure 6-10. A SIP client (X-Lite softphone)

6.2.3 Prototype Architecture

In this section we present the prototype architecture. First we explain the implementation scope of the prototype. Then we give a detailed description of the prototype architecture and explain the software tools used in the prototype. After that we provide the procedure of application activation and two procedures from application execution procedures. Lastly we explain the environmental setting of the prototype.

6.2.3.1 Implementation scope of prototype

As for the prototype we have implemented a subset of components from the proposed software architecture. In this section we explain which components are implemented and which are not.

For the IaaS layer we have implemented the service activation engine for activation of two substrates and a service execution engine instance to redirect the execution requests from PaaS to SubaaS. The substrate discovery/selection engine is not implemented as we assumed that IaaS already knows the substrates and will not search into the broker to find them. We have not used any activation request manager as there is only one activation request performed. We also have not used any execution request manager as there is only one service execution instance which receives the execution requests directly.

For the SubaaS layer we have implanted the substrate activation engine for activation of substrates. We have not implemented any substrate publication engine as substrates are already published in this step. We have not used any activation request manager as there

is only one activation request performed. We do not use any execution request manager as there is only one substrate instance which receives the execution requests directly. Monitoring and resource allocation engine is part of future work for the resource allocation management and is not implemented.

Furthermore, we have provided a PaaS layer which contains a simplified web GUIs to illustrate service creation, an application activation engine which comes with a web GUI. It is the PaaS layer which also hosts and runs the conference application.

6.2.3.2 Detailed Prototype Architecture

Figure 6-11 shows the architecture of the prototype. We have used a Xen server for the virtualization of substrates. We explain the components in prototype architecture:

- **Application Activation Engine**: a web application with a GUI provided by PaaS for service provider to request for activation of the dial-out audio conference application. Upon activation request by service provider, application activation engine sends the information of dial-out signaling and audio mixer substrates with a REST request to service activation engine provided by IaaS. Application activation engine is a REST client to the REST server provided by service activation engine of SubaaS.
- **Application Execution Engine**: which run the conference application and is provided as a web application with a GUI to be used by conference administrator. It contains the workflow of application and hosted on PaaS. It is a REST client to service execution engine of IaaS. The details about composition of substrates is

out of scope of this thesis and we only provide a simplified PaaS layer to use it for validation of our infrastructure. By having the REST interfaces of two substrates and knowing the sequence of execution of substrates, we have hardcoded the workflow of application.

- **Service Activation Engine:** provided as web application in IaaS to receive the service activation request from application activation engine of PaaS and send two activation requests to substrate activation engines of dial-out signaling SubaaS and audio mixer SubaaS. It is a REST client to the REST server provided by substrate activation engine of SubaaS.
- **Service Execution Engine:** a web application which contains the REST interfaces of both dial-out signaling and audio mixer substrates to redirect the execution requests from IaaS to the target substrate instances. It is a REST server for application execution engine of PaaS and a REST client for substrate instances.
- **Substrate Activation Engine:** receives the substrate activation request coming from substrate activation engine of IaaS and creates one instance of substrate (one VM from predefined templates on the Xen server which hosts the substrate using XML-RPC (XML based Remote Procedure Call) based Xen server APIs).
- **Dial-out signaling VM**: a VM running the dial-out signaling substrate, provided by SubaaS A (a substrate instance given to the dial-out audio conference application).
- **Audio Mixer VM:** a VM running the audio mixer substrate, provided by SubaaS B.

- **X-Lite softphones:** the SIP clients used by conference end users to receive/send signaling messages in order to join to a conference and send/receive audio.

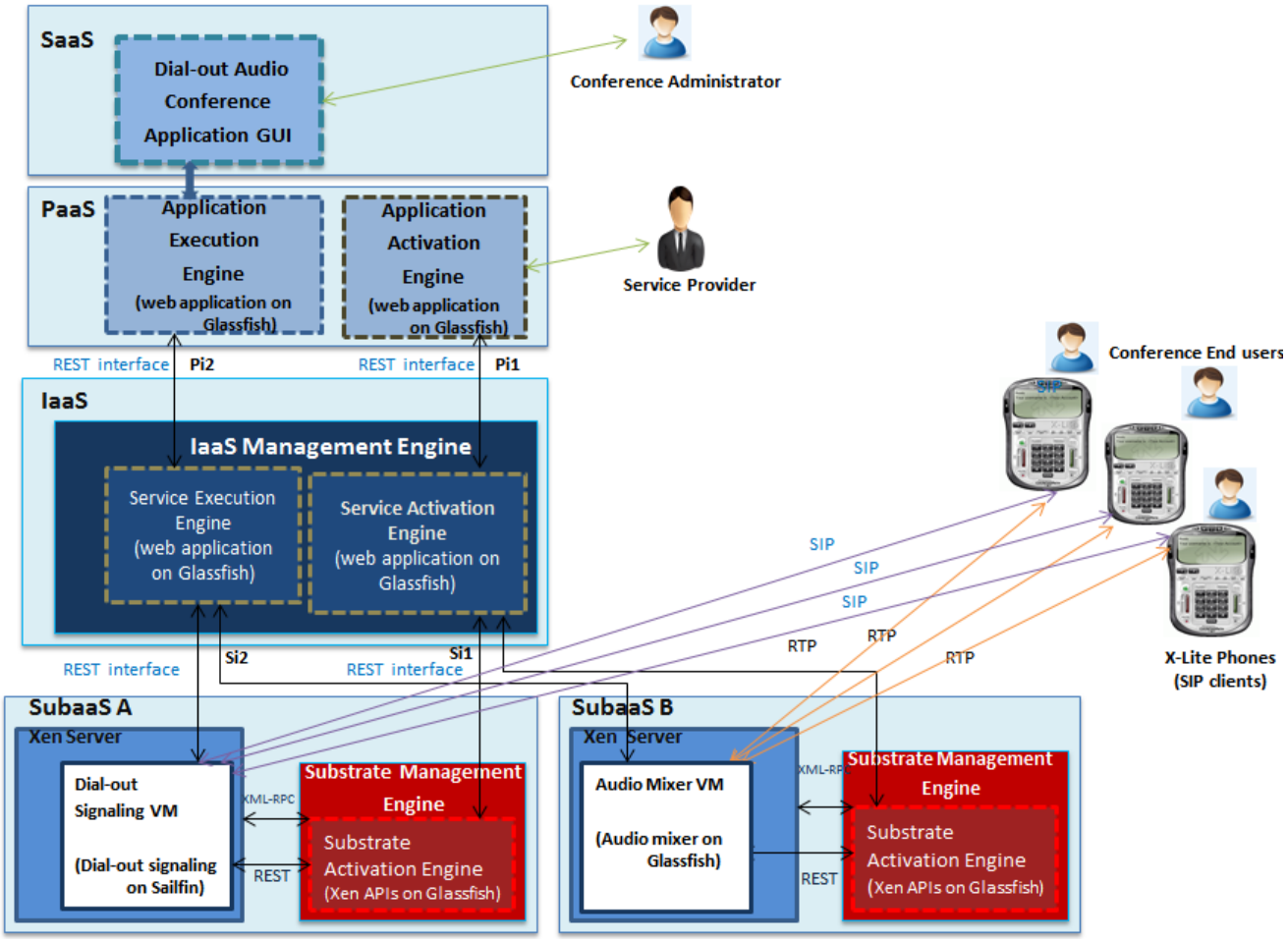

Figure 6-11. Prototype Architecture

6.2.3.3 Software Tools

In this section we explain the software tools and technologies used in the implementation of prototype.

6.2.3.3.1 Java

Java is the programming language used for the development of the software component in the prototype.

6.2.3.3.2 NetBeans

The Integrated Development Environment (IDE) used for the development of the java components is NetBeans (vesion 7.1).

6.2.3.3.3 Jersey

All these REST interfaces between SaaS, PaaS, IaaS and SubaaS in the prototype are implemented using Jersey. Jersey [45] provides an open source framework for RESTful web services. It has support for Java APIs for RESTful Services (JAX-RS). Libraries provided in Jersey contain implementation for both client and server sides of RESTful web services. JAX-RS is standard specification (JSR 311) for RESTful web services from the Java Community.

6.2.3.3.4 Xen Server

We have used the Xen Server of our laboratory to provide a test bed for our prototype and virtualize the substrates. It has Citrix Xen Server 6.1 installed with 12vCPUs (Intel(R) Xeon(R), Speed: 2532 MHz), 4 network interface card (NIC 0-3 with Speed 1000 Mbit/s), 600 GB local storage and 14326 MB memory.

Xen server provides XML-RPC based APIs for control and management of VMs during their life cycle. Java binding of Xen's XML-RPC based APIs is used for the VM management operations (creating, starting, shutting down, deleting, etc.). At the activation time, substrate VMs get created from predefined templates. The VMs are configured in a way to run the substrate at the start up (init scripts to run dial-out signaling and audio mixer programs at the boot time) and send notification back to the substrate activation engine by a REST POST request to notify that the VM has started and is running.

6.2.3.3.5 Medooze Multi Conference Server

For the dial-out signaling substrate, we reused an existing conferencing server which performs SIP signaling and is implemented by SIP servlets. SIP Servlet API defines a high-level extension API for SIP servers and enables SIP applications be deployed and managed based on the servlet model [52]. The conference server is called Medooze multi conference server [46], and is an open source project which allows several participants using a SIP compatible client (either softphones or videophone) join to a conference with audio, video and text mixing between all the participants. It provides a RESTful interface for the invocations. We extract and reuse only the modules which are related to dial-out SIP signalling operations and the other irrelevant modules in this conference server are ignored.

6.2.3.3.6 Medooze Media Mixer

As for the audio mixer substrate, we reused Medooze media mixer [47]. This media mixer is able to handle any media input and provide the desired output for the device that participant is using and participants don't need to share the same codecs. This mixer provides mixing for audio, video and text and transfers media using Real Time Transport Protocol (RTP) over User Datagram Protocol (UDP) and can be run on a Linux OS. We only reuse the modules related to audio mixing. It provides a XML-RPC interface for the invocations that we have built a RESTful interface on top of it to provide our audio mixer substrate with a REST interface.

6.2.3.3.7 Glassfish Application Server

GlassFish Server is the implementation of the Java Platform by Oracle. GlassFish Server delivers a flexible, lightweight, and production-ready Java EE 6 application server [51].

6.2.3.3.8 SailFin Application Server

SailFin is an open-source Java application server by Sun Microsystems. It implements JSR 289 specification of SIP servlet APIs and defines a standard application programming model to mix SIP servlets and Java EE components such as web services and persistence, and enable faster development of smarter communications-enabled applications, while integrating with existing GlassFish services [50].

6.2.3.3.9 MySQL Server

The dial-out audio conference application uses persistence unit to save the information of registered conference end users, conferences and the participants of the conferences. For this purposes a MySQL database (MySQL workbench 5.2) is used. MySQL is a popular open source database which MySQL can cost-effectively help to deliver high performance, scalable database applications [49].

6.2.3.3.10 X-Lite

To provide SIP clients for conference end users, we used X-Lite (version 3 and 4) which is a SIP based softphone. To enable making voice and video calls with X-Lite, first it should be registered with a VoIP service provider [48]. In our prototype, we register them with the dial-out signaling server.

6.2.3.4 Procedures

In this section we show the sequence of 3 main operational procedures in the prototype based on the REST interfaces designed: 1) Service activation procedure (Figure 6-12), 2) *create conference* execution procedure (Figure 6-13), and 3) *add participant* execution procedure Figure 6-14).

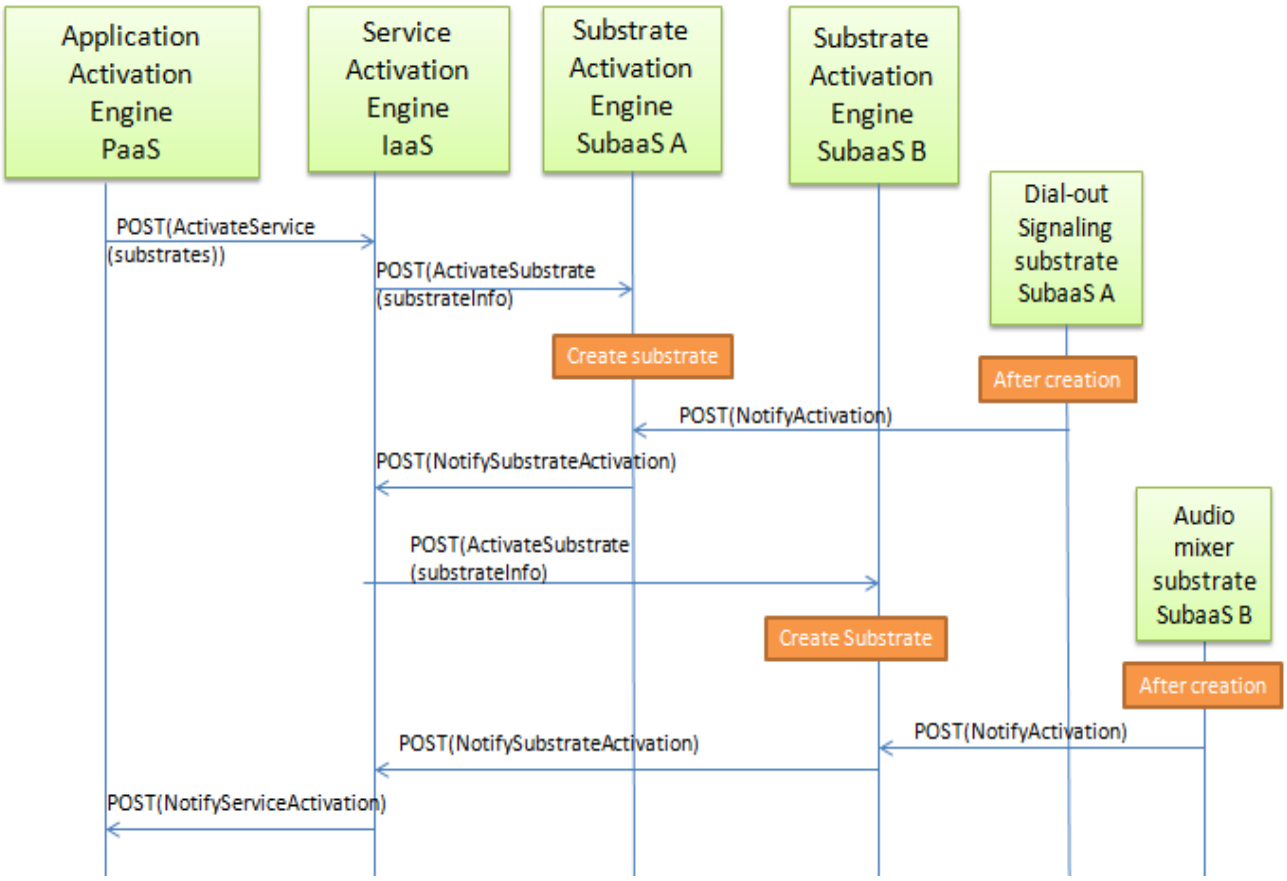

Figure 6-12 . Procedure of Activation of Dial-out Audio Conference Application

z

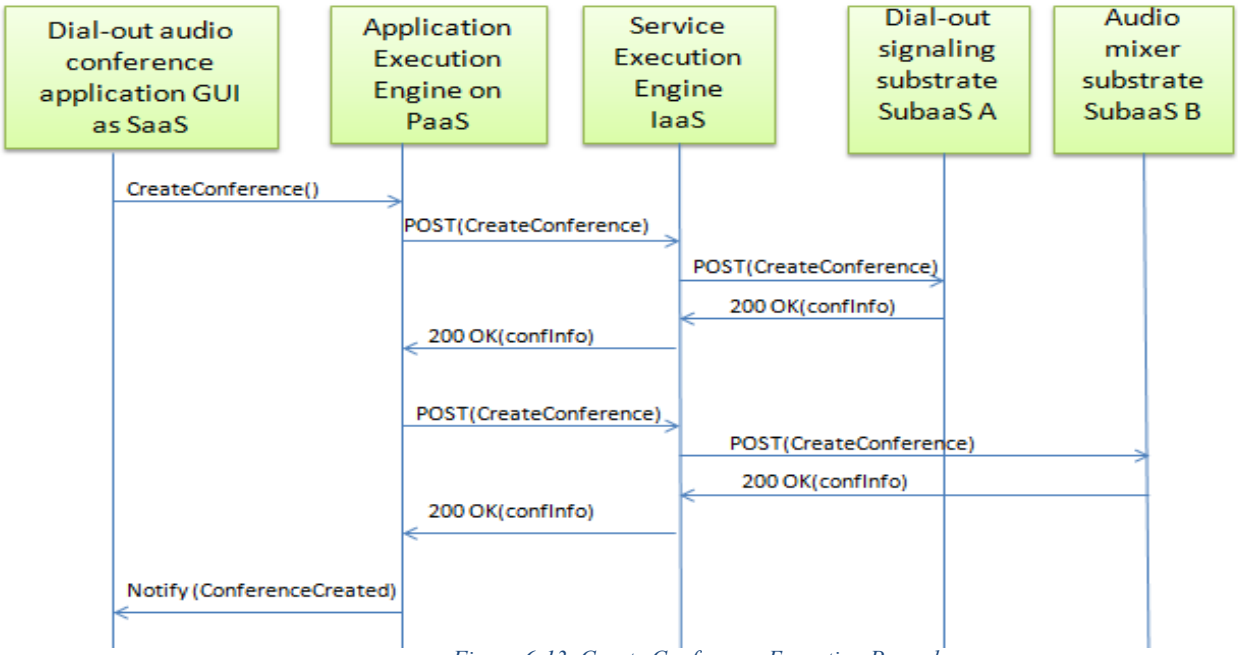

Figure 6-13. Create Conference Execution Procedure

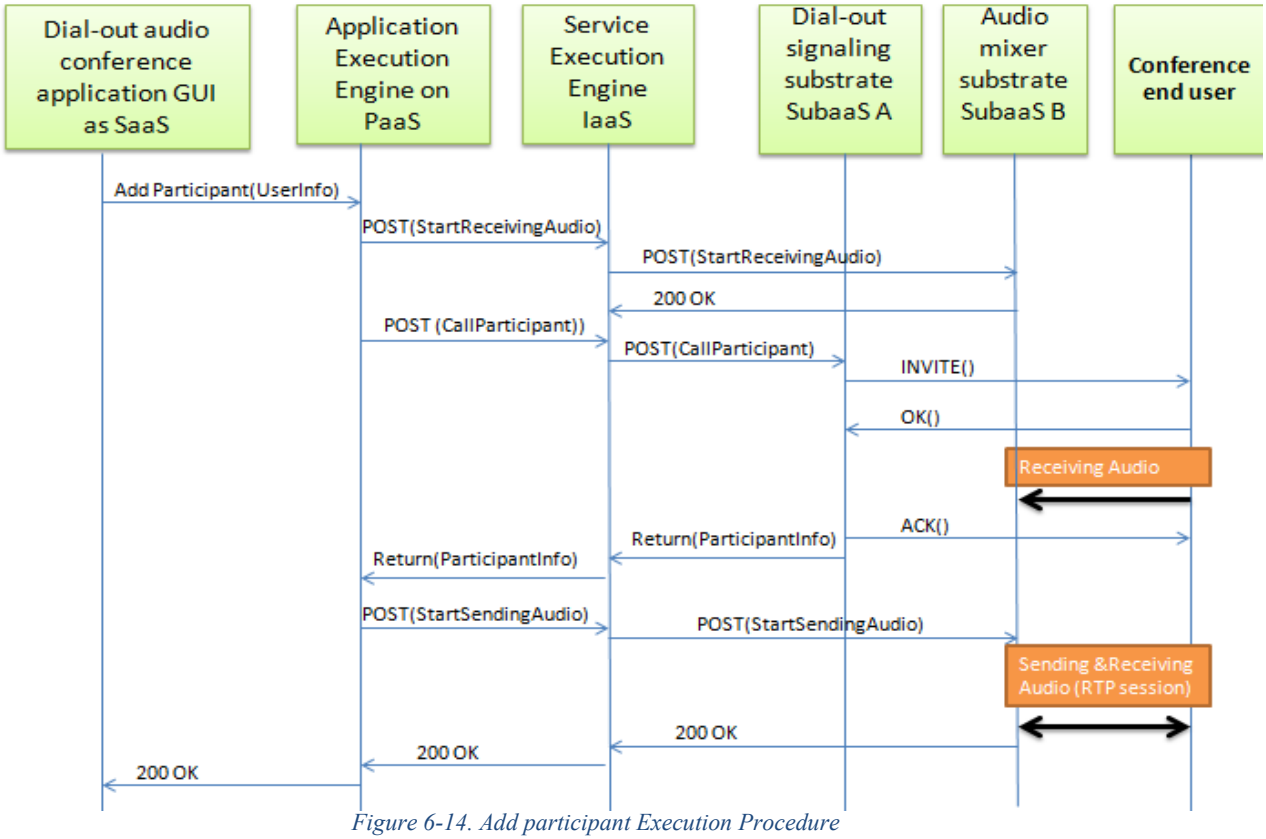

Figure 6-14. Add participant Execution Procedure

6.2.3.5 Environmental Setting

All the components of the prototype are located in a local network. We have used a Xen server from our laboratory to host the prototype components. Totally we have 5 nodes in the prototype. Each node is hosted on a separate VM on the Xen server. First node (named PaaS VM) hosts the PaaS components (the dial-out audio conference application and application activation GUI) and 2 X-lite clients, second one (IaaS VM) hosts IaaS components and 2 X-lite clients, third one (SubaaS VM) hosts substrate activation engine of SubaaS, forth one hosts dial-out signaling substrate and fifth one audio mixer substrate. Specification of these VMs is as follows:

- Dial- out signaling VM: Windows 7 (64 bit), storage: 9.5 GB, memory: 1024 MB, CPU: Intel core 2 (2.53 GHz)
- Audio mixer VM: Ubuntu 12.10, storage: 9.5 GB, memory: 1024 MB, CPU: Intel core 2 (2.53 GHz)
- PaaS VM: Windows 7 (64 bit), storage: 9.5 GB, memory: 1024 MB, CPU: Intel core 2 (2.53 GHz)
- IaaS VM: Windows 7 (64 bit), storage: 42 GB, memory: 1024 MB, CPU: Intel core 2 (2.53 GHz)
- SubaaS VM: Windows 7 (64 bit), storage: 9.5 GB, memory: 1024 MB, CPU: Intel core 2 (2.53 GHz)

6.3 Performance Evaluation

In this section we present the performance measurements from the implemented prototype. Please take note that the measurements are based on a prototype which is built on a local network. In a real cloud conferencing model each of providers may belong to different networks and be spread across different geographical locations.

6.3.1 Performance Metrics

We consider response time as the performance metric to study the time delay of prototype's different functions. We measure the response time of 3 main requests. They are as follows:

- Response time to substrate activation request in the activation phase.
- Response time to *create conference* requests in the execution phase.
- Response time to *add participant* requests in the execution phase.

6.3.1.1 Substrate Activation Measurements

The response time to substrate activation request is the time delay starting from the point a substrate activation request is sent from IaaS to SubaaS, until the point that substrate activation notification is sent back to the requester. We have measured the response time to substrate activation requests for a dial-out signaling substrate and an audio mixer substrate. The results are shown in the table 6-1. This time includes the time for sending activation request, creation of VM, starting it, the boot time of VM's operating system and sending the activation notification back to request. Booting time has the largest share

in this time delay. The windows VM has a longer activation time than the Ubuntu VM mainly because Windows' boot time is longer than Ubuntu's boot time (the other specifications of these two VMs are the same). As the activation operation is a one-time operation which happens once only at the time of service activation, this time delay is acceptable.

Substrate Name	VM Specification	Activation Time (ms)
Audio mixer	Ubuntu 12.10 Storage: 9.5 GB Memory: 1024 MB CPU: Intel core 2 (2.53 GHz)	41533
Dial-out signaling	Windows 7 (64 bit) Storage: 9.5 GB Memory: 1024 MB CPU: Intel core 2 (2.53 GHz)	53032

Table 6-1. Substrate Activation Time Delay

6.3.1.2 Create Conference Request Measurements

The diagram in figure 13 shows the response time to the *create conference* requests. It measures the time delay from the point that conference administrator asks for creating a certain number of conferences until they are created and response is sent back to the requester. This test is repeated for different number of conferences. The conferences are created one by one in a loop repeated for number of conferences. With the increase of number of conferences the response time grows, as server gets busier and uses more resources. The growth of response time is approximately linear with a slope around 3/2 with the number of conferences.

To have a comparison, we also built a dial-out audio conference application using the same substrates with the same specification and same environmental setting, but this time using a non-cloud distributed model in order to determine the overhead which is added to response time in the cloud model. In this configuration we have a dial-out audio conference application, a dial-out signaling server, an audio mixer server which are distributed in a local network (Figure 6-15). In contrast to the cloud model, there is no intermediate layer between conference application and the conferencing elements (in the cloud model there is an IaaS intermediate layer in between). Also despite to the cloud model which it was the IaaS interacting with two substrates for the execution coordination, in the non-cloud distributed model it is the signaling component which directly interacts with audio mixer. The conference application is a REST client to dial-out signaling server and dial-out signaling server is a REST client to audio mixer server.

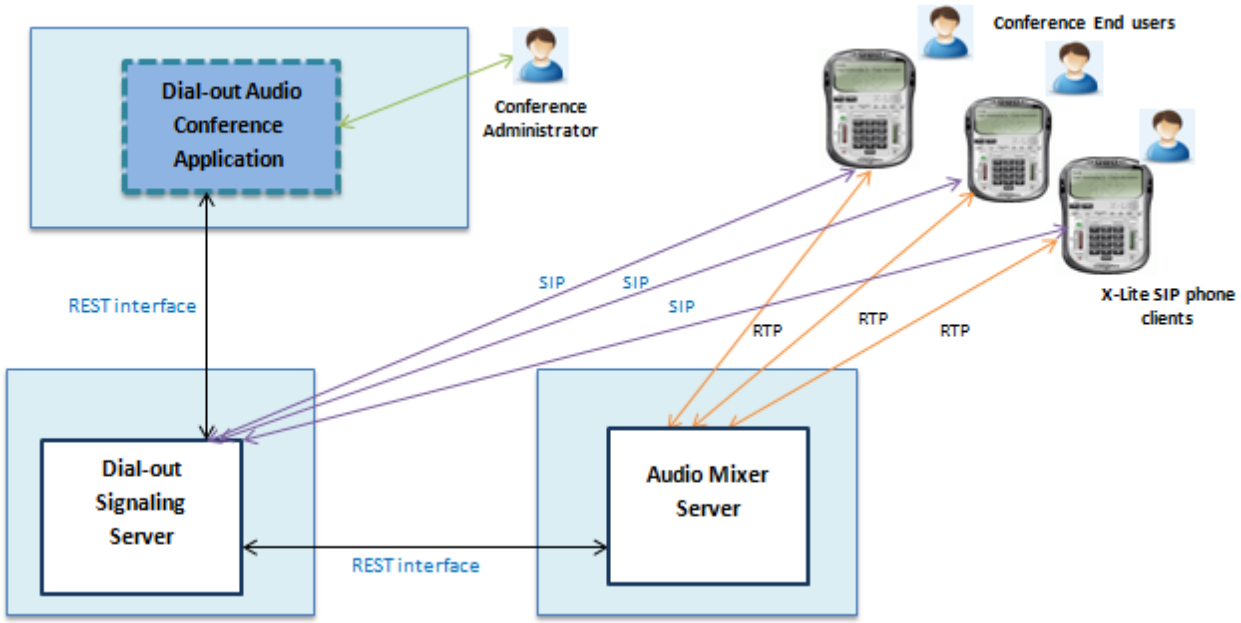

Figure 6-15. Non-cloud Distributed Conferencing Model

The results are collected for the same *create conference* requests in a non-cloud distributed environment (table 6-2). The response times of non-cloud distributed model (red line) are compared with the results collected from the cloud model (blue line) in the Figure 6-16.

Number of Conferences	Create Conference Response Time in cloud model (in ms)	Create Conference Response Time in non-cloud distributed model (in ms)
1	965	649
2	1459	910
4	2312	1431
6	3229	1982
8	3998	2586
10	6331	4102

Table 6-2. Measurement Result of Create Conference Request

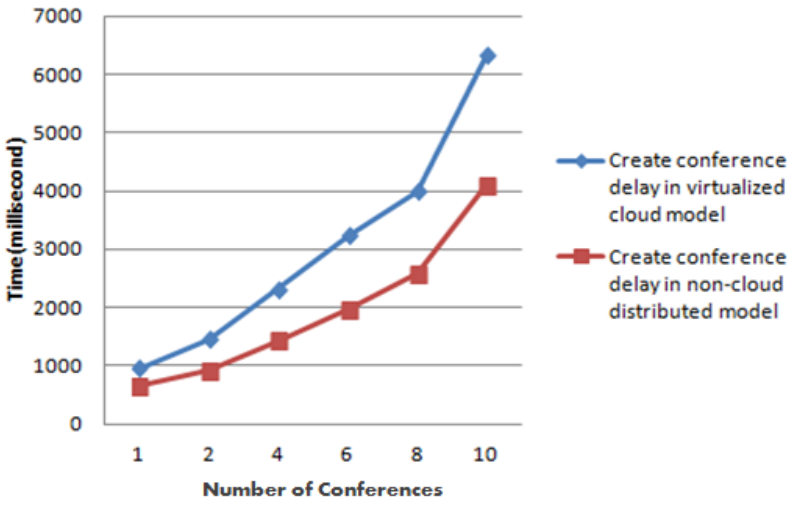

Figure 6-16. Create conference Delay in Cloud and Non-cloud Distributed Models

The result shows that the non-cloud distributed model has less overhead than cloud model by 32 to 38 %. Each conference can have many participants. After creation of a *conference*, participants can join/leave the conference. So, *create conference* will be a onetime operation for a group of participants. The time delays are a few seconds and are considered acceptable. The acceptability of these delays may also depend on the desired response time by service provider for creating a conference.

6.3.1.3 Add Participant Request Measurements

The diagram in figure 6-17 shows the time delay for adding participants to a created conference room. By this time we mean the delay from the point that conference administrator asks for adding a certain number of users to a conference until the point that the session is established for participants. This test is repeated for different number of participants. In order to have a comparison, again we compare the results collected from cloud model with the non-cloud distributed model to find out the overhead added in the

cloud model (Figure 6-17). Auto answering of X-Lite softphones is enabled and there is no human delay involved in answering the calls. With the increase of number of participants the response time grows significantly. To have a more clear view how the response time grows with the number of participants, simulations are required to generate huge number of participants.

For an add participant request several REST requests have to be processed by passing the PaaS and IaaS layers back and forth. After the establishment of the session, the audio is exchanged directly between the end user and audio mixer substrate. So, the actual service which is receiving/transmission of media from/ to end users will not be affected in this cloud model. This time delay which is a few seconds is reasonable. But again the acceptability of this time delays may also depend on the service provider's desired time delay for adding a participant to a conference.

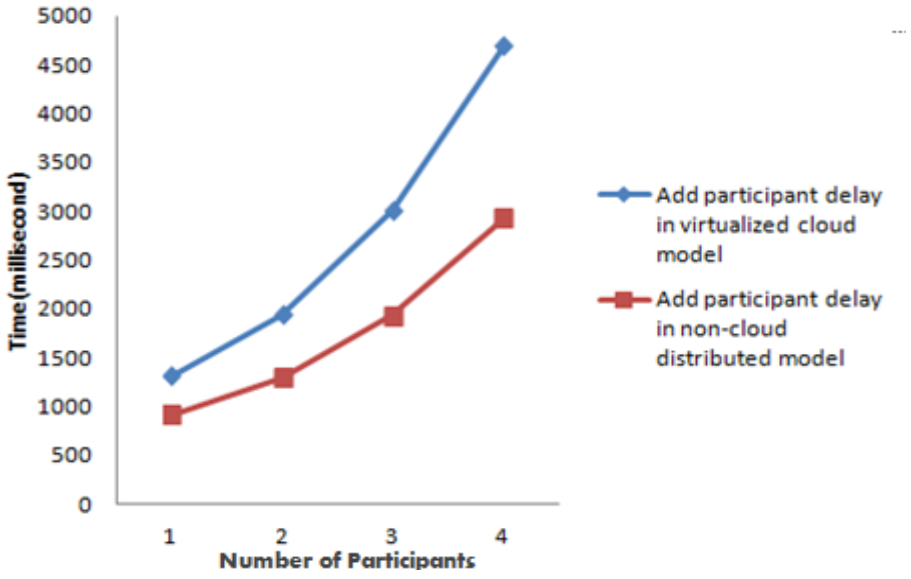

Figure 6-17. Add Participant Delay in Cloud and Non-cloud Distributed Models

The result shows that the response time of add participant requests in the non-cloud distributed model is less than the cloud model by 30 to 37 %.

6.4 Chapter Summary

In this chapter, we presented the overall software architecture and exposed the general architecture proposed in chapter 3. We explained each of architectural components and communication interfaces and then we presented the prototype implemented as a proof of concept. To validate our proposed architecture, we designed a scenario for the a dial-out audio signaling conference service where we assumed that we have a conference service provider offering a dial-out audio conference application, a conferencing platform provider, a conferencing infrastructure provider and two conferencing substrate providers offering dial-out signaling and audio mixer substrates. The conferencing substrates were virtualized and deployed on Xen server. We ran the prototype and showed the feasibility of our architecture. We collected some performance measurements and analyzed them. Through these experiments, we learned that our proposed architecture can be a valid approach for having a virtualized infrastructure for conferencing applications.

In the next chapter, we will summarize our contribution in the thesis and then we propose some additional future research directions.

Chapter 7

7 Conclusion and Future Work

In this chapter we conclude the thesis by summarizing our contributions and then we provide several research directions for the future work.

7.1 Contribution Summary

Conferencing applications are among important enterprise applications nowadays. However, fine grained scalability and elasticity remain quite elusive for multimedia conferencing applications. Cloud-based conferencing services can inherent several benefits such as resource usage efficiency, scalability, elasticity and easy introduction of different types of conferences.

This thesis relied on a recently proposed business model for cloud-based conferencing. The model has the following roles: conferencing substrate provider, conferencing infrastructure provider, conferencing platform provider, conferencing service provider, and broker. Conferencing substrates are generally the elementary building blocks of conferencing applications which can be virtualized and shared among several conferencing applications. This thesis focused on the conferencing infrastructure provider and conferencing substrate provider roles. We specified the requirements for a virtualized cloud infrastructure for multimedia conferencing applications. We have reviewed the state of the art and related work and evaluated them based on our requirements. We found that none of them satisfies all of our requirements.

We proposed an architecture to realize the roles of conferencing infrastructure provider and conferencing substrate provider. We integrated a broker architecture from a previous work into our architecture for publication and discovery of substrates. We also designed RESTful interfaces for the communication of components in the architecture. Furthermore, we proposed a resource allocation mechanism for scaling of conferencing substrates. It is fine grained resource allocation mechanism which performs scaling actions at the granularity of underlying resources.

We proposed a software architecture and implemented a prototype as a proof of concept to validate the feasibility of proposed architecture. The implemented scenario is where we have a dial-out audio conference application offered by a service provider, a conferencing platform provider, a conferencing infrastructure provider and two conferencing substrates of dial-out signaling and audio mixer given by two different substrate providers which the dial-out audio conference relies on. For the prototype we used XEN as virtualization framework. We collected some performance measurements on the prototype for different type of requests and evaluated the collected results. Based on the result, we concluded that the processing time of requests is reasonable.

7.2 Future Work

We peoposed an scaling mechanism to be used for scaling conferencing substrates. This mechanim is still in a priliminary level and it needs to be further investigated and enhanced to take some other factors into considerations. Some of potential enhancemnts are: adding a mechanism for distributing the application workload among VMs hosting the same application on different PMs, using VM migration as an alternative solution to

perform more resource level scaling actions when there is no resource available on the current PM to be allocated for resource level scaling. Eventually this mechanism should be implemented as an algorithm and the algorithm should be tested. Simulations should be done to produce different work loads and evaluate the algorithm based on different quality metrics of conferencing substrates. We studied the scaling at SubaaS level for conferencing substrates. Moreover, the scaling of the components in IaaS and PaaS layers which are transfer bottlencks should be also studied.

For the prototype we implemented a simple scenario where substrates are used by a single application. More complex scenario where a substrates is shared between several applications can be tested and the results should be evaluated. Application isolation should be provided by substrates and a chrging model should be proposed.

This thesis had focus on conferencing IaaS and SubaaS. The other cloud layers of PaaS and SaaS in the cloud based conferencing model should be further investigated in future. In the scenario we assumed the composite service is already created by PaaS. Creating of composite conferencing services from multiple conferencing substrates should be studied. Some tools for service creation should be provided, in which service providers with different levels of technical experience be able to use it and create new conferencing services. Usage of other client interfaces for conference end users rather than softphones (e.g. web browser communications using WebRTC, mobile phones, etc.) and also the usage of different signaling protocols (e.g. Extensible Messaging and Presence Protocol (XMPP)) should be further investigated.

We studied cloud-based conferencing applications in this thesis. Furthermore the usage of cloud conferencing domain in other cloud domains can be studied. E.g. in game applications or distance learning application to include conferencing capabilities by using the services given in cloud conferencing.

Bibliography

[1] R. H. Glitho, 'Cloud-based Multimedia Conferencing: Business Model, Research Agenda, State-of-the-Art', *2011 IEEE 13th Conference on Commerce and Enterprise Computing (CEC)*, 2011, pp. 226 –230.

[2] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, 'A break in the clouds: towards a cloud definition', Acm Sigcomm Comput. Commun. Rev., vol. 39, no. 1, pp. 50–55, 2008.

[3] S. NIST, 800-145: 'The NIST definition of cloud computing', 2012.

[4] 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.

[5] 'XenSource Inc: Xen'. [Online]. Available: http://www.xensource.com/. [Accessed: 8-Oct-2013].

[6] 'VMWare ESX Server'. [Online]. Available: http://www.vmware.com/products/esx/. [Accessed: 8-Oct-2013].

[7] M. Miller, 'Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online', QUE publishing, 2009.

[8] L. M. Vaquero, L. R. 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.

[9] I. Sriram, and A. Khajeh-Hosseini, 'Research Agenda in Cloud Technologies', 1st ACM Symposium on Cloud Computing, SOCC 2010.

[10] N. Antonopoulos, and L. Gillam, 'Cloud Computing: Principles, Systems and Applications', Springer publications, 2010.

[11] R. Buyya, J. Broberg , A. M. and Goscinski, 'Cloud Computing: Principles and Paradigms', John Wiley & Sons publications, 2011.

[12] 'What is cloud computing?'. [Online]. Available: https://developers.google.com/appengine/training/intro/whatiscc [Accessed: 8-Sept-2013].

[13] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong, 'The Characteristics of Cloud Computing', 39th International Conference on Parallel Processing Workshops, 2010.

[14] M. Barnes and C. Boulton, 'A Framework for Centralized Conferencing - RFC 5239', RFC 5239, June, June 2008.

[15] G. Camarillo, K. Drage, and J. Ott, 'Binary Floor Control Protocol (BFCP) - RFC 4582', 2006.

[16] O. Novo, G. Camarillo, and D. Morgan, 'Conference Information Data Model for Centralized Conferencing (XCON) - RFC 6501', RFC 6501, March 2012.

[17] 3GPP TS 24.147, 'Conferencing Using the IP Multimedia (IM) Core Network (CN)', Stage 3, Release 11', September 2012.

[18] H. P. Dommel, and J. J. Garcia-Luna-Aceves, 'Floor control for multimedia conferencing and collaboration', Baskin Center for Computer Engineering & Information Sciences, University of California, Santa Cruz,USA, 1997.

[19] J. Rosenberg, 'A Framework for Conferencing with the Session Initiation Protocol (SIP)', IETF RFC 4353, February 2006 [Online]. Available: http://tools.ietf.org/html/rfc4353. [Accessed: 20- Dec-2013].

[20] 'RFC 4245, 'High-Level Requirements for Tightly Coupled SIP Conferencing', [Online]. Available: http://tools.ietf.org/html/rfc4245 [Accessed: 10- Dec-2013].

[21] T. Zhao , and Y. Wang, 'Virtual High Performance Computing Environments for Science Computing on-Demand', High Performance Computing Center, 2011 Sixth Annual ChinaGrid Conference.

[22] G. J. Popek, and R. P. Goldberg, 'Formal Requirements for Virtualizable Third Generation Architectures', Communications of the ACM, 1974.

[23] 'Virtualization in education'. [Online]. Available: http://www-07.ibm.com/solutions/in/education/download/Virtualization%20in%20Education.pdf, IBM. October 2007. [Accessed: 20- Nov-2013].

[24] 'Xen Server'. [Online]. Available: 'http://www.citrix.com/products/xenserver/overview.html [Accessed: 20- Oct-2013].

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

[26] 'VMware'. [Online]. Available: http://www.vmware.com/pdf/virtualization.pdf [Accessed: 20- Oct-2013].

[27] 'Paravirtualization' [Online]. Available: http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf [Accessed: 20- Oct-2013].

[28] A. Buono, S. Loreto, L. Miniero, and S.P. Romano, 'A distributed IMS enabled conferencing architecture on top of a standard centralized conferencing framework', IEEE Communications Magazine, vol.45, No 3, March 2007.

[29] 3GPP TS 29.199-12, 'Open Service Access (OSA), Parlay-X Web Services' – Part 12: Multimedia Conference (Release 9), December 2009.

[30] 'WebEx'. [Online]. Available: http://www.webex.com/ [Accessed: 5-Dec-2013].

[31] 'Blue Jeans'. [Online]. Available: http://bluejeans.com/video-conferencing [Accessed: 5-Dec-2013].

[32] 'Hypervisor' .[Online]. Available: http://en.wikipedia.org/wiki/Hypervisor [Accessed: 20-Oct-2013].

[33] J. N. Matthews, E. M. Dow, T. Deshane, W. Hu, J. Bongio, P. F. Wilbur,and B. Hohnson, Running Xen; A hands-on guide to the art of virtualization, Prentice Hall, 2008.

[34] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for cloud systems," in CNSM'10, Niagara Falls, Canada, 2010, pp. 9 - 16.

[35] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: elastic resource scaling for multi-tenant cloud systems," in SOCC'11, Cascais, Portugal, 2011.

[36] J. Cervino, F. Escribano, P. Rodriguez, I. Trajkovska, and J. Salvachua, 'Videoconference Capacity Leasing on Hybrid Clouds', 2011 IEEE 4th International Conference on Cloud Computing.

[37] P. Rodriguez, D. Gallego, J. Cerviiio, F. Escribano, J. Quemada, and J. Salvachua, 'VaaS: Videoconferencing as a Service', 5th International Conference on Collaborative Computing: Networking, Application and Worksharing, 2009.

[38] J. George, F. Belqasmi, R. Glitho, and N. Kara, 'A Semantic-Oriented Description Framework and Broker Architecture for Publication and Discovery of Cloud Based Conferencing'. 2013.

[39] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, 'Lightweight Resource Scaling for Cloud Applications', Department of Computing Imperial College London, 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012.

[40] Y. Song, Y. Sun, and W. Shi , 'A Two-Tiered On-Demand Resource Allocation Mechanism for VM-Based Data Centers', IEEE transactions on services computing, VOL. 6, NO. 1, JANUARY-MARCH 2013.

[41] G. Carella, L. Foschini, and T.Magedanz, 'QoS-aware elastic cloud brokering for IMS infrastructures', IEEE Symposium on Computers and Communications (ISCC), 2012.

[42] 'Amazon elastic compute cloud (EC2)'. [Online]. Available: http://aws.amazon.com/ec2/ [Accessed: 2-Jan-2014].

[43] 'GoGrid'. [Online]. Available: http://www.gogrid.com/ [Accessed: 2-Jan-2014].

[44] 'RightScale'. [Online]. Available: http://www.rightscale.com/ [Accessed: 2-Jan-2014].

[45] 'Jersey Web Service Framework'. [Online]. Available: https://jersey.java.net/ [Accessed: 28-Feb-2014].

[46] 'Medooze Multi conference server'. [Online]. Available: http://www.medooze.com/products/mcu.aspx . [Accessed: 28-Feb-2014].

[47] 'Medooze media mixer server'. [Online]. Available: http://www.medooze.com/products/media-mixer-server.aspx [Accessed: 28-Feb-2014].

[48] 'X-Lite'. [Online]. Available: http://www.counterpath.com/x-lite [Accessed: 28-Feb-2014].

[49] 'MySQL' [Online]. Available: http://www.mysql.com/products/ [Accessed: 28-Feb-2014].

[50] 'Sailfin'. [Online]. Available: https://sailfin.java.net/ [Accessed: 28-Feb-2014].

[51] 'Glassfish'. [Online]. Available: http://www.oracle.com/technetwork/middleware/glassfish/overview/index.html?ssSourceSiteId=otncn [Accessed: 28-Feb-2014].

[52] 'JSR 1116'. [Online]. Available: https://jcp.org/en/jsr/detail?id=116 [Accessed: 28-Feb-2014].

[53] 'RFC 3261 SIP: Session Initiation Protocol'. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt [Accessed: 11-Dec-2013].

[54] 'Building Voice Over IP'. [Online]. Available: http://www.networkcomputing.com/netdesign/1109voip2.html?ls=NCJS_1107bt [Accessed: 20-April-2014].

[55] 'SIP Protocol'. [Online]. Available: http://www.ingate.com/files/422/fwmanual-en/xa9835.html [Accessed: 20-April-2014].

[56] 'The SIP advantage'. [Online]. Available: http://www.voip-info.org/wiki/view/The+SIP+advantage [Accessed: 20-April-2014].

[57] N. R. Herbst, S. Kounev, and R. Reussner, 'Elasticity in Cloud Computing: What It Is, and What It Is Not', ICAC the 10th International Conference on Autonomic Computing, 2013.