ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLEMENT OF THE REQUIREMENTS FOR
A MASTER'S IN ENGINEERING
CONCENTRATION INFORMATION TECHNOLOGY
M.ENG

BY
Mbarka SOUALHIA

RESOURCES MANAGEMENT ARCHITECTURE AND ALGORITHMS FOR VIRTUALIZED IVR APPLICATIONS IN CLOUD ENVIRONMENT

MONTREAL, MARCH 11TH 2013

ACKNOWLEDGMENT

Although carrying out this project and writing this thesis was tough at times, but I enjoyed these times immensely and I learned a lot from that experience. Nevertheless, I would not be able to finish it or would not enjoy it without the help and the support of many people. It was a great pleasure to share that experience with them that's why I would like to thank all of them! However, in the hope that I don't forget anyone, a number of people deserve my deepest thanks and I owe my sincere gratitude to them because they were always with me along the way, helped and supported me during my thesis.

First and foremost, I owe my deepest gratitude and thanks to the Tunisian Government and the University Mission of Tunisia in Montreal for their support that allowed me to carry my studies in comfortable conditions.

It is a good opportunity also to express my special thanks of gratitude to my supervisor Dr. Nadjia Kara for all her support, continuous guidance, meticulous suggestions and her inexhaustible patience during the two years. I enjoyed immensely the work with her as she gave me the opportunity to acquire a broad range of experience under her supervision. Without her support, this work would not have been done. Thank you doctor for all your efforts!!!

I am grateful also for Dr. Chamseddine Talhi and Dr Abdelouahed Gherbi for serving as members of my thesis committee and for their valuable comments and feedbacks on my thesis.

A part of my research work was done in the Telecommunication Service Engineering (TSE) research lab at Concordia University where I discussed my presentations face-to-face with Dr. Nadjia, Dr. Roch and Dr. Fatna. So, I would like to thank them for their detailed feedbacks and insightful comments on my project because their help was crucial to enhance and improve it.

I am also very thankful to Mr. Patrice Dion our technical assistant in the department who introduced me to various techniques and helped me to solve problems concerning my practical work.

Furthermore, I dedicate this thesis to my deceased mother who supported me before and gave me the ambition to go on my studies. Also, I would express a deep sense of gratitude to my family, especially to my dad, sisters, brothers and nephews because they have always stood by me like a pillar in times of need. They helped and supported me all over the time.

Finally, I owe a great thank to my friends here in Canada and Tunisia for their encouragement, moral support and blessing. My special thanks are due to Taoufik Bdiri, Ons Seddiki and Christian Azar who strengthened my moral by standing by me in all situations.

ARCHITECTURE ET ALGORITHMES DE GESTION DE RESOURCES POUR DES APPLICATIONS IVR VIRTUELLES DANS UN ENVIRONNEMENT INFONUAGIQUE

Mbarka SOUALHIA

RESUMÉ

Les systèmes vocaux interactifs (en anglais Interactive Voice Response : IVR) sont omniprésents de nos jours. IVR est une technologie de téléphonie qui permet aux utilisateurs d'accéder aux services offerts par un système d'information automatisé d'une entreprise via un clavier de téléphone ou des commandes vocales. L'infonuagique (en anglais Cloud Computing) est un nouveau paradigme qui par l'entremise de serveurs permet d'héberger et de fournir des services sur l'Internet avec de nombreux avantages inhérents. Il offre trois modèles de services majeurs: Infrastructure en tant que service (en anglais Infrastructure as a Service : IaaS), Plateforme en tant que service (en anglais Platform as a Service : PaaS) et Software en tant que service (en anglais Software as a Service : SaaS). L'infornuagique est basée sur la technologie de virtualisation qui permet la coexistence de différentes entités sur le même matériel physique. Ces entités peuvent être des systèmes partageant le même équipement matériel, des applications hébergées par le même système d'exploitation ou bien des réseaux complets partageant les mêmes routeurs. Le principal objectif visé par cette technologie est le partage efficace de ressources physiques.

Plusieurs applications multimédias sont offertes aujourd'hui dans les environnements infonuagiques. Cependant, au meilleur de notre connaissance, il n'existe pas d'architecture qui crée et gère les applications IVR dans de tels environnements. Par conséquent, ce mémoire propose une nouvelle architecture permettant de créer à la volée et de gérer d'une manière efficace des applications IVR dans un environnement infonuagique. Il décrit une nouvelle architecture d'une infrastructure de virtualisation d'applications IVR et démontre son potentiel à travers un cas d'utilisation. Il propose deux stratégies de gestion de ressources et d'ordonnancement de tâches comme un part essentiel au partage des ressources dans une telle architecture.

RESOURCES MANAGEMENT ARCHITECTURE AND ALGORITHMS FOR VIRTUALIZED IVR APPLICATION IN CLOUD ENVIRONMENTS

Mbarka SOUALHIA

ABSTRACT

Interactive Voice Response (IVR) applications are ubiquitous nowadays. IVR is a telephony technology that allows interactions with a wide range of automated information systems via a telephone keypad or voice commands. Cloud computing is a newly emerging paradigm that hosts and provides services over the Internet with many inherent benefits. It has three major service models: Infrastructure as a service (IaaS), Platform as a service (PaaS), and Software as a Service (SaaS). Cloud computing is based on the virtualization technology that enables the co-existence of entities in general on the same substrates. These entities may be operating systems co-existing on the same hardware, applications co-existing on the same operating system, or even full-blown networks co-existing on the same routers. The key benefit is efficiency through the sharing of physical resources.

Several multimedia applications are provided in cloud environments nowadays. However, to the best of our knowledge, there is no architecture that creates and manages IVR applications in cloud environment. Therefore, we propose to develop a new virtualized architecture that can create, deploy and manage IVR applications in cloud environment. We also propose two new algorithms for resources management and task scheduling as an essential part of resource sharing in such environment.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABREVIATIONS

API	Application Programming Interfaces
IAAS	infrastructure as a Service
PAAS	Platform as a Service
SAAS	Software as a Service
IVR	Interactive Voice Response
TTS	Text To Speech
VTT	Voice To Text
QoS	Quality of Service
VM	Virtual Machine
VMM	Virtual Machine Manager
REST	REpresentation State Transfer
HTTP	HyperText transfer Protocol
XML	eXtensible Markup Language
URI	Uniform Resources Identifier
WADL	WebApplication Description Language
JSON	JavaScript Object Notation
PVS	Presence Virtualization Server
IPTV	Internet protocol TV
VoD	Video on Demand
AuC	Authorization Control
AL	Allocation manager

OM	Optimization Manager
GUI	Graphical User Interfaces
SIP	Session Initial protocol
MSCML	Media Server Control Markup Language
SII	Substrate IVR Instance
GA	Genetic Algorithm
SLA	Service Level Agreement
SubP	Substrate Provider
InfP	Infrastructure Provider
PP	Platform Provider
ServP	Service Provider
DSM	Donkey State Machine
SEMS	SIP Express Media Server
IDE	Integrated Development Environment
QTP	Quick Test Professional.

INTRODUCTION

Multimedia applications such as video streaming, conferencing, multiparty games play a important role in business and everyday life. Developing, managing and accessing such applications using different devices (mobile and fixed devices) is challenging tasks that cloud computing and virtualization can help in solving. However, very few multimedia applications are provided in cloud environment.

Cloud computing is a newly emerging paradigm that hosts and provides services over the Internet with many inherent benefits. It offers many advantages such as the easy introduction of services, scalability and resources efficiency. However, it has not yet a standard and agreed definition. It is based on sharing data, services and resources over the network and end users can access cloud applications through web browsers, mobile applications, API (Application Programming Interface), etc. Nevertheless, a consensus is emerging around the most critical facets it encompasses: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Zehua Zhang and Xuejie Zhang , Nov. 2009). Platforms providers offer platforms as service to the service providers that develop and manage applications. These applications are offered later to the end-users as SaaS based on the pay as you use paradigm. Platforms can be considered as an abstraction of the infrastructure level that is offered as IaaS by infrastructure providers. These infrastructures represent the dynamic pool of virtualized resources to deploy applications and satisfy users' requests.

Cloud computing is based on the virtualization technology that attracted a great deal of attention from researchers. Virtualization technology provides platforms upon which novel network architectures can be built, experimented and evaluated by deploying several applications. Furthermore, network virtualization opens up new possibilities for the evolution of the future Internet by enabling the deployment of different architectures and protocols over shared physical infrastructures. Indeed, it goes a step further by enabling researchers to use multiple approaches of implementation, management and test of new technologies. It enables the co-existence of entities (ex., virtual network and services) in general on the same substrate so that users can share physical resources provided by IaaS. Virtualization improves the sharing and utilizing of the underlying resources in virtualized systems and enables a more efficient utilization of existing computing resources.

Several applications are offered today in cloud settings (e.g. enterprise database, IT help desks). However, despite all the benefits of virtualization and cloud computing, there is no architecture based on the virtualization technology that creates Interactive Voice Response (IVR) applications in cloud environment. IVR is a telephony technology that allows interactions with a wide range of automated information systems via a telephone keypad or voice commands. Its key function is to provide self-service voice information to assist users (S. Xu, et al., 2010). IVR applications rely on substrates such as key detector, voice recorder and dialog manager. They allow incorporating advanced features such as Text to Speech (TTS) and language translators to enhance service quality and customers' satisfaction. However, there is no full-fledged cloud environment that enables the development, management and offering of the full range of IVR applications. Therefore, the overall objective of this project is to carry out research activities to virtualize IVR substrates to enable and to ease the development and the management of IVR applications in cloud computing environments, and to guarantee more efficient resource usage.

Nowadays, many industries such as telecommunications, banking and utilities rely heavily on IVR applications to guarantee a superior quality of experience to customer and to improve their satisfaction. Automated attendant and automated survey are two examples of IVR applications. The automated attendant application allows transferring callers to appropriate callees or extensions while the automated survey application invites respondents to answer to a questionnaire that is played over the phone by pressing the appropriate keys on the phone keypad or by voice. Several benefits are associated to IVR applications such as extending of business hours, reduction of callers' waiting time and service offered to multiple users simultaneously. However, to the best of our knowledge, there is no virtualized architecture that can create, deploy and manage IVR applications in cloud environment despite the entailed benefits. Therefore, we suggest developing a new virtualized architecture to deploy these applications. Furthermore, we can exploit the benefits of cloud computing to develop and manage IVR platforms on the cloud. As a result, IVR service providers can offer IVR applications as services to users. The IVR platform providers add one or more levels of abstraction to the IVR infrastructures provided as IaaS by infrastructure providers. They ease IVR application development and management. They allow the abstraction and sharing of different IVR substrate components such as key detectors, voice recorders and dialog managers, and computer and networks resources. In addition, we define and validate resources management strategy that provides efficient physical resource (computer and networks resources) sharing among different IVR applications. Resource management is part and parcel of IVR virtualization and poses a challenge in virtualized environments where both processing and network constraints must be considered. Considering several objectives to optimize the resource usage makes it even more challenging. In this project, we propose IVR virtualization task scheduling and computational resource sharing (among different IVR applications) strategies based on genetic algorithms, in which different objectives are optimized. The algorithms used by both strategies are simulated and the performance measured and analyzed.

Offering IVR applications in virtualized infrastructure is a new topic in cloud environment. Meanwhile, this project represents a new contribution in this research domain as it may solve many issues such as the creation, deployment and management of the virtualized IVR platforms in cloud environments. We propose in this project a new architecture based on the concept of web services for a virtualized IVR infrastructure in order to deploy IVR applications as services in cloud settings. Furthermore, we define new task scheduling and computational resource sharing strategies based on genetic algorithms for virtualized IVR application. The task scheduling strategy guarantees maximum utilization of resources while minimizing the execution time of tasks for virtualized IVR applications. The computational resource sharing strategy minimizes substrate resource utilization and the resource allocation time while maximizing the satisfactory factor of IVR applications.

The performance measurements conducted showed that the proposed algorithms are promising. Indeed, compared to two commonly used algorithms, the proposed instantiation request task scheduling outperformed in terms of total completion time of processors and average resources utilization. The computational resource sharing algorithm allows efficient resource usage. The main Contributions of this project are:

- Novel architecture for the virtualized IVR infrastructure and IVR platform with specification languages for IVR substrates, algorithms and protocols for dynamic and efficient instantiation and management of the substrates.

- New management plan that enables the actual control and monitoring of substrates resources. It enables and eases the instantiation and resource management of virtualized IVR applications.

- New task scheduling and computational resource sharing strategies that guarantee efficient resource sharing and usage.

- A working prototype that demonstrate the capabilities and performance of the virtualized IVR infrastructure. It represents a first step towards a full-fledged cloud prototype for multimedia applications.

The proposed virtualized IVR architecture has been published in refereed conference and accepted for publication in refereed journal. The resource management strategies are under review by another refereed journal:

- Fatna Belqasmi, Christian Azar, Mbarka Soualhia, Nadjia Kara and Roch Glitho, “A Virtualized Infrastructure for Interactive Voice Response Applications in the Cloud,” ITU-T Kaleidoscope 2011 the fully networked human - Innovations for future networks and services, December 2011.

- Fatna Belqasmi, Christian Azar, Mbarka Soualhia, Nadjia Kara and Roch Glitho, “A case study of a virtualized Infrastructure and its accompanying platform for IVR applications in Clouds,” IEEE Communication Magazine, September 2012 (accepted with revision).

- Fatna Belqasmi, Nadjia Kara, Roch Glitho, Mbarka Soualhia and Christian Azar,“Genetic-based algorithms for resource management for virtualized IVR application,” IEEE Transactions on Network and Service Management, June 2012 (submitted).

The rest of this thesis is organized as follow. Chapter 1 is devoted to related work. It presents the main concepts and definitions related to IVR, cloud computing, virtualization and RESTFUL web services. It review the state of the art related to development of multimedia applications in cloud settings and new task scheduling and computational resource sharing strategies in virtualized and non-virtualized environments. Chapter 2 describes the virtualized IVR architecture proposed in this project. It presents the business model and identifies the substrates that can be shared by IVR applications to enable flexible and efficient resource usage and easy introduction of new functionality. Chapter 3 presents the task scheduling and computational resource sharing strategies that control and manage the use and allocation of computational resources and IVR substrates. The implementation of the proposed prototype as proof of concept to create, deploy and manage virtualized IVR applications is discussed in chapter 4. The performances analysis and results are described in chapter 5. We conclude in the last chapter, with a summary and discussion of the lessons learned as well as our recommendations and suggestions for futures works.

CHAPTER 1

STATE OF THE ART REVIEW

1.1 Introduction

This chapter describes the main background information on four different subjects discussed in this thesis: Cloud Computing, virtualization, Interactive Voice Response (IVR) and resource management in cloud environment. It also describes RESTFul Web services as it is a key technology of our proposed architecture for virtualized IVR applications.

1.2 Thesis's Background and Definitions

1.2.1 Cloud Computing

Cloud computing is an emerging multi-facet paradigm with many inherent advantages, such as the easy introduction of new applications, resource usage and management efficiency and scalability (Shaikh F.B., Haider S, Dec. 2011). A general overview of this concept, its architecture, its service models and the type of clouds are presented in this section.

1.2.1.1 General Overview

Cloud Computing has become one of the most advanced IT paradigm in the new industries as it has many benefits for both cloud providers and users (Bo Peng Hammad et al., Dec. 2011). Cloud Computing has not yet a standard technical definition; however consensus is emerging around the most important facets it encompasses: Infrastructure as a service (IaaS), platform as service (Paas) and software as service (SaaS) (L. M. Vaquero et al., Jan. 2009) (Abdullah R. Eri, Nov. 2011) (Zhengxiong Hou et al., Sept. 2010). Service providers have access to platforms as PaaS by platform providers to develop and manage applications. End-users (or other applications) have access to these applications offered as SaaS on a pay-per-use basis.

Platforms offer one or more levels of abstraction to the infrastructures provided as IaaS by infrastructure providers to ease application development and management. Infrastructures represent a dynamic pool of virtualized resources used by applications. (L. M. Vaquero et al., Jan. 2009) (Bo Peng Hammad et al., Dec. 1 2011) (Fu Wen Li Xiang, Dec. 2011).

This advanced concept is based on the technology of virtualization that can support dynamically the increasing demand for resources in order to satisfy the arrived requests. Actually, these resources are reconfigured transparently to adjust the incoming resources' utilization demands so that they can improve their utilities. Cloud computing is an emerging paradigm that host and provide services over the Internet with many inherent advantages like the ease introduction of new applications and resources, the efficiency and the scalability (Qiang Li et al, Dec. 2009).

1.2.1.2 Cloud Computing Architecture

Cloud computing relies on a business model which is composed of 3 layers: IaaS providers, PaaS providers, SaaS providers and end users. IaaS layer includes computing hardware resources, PaaS the platforms and SaaS the application as shown in Figure 1.1. Amazon EC2 is a well-known as an IaaS provider in the cloud environment (Zhengxiong Hou et al, Sept. 2010). Software development tools, platform, APIs and operating systems are provided by the PaaS providers. Google App Engine is one of the most popular platform providers that facilitate the deployment of applications on the cloud (Prodan R. et al, April 2012). SaaS providers have to deliver the deployed services to the end users through the Internet.

- **Infrastructure as a Service:** IaaS is composed of two layers: the hardware layer and the infrastructure layer. The first layer represents the entire hardware resources provided by infrastructure providers such as CPU, memory, disk space, servers, routers and switches (Voith T. et al, Sept. 2010). The infrastructure layer is responsible for creating virtualized computing and storage instances by partitioning the hardware resources using virtualization technologies such as XEN and VMware

(Citrix, 2011) (VMware ESX Server, 2011). It is also responsible for dynamic resource allocation.

- **Platform as a Service:** PaaS is a mediation layer between the IaaS and the SaaS. It provides operating systems, application framework and services that enable developers to build, test and evaluate their applications on the fly (Fatna Belqasmi et al., Dec. 2011). So developers can deploy transparently their applications in the cloud.

- **Software as a Service:** SaaS provides applications to the end users through standardized interfaces. It enables services providers to access to platform, infrastructure and physical hardware resources offered as on-demand services (Zhengxiong Hou et al., Sept. 2010).

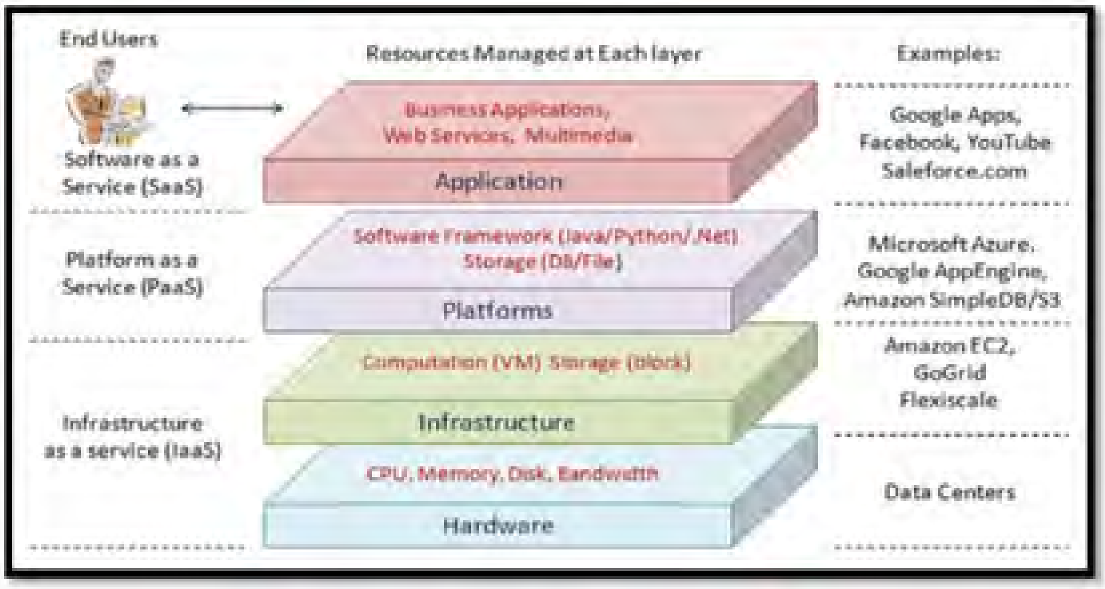

End Users
Resources Managed at Each layer
Examples:
Software as a Service (SaaS)
Business Applications, Web Services, Multimedia
Application
Google Apps, Facebook, YouTube Saleforce.com
Platform as a Service (PaaS)
Software Framework (Java/Python/.Net) Storage (DB/File)
Platforms
Microsoft Azure, Google AppEngine, Amazon SimpleDB/S3
Infrastructure as a service (IaaS)
Computation (VM) Storage (block)
Infrastructure
Amazon EC2, GoGrid Flexiscale
CPU, Memory, Disk, Bandwidth
Hardware
Data Centers

Figure 1.1 Cloud Computing Architecture (Zhengxiong Hou et al, Sept. 2010)

Each layer of Cloud computing architecture could provide one or more services to the upper layer. IaaS offers on-demand infrastructure resources such as virtual machines and virtual routers to the platform providers. Examples of well-known IaaS providers are Amazon EC2

(Amazon Elastic Computing Cloud, 2011) and GoGrid Cloud Hosting (Cloud Computing and Hybrid Infrastructure from GoGrid, 2011). PaaS provides platform resources such as software development and operating system to service providers. Google App Engine (Google App Engine, 2011) and Salesforce (Salesforce CRM, 2011) are two examples of platform providers than give access to a set of APIs to developers to create and test applications in cloud environment. SaaS provides applications to end-users.

1.2.1.3 Types of Cloud

There are three types of clouds: Public, private, hybrid and community. The main characteristics of each type are described in this section.

- **Public Clouds:** These clouds offer their resources on a pay-per-use basis or for free to general public through the Internet. The key issues with this type of cloud are lack of control for stored data and lack of security in provided resources (ex. Operating systems) (Barcomb, K.E. et al., Nov. 2011).

- **Private Clouds:** These clouds provide proprietary computing architecture to organizations that want to offer to users more control over stored data, and secure and reliable access to resources and applications (Lamb, J., Nov. 2011). Because, this type of cloud is implemented and controlled by organizations, it gives responsibility to IT department to run and manage it rather than passing such control to another cloud provider.

- **Hybrid Clouds:** These clouds share resources between public and private clouds through a partnership established between providers of both types of clouds. They run and control resources internally and have others deployed and managed by third-party cloud provider. Hybrid clouds are more flexible than public and private clouds, because they could take advantage of cost provided by public cloud part while protecting and managing sensitive data and applications within private cloud part (Rock, M. and Goscinski, A., July 2012).

- **Community Clouds:** these clouds are shared by multiple organizations belonging to community that have specific concerns such as security and reliability of resources and applications whether they are deployed and managed in-house or by third-party. They can be seen as clouds that are tailored to specific needs of different industrial and government sectors such as finance and healthcare (Briscoe G. and Marinos, A., June 2009).

1.2.1.4 Cloud Characteristics and Issues

Cloud computing has many characteristics that distinguished it from traditional computing environment:

- **Resources sharing:** Clouds are based on sharing pool of virtualized resources over the network so that infrastructure provider can allocate them dynamically and share them among different consumers (Moses, J. et al, May 2011) (Cheng-Jen Tang and Miau-Ru Dai , Dec. 2011). This feature provides high level of resource consolidation to cloud computing infrastructure providers where resource are efficiently used with reduced cost in terms of power consumption and cooling. It allows consumers to benefit from this reduced cost while giving him the resource needs.

- **Elasticity:** Cloud computing provides dynamic and efficient resource allocation to consumers. This dynamic resource provisioning requires the ability to augment or reduce resources in real-time according to consumer’s needs. This elasticity scaling is transparent to applications that are deployed in cloud environment. In other words, adding or reducing resources does not affect, neither application concerned by this resource scaling, nor other applications that share this computing environment (Li Wenrui et al., June 2012).

- **Availability:** Making applications available all the times is a real issue for most organizations. It requires redundancy with specific equipment and specialized staffs to define and maintain failover planning that reduce the impact of resource unavailability. Such recovery process could be very complex. Cloud computing could take that responsibility and hide such process from consumers (Singh D. et al., May 2012) (Pandey S. and Nepal S., June 2012).

1.2.2 Virtualization

Cloud computing is based on the technology of virtualization. A general overview, types and some characteristics of virtualization are given in this part of this thesis to describe the utility and the benefits of the virtualization technology.

1.2.2.1 General Definition

Virtualization can be seen as an emerging trend in IT technologies as it enables the co-existence of multiple entities in general on the same substrates (Khan A. et al., Jan. 2012) (Wei Chen et al., Nov. 2008). These entities may be operating systems co-existing on the same hardware, applications co-existing on the same operating system, or even fill-blown networks co-existing on the same routers. It's the basis of clouds as it insures the dynamic assignment and the on-demand resources sharing. It provides multiple approaches to partition, to deliver and to manage the physical resources in cloud settings using virtualization technologies (Iang Li et al., Dec. 2009).

In addition, it reduces the costs of resources uses as it's based on sharing resources across the network and multiple VMs can use the same hardware without affecting the performance of the application running on these VMs. As a result, virtualization can be considered as a powerful way to deploy multiple applications in different domains on the same platform with high resource usage efficiency and low costs (Junghan Kim et al, 2009).

Whether applied to networks, servers or applications, Virtualization has attracted a great deal of attention from researchers. It provides an abstraction layer between users and the physical infrastructure to ease development and management of networks and applications by service and network providers (Uhlig R. et al., May 2005).

1.2.2.2 Virtualization Types

There are different types of virtualization. This section describes some of these types:

- **Hardware Virtualization:** It allows embedding the virtual machine manager called hypervisor in a hardware component (e.g., server or desktop) instead of running on a software application. The hypervisor allows controlling physical resources such as processor and memory. It enables the creation of virtual machines with different operating system running on the same physical equipment without requiring any changes in its source code. The resources such as processor and memory seem to be allocated all to a virtual machine while the hypervisor is allocating resources needed to each running machine. This type of virtualization facilitates the consolidation process of resources and workloads on single physical equipment without requiring specific software application (Skejic Emir, May 2010) (Xianxian Li, May 2011).

- **Software virtualization:** It allows creating and running virtual machines on a single hardware component, most often by emulating a complete computing system of the machine. For instance, Linux based virtual machine running on top of Microsoft Windows operating system of hardware equipment (Kyong-I Ku et al., Feb. 2010).

- **Server virtualization:** It allows masking resources of a server such as processors and operating systems from users. Physical resources of server can be divided into multiple isolated virtual servers or machine using software application. The main virtualization approaches are: full virtualization, para-virtualization and operating system level (OS-level) virtualization. For the three approaches, the physical servers and the virtual servers are called hosts and guests respectively. The difference between them is the way the physical server resources are allocated to virtual server needs. The full virtualization and para-virtualization use hypervisor as platform to support guests, but unlike the full virtualization, the virtual servers in the para-virtualization are kept aware of each other. The OS-level virtualization approach uses the operating system of host to provide a fully virtualized hypervisor. It doesn't support a separate software entity acting as a hypervisor. In this case, all guests run the same operating system, but remain independent from each other (Xianmin Wei, Sept. 2011) (Prangchumpol D., Nov. 2009).

- **Network virtualization:** It allows isolating and dissociating virtual networks from underlying physical network infrastructure. It enables the sharing of available network resources such nodes (e.g., CPU, and memory) and links (e.g., bandwidth) by splitting them into virtual nodes and virtual links. These nodes and links are used to deploy virtual networks on top of physical network infrastructure (Khan A. et al. , Jan. 2012).

- **Storage Virtualization:** It allows pooling physical devices from multiple storage service providers and making them appearing to users as single storage equipment. It allows fast and efficient management of data storage. It eases the backup, archiving and recovery of data (Qiao L. Iyer et al., April 2005) (Jun-wei Ge et al.,Oct. 2010).

1.2.2.3 Virtualization Characteristics

In general, virtualization is based on a Virtual Machine Manager (VMM) (Yudong Guo et al., May 2010) that manages the relationship between the physical resources and the virtual ones. This manager enables the allocation, assignment, management and the control of sharing resources procedure between the virtual machines. It controls multiple virtual machines on the same hardware and runs them on the same environment to improve physical server utilization and application performances simultaneously. In fact, the VMM or the hypervisor can be considered as an abstraction layer of the hardware equipment; it enables isolation in terms of partition between the physical and virtual components (Euiyoul Ryu et al., July 2010).

Virtualization has many benefits as it can reduce the hardware costs management: the physical resources are assembled on a same server so we have less number of servers. In addition, it enables separation between applications as they are running separately in different VMs. It not only supports consolidation strategy to improve resources utilization but it also allows deploying multiple operating systems on the same server. In fact, virtualization is a promising technology which facilitates the development of services published to users in future Internet as it eases the introduction of new advanced concepts. In the next subsection, we describe some examples of virtualization projects that show the obvious potential benefits of virtualization.

1.2.3 Virtualization Projects in Cloud Settings: Examples

1.2.3.1 Presence Service Virtualization

Actually current solutions offering this service are using a huge amount of resources to create the associated interfaces and applications and to manage them. As the number of service providers increases with the growth of Internet applications, so they need more resources (database, bandwidth, CPU, etc.) to deploy the functionalities and to control the use of these resources in order to guarantee a high level of performance. For that purpose, it's necessary to develop an architecture that can deploy this service using the optimal resources in network infrastructure and resolve the trade-off between QoS and the required resources.

Presence server is used by a wide range of Internet applications such as presence enabled conferencing and instant messaging. Its deployment is expected to grow rapidly due to increasing demand for Internet application such as social networks. Current standard or non-standard implementations of presence application don't allow the re-use of a presence substrate. Deploying new presence services requires implementing them from scratch. Therefore in (Fatna Belqasmi et al., 2011) (Arub Acharya et al. 2009), authors define a virtualized presence services using virtualization technologies to manage presence service in new generation networks. They propose platforms to supervise and control the interaction between the presence service substrates and to manage the quality of the provided services.

In (Fatna Belqasmi et al., 2011), authors described a business model and an overall architecture that enables re-use of the substrate and the rapid development of virtualized presence services. Two interacting planes are defined in this architecture: presence service and virtualization control and management. The presence service plane allows provisioning of virtualized presence services while the control and management plane is responsible of the virtualization control and management tasks. This architecture also defines three layers. Layer 1, layer 2 and layer 3 provide the presence substrate infrastructure, the virtualization infrastructure and virtual presence services respectively. Concrete examples of deployment, control and management of SIP SIMPLE and XMPP virtualized presence services are given in (Fatna Belqasmi et al., 2011).

In (Arub Acharya et al. 2009), presence service virtualization consists in decomposing presence servers in different entities where the principal one is the Presence Virtualization Server (PVS) which receives clients' requests and then transforms them into XML format in order to process them in XML processing engine. In fact, the proposed solution is based on building a new virtualized architecture based on the XML processing engine and manipulating the XML streams which contains the presence status schemas. Virtualization here allows clients to show and share their presence transformations with other potential clients and to support a wide variety of virtualized queries dynamically. It enables also the control of the presence updates and notifications by facilitating the process of schemas streaming from the presence server. In addition, these streams facilitate the management of presence queries and it can reduce waiting time for users as described in the results of experiences realized in (Fatna Belqasmi et al., 2011). The performance results observed by testing the PVS manipulations show that applications, via virtualization, are no longer obliged to fetch data from different servers because presence information are collected into the same presence server (PVS). Furthermore, the proposed platform enables the storage of common functions for example for subscription loads or update notifications in order to re-use them easily and to address the scalability issue in this platform.

1.2.3.2 Video Conferencing Service Virtualization

In (Junchao Li et al., 2010), authors suppose framework to provide video conferencing as a service in cloud computing environments. Therefore, cloud computing seems to be a good alternative as video conferencing services are provisioned to the conference organizer on a pay-per-use basis. Authors divided the cloud conferencing system into 4 layers: physical, virtualization, platform and application. Physical layer is the hardware layer which is composed of physical computing, storage and networking resources. Virtualization layer is composed of virtualized computing, storage and network resources. Platform layer has two sub layers: computing framework and application capability layer. Computing framework layer manages the task scheduling while the application capability layer supports required functions to deploy the applications. Application layer is top layer of this framework, which delivers the conferencing applications as a service over the Internet to end-users.

1.2.3.3 Internet Protocol- TV (IPTV)

The increasing demand for multimedia data delivery requires sufficient and sophisticated transport solutions for multimedia streams. In (Phooi Yee Lau et al., 2010), researchers propose architectural framework to support ideo-on-demand (IPTV) as a service using server virtualization and application virtualization approaches. Currently, service providers are looking to acquire new technologies like cloud computing and virtualization for delivering multimedia application such as Video on-Demand (VoD). In fact, the results of the case study developed using OPNET Modeler at the Media Communications Laboratory (Phooi Yee Lau et al., 2010), indicate that clouds improve the process of files streaming in Internet

Protocol Television (IP-TV) (Zeadally S. et al., Dec. 2011), because these files are no longer located in a distant server but they are saved in the cloud's database.

Using these trends, videos can be streamed from any part of virtual servers while infrastructure will handle the peak loads, avoid the overload and guarantee a high level of QoS. As a consequence, researchers agree that the use of virtualization can obviously reduce the response time of streaming and enhance the interaction with clients by providing efficient virtualized multimedia services.

In this project, we propose a new virtualized architecture for Interactive Voice response (IVR) applications (Fatna Belqasmi et al., Dec. 2011) (Fatna et al., Sept. 2012). To the best of our knowledge, there is no full-fledged cloud environment that enables the development, management and offering of the full range of IVR applications. In the following section, we will describe IVR applications and give some example of its services.

1.2.4 Interactive Voice Response (IVR)

Many enterprises in private and public sectors, such as banking, call centers and utilities use IVR applications that allow interactions with a wide range of automated information systems via keypad or by voice commands (Atel P.B., Marwala, T.,Oct. 2008). IVR can guarantee a superior quality of service and improve customer satisfaction, because it reduces callers waiting time, serves multiple users simultaneously and extends business hours. Users have the possibility to use either cell phones or soft phone like X-lite, Yate, EyeBeam, etc. to communicate with automated attendant. IVR can support multiple language options that it will use to answer the customer in real time (Trihandoyo A. et al., May 1995). IVR has various features which change from IVR system to another; some of them are described in this section:

- **Speech Recognition:** IVR is able to translate user's speech into words that automated system can understand. It addresses many types of languages, accents, grammars, voices, etc.

- **Voice Messaging:** IVR provides a mean to leave a voice message to a specified destination.

- **Voice to Email:** It allows translating users' speech to email that will be sent to a preconfigured address.

- **Text to Speech:** IVR can read texts and transform them into voice messages and deliver them to users by playing audio files with high quality of voice.

There are several examples of IVR application deployment such as automated attendant, automated survey and automating meter reading. In (Fatna Belqasmi et al., Dec. 2011) (Fatna et al., Sept. 2012), authors propose a virtualized infrastructure for IVR applications in clouds that ease IVR applications development and management. This infrastructure is composed of three layers (substrate, infrastructure, and platform) and an IVR substrate repository. The substrate layer provides IVR substrates that can be composed and assembled on the fly to build IVR applications. These substrates are accessible via the infrastructure layer. The platform layer for its part adds one or more abstractions and makes the substrates available to the IVR applications' developers while the IVR substrate repository is used to publish and discover existing IVR substrates. The three layers communicate via three planes: service, composition and management. The service plane handles the service execution, including coordinating the execution of services that involve several substrates; the composition plane intervenes in the composition of the appropriate substrates to create a given IVR application and the management plane is responsible for the actual control and management of substrate resources. It allows the instantiation of IVR applications and related substrates, enables fault and performance monitoring, and performs accounting for charging purposes.This architecture is based on RESTFul Web Service that will be described in the next section with more details to justify its use in the proposed solution.

1.2.5 RESTFul Web Services

RESTFul Web services follow the Representational State Transfer (REST) design paradigm. REST uses the Web's basic technologies (e.g. HTML, XML, HTTP, and URIs) as a platform to build and provision distributed services. It is one of the players of Web 2.0, a concept that promotes interactive information sharing and collaboration over the Web, as well as Web application consumption by software programs (Haibo Zhao, Doshi, P., July 2009). REST adopts the client-server architecture. REST does not restrict client-server communication to a particular protocol, but more work has been done on using REST with HTTP, as HTTP is the primary transfer protocol of the Web (L. Richardson and S. Ruby, May 2007).

RESTFul Web services can be described using the Web Application Description Language (WADL). A WADL file describes the requests and the sources provided by a service and the relationship between them such as the service's URI and the data's service (W3C Member Submission, 2009). REST supports a wide range of representation formats, including plain text, HTML, XML and JavaScript Object Notation (JSON). JSON is an open standard data interchange format for representing simple data structures (e.g. linked lists) and associative arrays (i.e. a collection of pairs e.g., keys, values) (L. Richardson and S. Ruby, May 2007). RESTFul Web services are simple and easy to use for clients because they are based on the well known Web standards (e.g HTTP, XML). RESTFul Web services are perceived to be simple and easy for clients to use because REST leverages existing well known Web standards (e.g. HTTP, XML) and the necessary infrastructure has already become pervasive. RESTFul Web services' clients (i.e. HTTP clients) are simple and HTTP clients and servers are available for all major programming languages and operating system/hardware platforms (L. Richardson and S. Ruby, May 2007).

IVR is an interesting technology in telecommunication applications as it provides many benefits as mentioned above. The motivation of this project is to develop virtualized infrastructure that ease IVR application deployment and management in the cloud environment. Thus, IVR service providers can build their services quickly, easily and with lower cost.

Furthermore, the virtualized infrastructure will give infrastructure providers the ability to instantiate on-demand IVR substrates and run them simultaneously. To guarantee efficient use and sharing of this infrastructure, resources allocation and task scheduling shall be supported. However, to the best of our knowledge, these two issues have not been addressed for such virtualized infrastructure. Therefore, the next section gives a general overview about some related work to the task scheduling and resources allocation in cloud environments.

In last decade, a lot of approaches have been proposed to control the resource utilization and ensure that cloud providers are offering a high level of QoS. In this subsection, we describe some resource allocation approaches and task scheduling algorithms at the substrate layer.

1.3 Resources allocation approaches

Resources allocation is one of the fundamental issues in cloud computing. To share physical resources between multiple guests, the substrate layer should identify the resources needed, verify resource availability and then allocate the appropriate resources. Several researchers have tackled this issue.

Kelly mechanism (Sichao Yang, Hajek, B., August 2007) is one of the proposed resource allocation strategies in a communication network with a single operator and different buyers. Furthermore, this mechanism is one of the proposed models to improve the QoS and to scale applications so that they can handle the dynamic arrival of resource requests. Kelly's mechanism aims at maximizing the valuation function which allows measuring the satisfaction of requests while using the assigned resources. As a result, maximizing this function will increase the QoS and satisfy the network users' bids (buyers' bids). Basically, each network buyer sends a one-dimensional vector in which he specifies the parameters of his request in order to reduce the difference between the values of the allocation and the payment. Then, the network selects the optimized outputs which are defined as the sum of the buyers' valuations to get an equilibrium point that allows the network provider distributing the physical resources adequately between the buyers. In fact, an efficient resource distribution between the VMs is not a linear problem as it depends on multiple parameters. It is an NP-hard problem. To solve this type of problems, some heuristic approaches and approximation methods have been developed such as the genetic algorithm (GA) which can be successfully used to get an optimal solution that may satisfy the user resource needs.

In (D. Dutta and R.C. Joshi, Feb. 2011), authors proposed to use the GA algorithm to balance the assigned resources between VMs. This algorithm allows managing dynamically the mapping between VMs and physical resources based on their workload changes while optimizing the use of the resources.

In (Zhen Ye et al., 2011), authors proposed two algorithms to monitor the resources allocation mechanism for the SaaS (Software as a Service) in the cloud environments in order to maximize profits. The first algorithm is based on minimizing the cost of the physical resources by reusing VMs which have the maximum available space to host new services. However, this algorithm may decrease the resource utilization or the profit of the server by assigning these VMs to host tasks that require less that the resources offered by the assigned VMs as the profit is defined as the ratio between the allocated resources and the required ones. As a result, another algorithm was proposed in the same work and it looks for the minimum available space in the server that can host the new services while minimizing the cost and maximizing the profit. In fact, it assigns the minimum of the available space which can minimize the number of SLA violations over the cloud environment.

Meanwhile, the relationship between the cloud service provider and cloud service consumer must be defined by a contract that describes formally the level of service needed. That’s why, it’s quite necessary to define this document which includes the terms and conditions to deliver the services. Consumers have trust on the provider that they will guarantee a high level of service as described in the SLA contract (Cloud Computing Use Cases White Paper, 2010). To classify applications processes in cloud computing, we have to determine their QoS level that has three parameters: execution, network and storage. A proposed model in (Zhen Ye et al., 2011) can calculate these values in cloud computing and classify the level of services under the umbrella of different constraints. This model selects some service attributes (time, price, availability and reputation) and identifies their aggregation function. Using these values, the model uses the GA to get an optimal solution to ensure the QoS for the hosted applications in the cloud. This approach guarantees that the delivered services are received by the consumer and by the provider as requested.

1.4 Task Scheduling Algorithms

Task scheduling is one of the key functions in operation of parallel and distributed computing systems. Most of the available techniques aim at identifying task assignment across multiple processors that guarantee minimum execution time of tasks and maximum of processor utilization.

As claimed in (D. Dutta and R.C. Joshi, Feb. 2011), GA algorithm can provide a better scheduling in cloud computing environment, guarantee the satisfaction of customers’ requests and maximize cloud providers profits. GA’s scheduling is based on vectors that indicate the appropriate resources and the associated users’ jobs. Then, it adequately schedules tasks to balance the traffic load across the processors that will execute them. In (M. D. Kidwell and D. J. Cook, 1994), authors propose task scheduling algorithm for system that receives tasks dynamically to optimize the resources utilization and minimize the execution time across different processors. The proposed scheduler distributes dynamically the tasks using a set of parameters such as time, task length, etc. The GA algorithm will send the task to specific processor or put it in a queue file. The main was to get schedules with optimal makespans which can minimize the total execution time among the processors. The results obtained in (M. D. Kidwell and D. J. Cook, 1994) show that the genetic algorithm can enhance the processor allocation by balancing the load across them so that one processor is dedicated to the scheduling calculations and the remaining processors will execute the received tasks.

In (Jia Yu et al., 2008) and (Albert Y. Zomaya et al., 1999), authors propose a framework based on genetic algorithm to solve scheduling problem for parallel processor systems and to highlight the condition under which this algorithm outperforms other heuristic algorithms. GA-based algorithm is used to equally spread the workload among different processors in order to maximize their utilization and minimize the task execution time. The proposed algorithm is based on minimizing the difference between the heaviest-loaded and the lightest-loaded processors so that the server can execute the received request in the minimum required time. Moreover, it shares the tasks among the processors so that it transfers tasks from a processor which has the greatest load to another one that has the lightest load in order to get an equilibrium level across the different processors. The results show that the proposed mechanism to balance the load in the framework has been very effective as it can reduce clearly the execution time especially in the case of large number of tasks. We also noticed that the GA is an effective solution to schedule waiting list which are long while other heuristics fail and don't get the optimal solution.

In (D. E. Goldberg, 1989), Yujia et al propose to use the genetic algorithm to solve the problem of task scheduling in cloud computing settings. In this work, the algorithm is used to better distribute the tasks in the Haddoop which is an open source framework that deals with data-intensive distributed applications. It supports a huge number of computational computers which are independent with large amount of data. The new proposed scheduler makes a scheduling decision by evaluating all the tasks while getting the shorter makespan and the short scheduling policies.

The proposed model is made of 5 components: 1) the System model which describes the data in the cloud nodes and facilitates the conception of the workload and the data location models, 2) the Task model which collects the data related to the tasks and their dependencies and especially the related computing factors which can influence the execution process, 3) the predicted execution time model which can predict and estimate the computation time and the required time for the communication between the distributed nodes in the cloud, 4) the Objective Function which can evaluate the completion time for all the tasks and the makespan, 5) the Schedule Optimizer which has to optimize the proposed scheduling based on the obtained makespan in 4).

The proposed model can get a better load balancing across all the nodes in the Haddoop framework based on the genetic algorithm performances. Thus, the GA algorithm is used to spread the workload across the processor so that it can maximize their utilization. We have to notice that the GA parameters are very important in the GA calculations as it cannot influence only the time required to execute the algorithm, but it can also affect the quality of the obtained results. GA parameter can be the population size, the type of permutation, generation number, crossover probability, etc. The experimental results in this work show the performance of the GA compared to the FIFO algorithm.

In (Yang Gao et al., Sept. 2009) and (Joanna Kolodiej and Samee Ullah Khan, 2012), authors claimed that heuristics rules and linear programming cannot solve efficiently the problem of task scheduling in manufacturing grids because they can’t adapt the dynamic environment as at each state the system has new information and new unpredicted status. Therefore, in (Yang Gao et al., Sept. 2009), authors proposed to use the genetic algorithm to manage the scheduler for a layered hybrid ant colony in manufacturing grid. The proposed algorithm is known as the layered hybrid ant colony and genetic algorithm (HACGA) and it is structured as follow: the first layer is dedicated for the ant colony algorithm to select the node machine in the grid that will handle the execution of the service and the second layer uses the GA to make the execution plan for the received tasks based on the real-time status of the resources. The HACGA in the manufacturing grid has a multi-objective optimization function which realizes the collaborative search with the neighborhood search to combine the results and get the optimized job-shop scheduler.

Nevertheless, in (Joanna Kolodiej and Samee Ullah Khan, 2012), they developed a hierarchic-genetic model to generate schedulers for the distributed grid computations namely the Hierarchic Genetic Scheduler (HGS-Sched). The main idea in this work as well as in (F. Xhafa et al., 2008), (S. Prabhu, 2011) and (S. Tayal, 2011) is to launch a concurrent search in the grid system by activating several processes which will explore the computation nodes and collect the partial solutions that can be optimized based on the proposed optimization function. The HGS is used to get the optimal solution for permutation flowshop scheduling and some other practical engineering problems as mentioned in (Joanna Kolodiej and Samee Ullah Khan, 2012). It can provide an effective distribution of computations tasks all over the grid nodes in order to minimize the execution time and maximize the resource utilization.
Several research projects tackle the task scheduling issue in cloud for many applications like workflow and e-learning applications (E. Barrett et al., 2011), (J. Yu and R. Buyya., 2006) and (O. Morariu, C. Morariu and T. Borangiu, 2012), but no resource optimization mechanism is provided in order to guarantee both efficient task scheduling and resource usage.

1.5 Conclusion

In this chapter we gave a general overview about virtualization technology and the cloud computing paradigm and the IVR applications and their benefits. Afterwards, we discussed the main resource allocation and task scheduling across processors or computers in non-virtualized and virtualized environments. The next chapter is dedicated to describe our proposed architecture to deploy virtualized infrastructures for IVR application.

CHAPTER 2

PROPOSED ARCHITECTURE AND SOFTWARE ARCHITECTURE

2.1 Introduction

The previous chapter gives a general description about the main topics to my thesis and reviews the most relevant related work. Then, we conclude that there is no full-fledged cloud environment that enables the development, management and offering IVR applications. That's why, a new architecture for virtualized infrastructure of IVR applications is proposed in this chapter. We start by giving a general overview to describe the proposed architecture which relies on proposed business model. Then, we describe the functional entities of the proposed architecture, planes, interfaces and the operational procedures in the proposed architecture. Finally, we describe the software architecture that relies on the proposed virtualized architecture for IVR applications.

2.2 Proposed Architecture

2.2.1 General Overview

Our thesis describes a novel architecture for a virtualized IVR infrastructure (Fatna Belqasmi et al., Dec. 2011) as a first step towards the deployment of full-fledged IVR applications in cloud settings. Figure 2.1 depicts our vision:

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

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

- The middle layer is the platform layer. It includes graphical user interfaces (GUIs) and application programming interfaces (APIs) that may ease the development and management of the applications in the top layer.

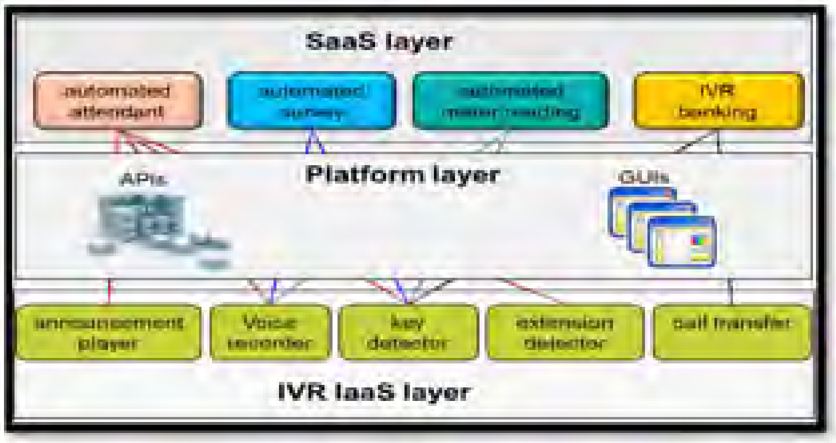


Figure 2.1 Different services share the same substrates

2.2.2 Business Model

2.2.2.1 Analysis of Existing Business Models

There are several business models which describe the different parts involved to provide services and the relationships between them. However, in the conventional network there are two main types of business model which are the telecommunication business models and the data communication business models (El Barachi, M. et al., Dec. 2010).

In the Telecommunication domain, the main existing business models are TINA and Parlay models (El Barachi, M. et al., Dec. 2010). The TINA business model describes architecture where different actors are separated and act in the same domain to provide flexible and agile services in the world of telecommunication and information (Abarca, C., Peters, M., 1999). This model has specified several business role: the client is the entity that will use the service or the entity that have the agreement of the service, the retailer is the entity providing the service and it has also an agreement with the clients for service usage, the third party provider that has an agreement with the retailers to provide the services and the broker is the entity that contains information about the providers and the services (El Barachi, M. et al., Dec. 2010). In (Van Halteren et al., 1999), authors used the TINA business model combined with the Web to provide the Value Added Web (VAW) to provide more sophisticated and enhanced services for users while guaranteeing a high level of QoS and facilitating their use.

Parlay applications programming interfaces use the concept of the programming services and aim to facilitate the development of applications for telecom network based on some technologies such as CORBA, java and web service with their related tools (Sungjune Hong et al., Nov. 2005). It has 3 main business roles: the client who consumes the services offered by the Parlay service provider which can be the operator subscribing to the service and the framework operator which handles all the required operations and procedures to execute the services (El Barachi, M. et al., Dec. 2010).

In the data communication domain, the most important business model is the Web Service (WS) that aims to realize a Service Oriented Architecture (SOA) where services are deployed, discovered, published and composed. It is based on 3 main entities: the WS provider who offers services for the customers and the WS Registry which manages the relationship between the client and the provider (El Barachi, M. et al., Dec. 2010) (Karunamurthy, R. et al., July 2007). The WS is based on the concept of service composition that means the reuse of the existing services to create and obtain a new integrated service.

The TINA and the WS business models can be considered as powerful models because they enable the separation between the different actors and operators in network. In addition, both of them use the concept of the broker which contains the related information about the created services. Also, the composition model in the WS enable the creation of different types of services and enable the reuse of the published service to get new ones. Therefore, inspired by the benefits and concepts of the TINA and WS business models we will propose a new model for the virtualized infrastructure for IVR applications.

2.2.2.2 Design Goals and proposed Business Model

In this section, we will describe our business model/design which is based on the WS especially on the RESTFul Web Service that was described with more details in the first chapter. One of the first design goals we have in mind is that different IVR applications in different domains should be able to share substrates, as illustrated by Figure 2.1. Conversely, an IVR application should also be able to use many instances of the same substrate, for scalability. Another design goal is that it should be possible to publish and discover substrates and substrate instances. Yet another goal is that IVR service providers should be able to compose the substrates available in the infrastructure into powerful IVR applications, using appropriate platforms. Figure 2.2 shows the proposed business model (Fatna Belqasmi et al., Dec. 2011).

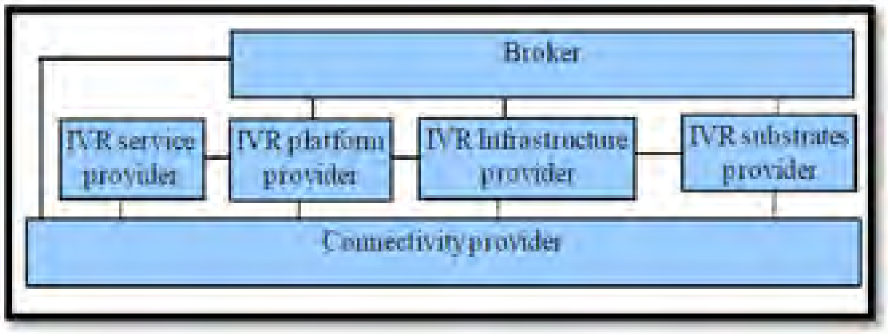


Figure 2.2 Proposed business model

The IVR service provider offers IVR applications as SaaS, accessible by end-users and other applications. It develops and manages these applications using the IVR platform offered by IVR platform providers. The platform adds levels of abstraction to the IVR infrastructures in order to ease IVR application development and management by IVR service providers.

IVR substrates are offered by IVR substrate providers to IVR infrastructure providers. A given IVR substrate provider may interact with several IVR infrastructure providers and offer them the same substrates, since these substrates are sharable. A given IVR infrastructure provider may also interact with several IVR substrate providers. The broker enables the publication and discovery of substrates so that the providers can reuse them. The connectivity provider enables connectivity between the different actors.

2.2.3 Architectural Components and Interfaces

2.2.3.1 Overview

Figure 2.3 shows the proposed architecture. It includes two layers, three planes architecture and a repository. The first layer contains the functional entities that realize the infrastructure provider role. The second layer is comprised of entities that realize the IVR substrate provider role. The interactions between the two layers are organized via three planes: service, management and composition. The repository realizes the role of the broker. Entities, planes functionality, interfaces and operational procedures are describes in the next sections (Fatna Belqasmi et al., Dec. 2011).

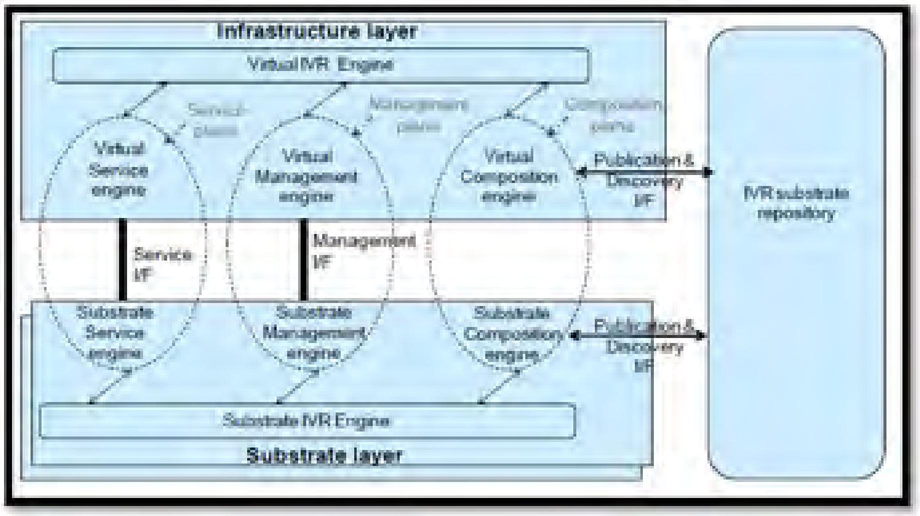


Figure 2.3 Overall architecture

2.2.3.2 Entities

The key functional entity of the first layer is the virtual IVR engine and the key entity of the second layer is the substrate IVR engine (Fatna Belqasmi et al., Dec. 2011).

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

- **The substrate IVR engine**: it coordinates the activities of the substrate service engine, the substrate management engine, and the substrate composition engine.

2.2.3.3 Planes Functionality

We have three planes in the proposed architecture: service, composition and management planes (Fatna Belqasmi et al., Dec. 2011):

- **Service Plane:** the main functionality handled in the service plane is mediation. IVR infrastructure providers may decide to make substrates available to platform providers using interfaces other than the original interfaces with which they were made available by substrate providers. In addition to mediation, the service plane also coordinates the execution of services that involve several substrates.

- **Management Plane:** it handles the actual control and management of substrate resources. It enables the instantiation of IVR applications and related substrates, and their configuration. It also enables fault monitoring, performance monitoring, and the accounting for charging purposes.

- **The composition plane:** it interacts with the repository and enables the publication and discovery of the substrates and substrate instances that are used in composition.

2.2.3.4 Interfaces

We have three types of interfaces in the proposed architecture to handle the interaction between the different planes (Fatna Belqasmi et al., Dec. 2011):

- **Service I/F:** The virtual service engine and the substrate service engine communicate via the interface supported by the substrate service engine. This is motivated by the fact that existing potential IVR substrates come with multiple interfaces (e.g. VoiceXML, Session Inition Protocol-SIP, Media Server Control Markup Language-MSCML). They communicate with the virtual service engine via mediators that are incorporated in the virtual service engine.

- **Management I/F:** A key requirement for the management interface is to accommodate a plurality of resource description mechanisms (e.g. XML, plain text). Another requirement is that the interface should be supported by commonly used virtualization servers such as XEN to ease the creation of instances. These requirements have led to our selection of RESTFul Web services as the natural choice.

- **Publication & Discovery I/F:** We have also decided to use RESTFul Web services for the publication/discovery interfaces to minimize the number of interfaces in our proposed architecture. The fact that RESTFul services support a wide range of resource description mechanisms is also an advantage when it comes to publication and discovery. The next sub-section provides more information on RESTFul Web services, since it is a key technology of our architecture.

2.3 Overall Software Architecture

We describe in this section the proposed software architecture for IVR application in virtualized settings, and then we describe the software operational procedures with a focus on the management phase.

2.3.1 General Overview

Figure 2.4 depicts our vision about the software architecture which is composed of the infrastructure layer, the substrate layer and the IVR substrate repository (Fatna Belqasmi et al., Dec. 2011).

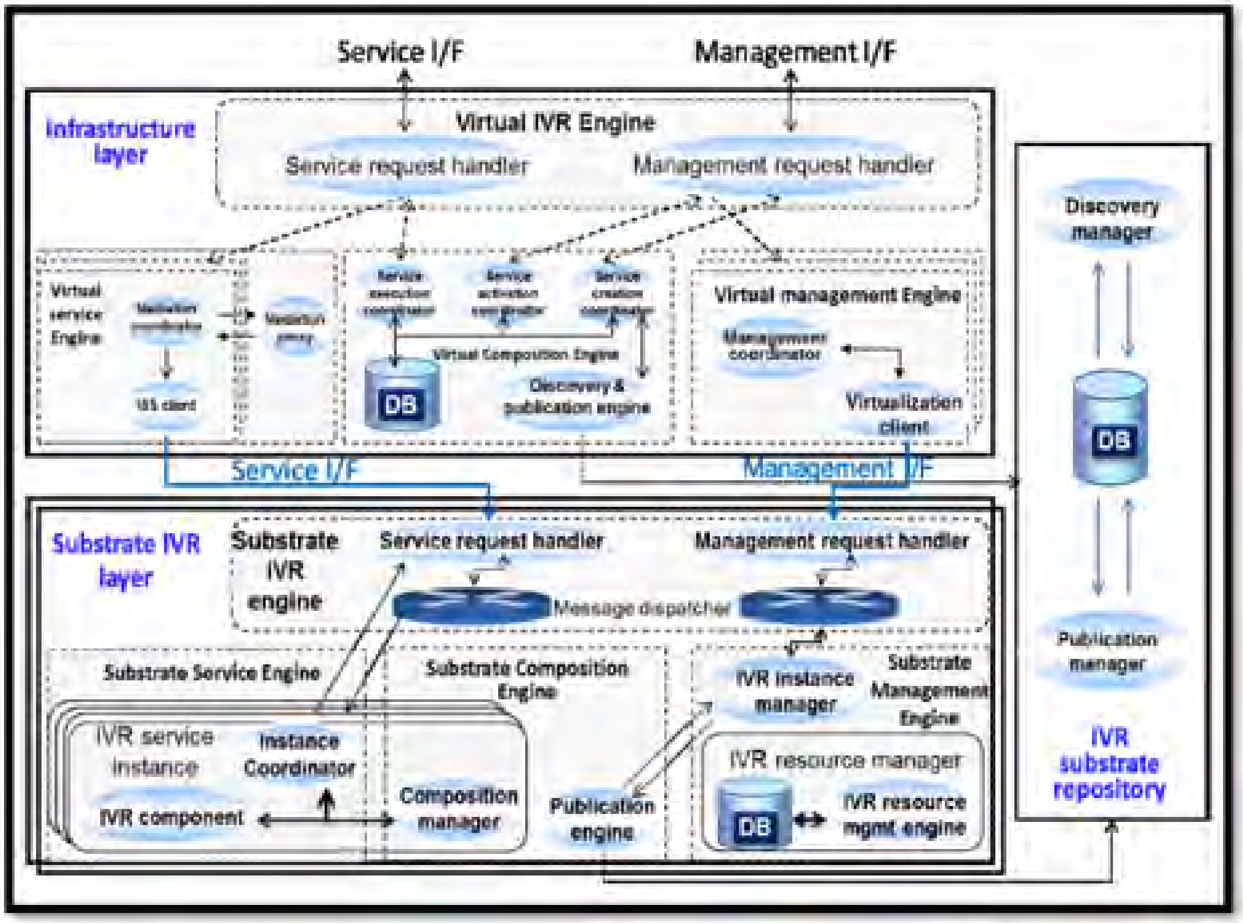


Figure 2.4 Overall Software architecture

2.3.1.1 Infrastructure Layer

The infrastructure layer is made of four engines: the virtual IVR engine, virtual composition engine, the virtual management engine and the virtual service engine. It's known also as the virtualization layer (Fatna Belqasmi et al., Dec. 2011).

- **Virtual IVR engine:** it handles and manages the received requests/responses from/to the platform layer. It includes two entities: the management request handler and the service request handler. The management request handler acts as management interface provided to the cloud platform provider, whereas the service request handler realizes the execution interface provided to the cloud platform provider.

- **Virtual composition engine:** it's composed of four main engines: the service creation coordinator, the service activation coordinator, the service execution coordinator and the discovery and publication engine. The service creation coordinator is responsible for managing and coordinating the composition of a new IVR service. The service activation coordinator is responsible for managing and coordinating the instantiation of new Substrate IVR Instance (SIIs) of the composed service. The service execution coordinator is responsible for managing and coordinating the execution of a given request. Finally, the discovery and publication engine is used to publish the virtual IVR services and to discover the available substrates from the repository.

- **Virtual management engine:** It's composed of two main entities: the management coordinator and the virtualization client. The management coordinator translates the requests received from the service activation coordinator into requests that the virtualization client must send later to the target substrate. The second entity handles the communication with a substrate at the creation and the activation phases of the SII.

- **Virtual service engine:** it includes two entities: the mediation coordinator, and the SII client. The mediation coordinator translates the request received from the service execution coordinator into requests and the SII client should send them to the target substrate. The SII client allows the communication with a substrate at execution. A mediation proxy is used to manage the execution of substrates from many substrates provider to provide a composed virtual IVR service.

2.3.1.2 Substrate IVR Layer

The substrate layer is made of the substrate IVR engine and the substrate composition engine, the substrate management engine and the substrate service engine (Fatna Belqasmi et al., Dec. 2011).

- **Substrate IVR engine:** It includes a message dispatcher which dispatches the received requests to the appropriate IVR instance manager. It includes also two main entities: the management request hander and the service request handler. The first engine offers a management interface for virtual IVR provisioning. The second engine is responsible to handle the service execution at the substrate level.

- **Substrate management engine:** it's composed of an IVR instance manager and IVR resources manager. Each substrate is created and managed by a separate IVR instance manager. The IVR resources manager controls the access to the IVR substrate resources. It monitors and controls the availability of the physical resources and their assignment. It manages the resources use and solves problems that may affect an IVR instance (e.g., traffic overload on a VM).

- **Substrate composition engine:** it includes two essential entities: a publication engine and the composition manager. The publication engine is an entity which is responsible for publishing the IVR substrates at each substrate provider; it handles also the publication of the created instances by the IVR instance manager. The composition manager is responsible for creating new composed service component from existing ones at each substrate provider. It is a part also of the substrate service engine.

- **Substrate service engine:** it is composed of an instance coordinator, an IVR component and the composition manager from the substrate composition engine. The IVR component can be defined as a set of services like announcement player and voice detector. It handles the coming request through service interface from end users. The instance coordinator can identify the required service component to execute the received requests.

2.3.1.3 IVR Substrate Repository

The repository or the broker are made essentially of two entities: the publication manager and the discovery manager (Fatna Belqasmi et al., Dec. 2011).

- **Discovery manager:** it handles requests to discover the published service in the database and get their descriptions.

- **Publication manager:** it is used to publish into the database the new created IVR substrates and the composed services including their description files to be discovered and used by other IVR providers or other applications.

Our architecture covers four main operation procedures including application creation, activation, management and execution. The following section gives an overview about these phases.

2.3.2 Software Operational Procedures

In this section, we focus on the description of the management architecture part which is the main topic of this report. The other parts of the overall architecture are described in (Fatna Belqasmi et al., Dec. 2011) and (Fatna et al., Sept. 2012).

2.3.2.1 Creation

This phase is executed at the IVR Infrastructure layer. When the management request handler gets a creation request, it sends it to the service coordinator in the virtual composition engine. The service coordinator sends a discovery request to the discovery engine in the broker to get all the available substrates from the IVR substrate repository and sends them to the platform provider through the management request handler. In addition, when service providers compose a service, the service creation coordinator will take the inputs from the platform provider GUI for the service composition, create the description file for the composed service along with the request plan and store them in the local database of the infrastructure layer (Figure 2.5). The chosen IVR service substrates, the execution sequence of the composed service and the services description are all described in these files (Fatna Belqasmi et al., Dec. 2011).

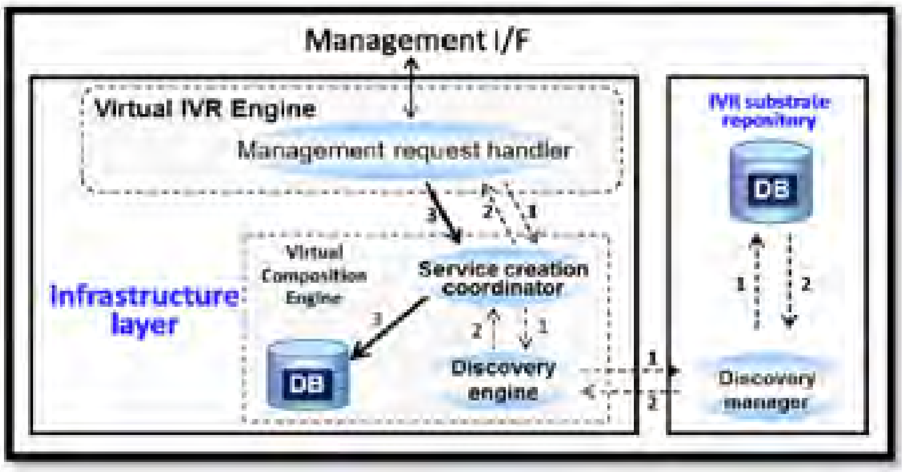


Figure 2.5 Creation Phase

Figure 2.6 depicts the architecture of our case study. This architecture is composed of the four substrates (i.e. announcement player, voice recorder, key detector, extension detector). We assume that the substrates are supplied by substrate providers SubP1, SubP2, SubP3, SubP4, and SubP5, respectively. We further assume that we have one infrastructure provider (InfP), one platform provider (PP), two service providers (ServP1, ServP2) and one end-user with a subscription to one of the two service providers (Fatna Belqasmi et al., Dec. 2011). IVR substrates are described using WADL and also with Donkey State Machine (DSM).

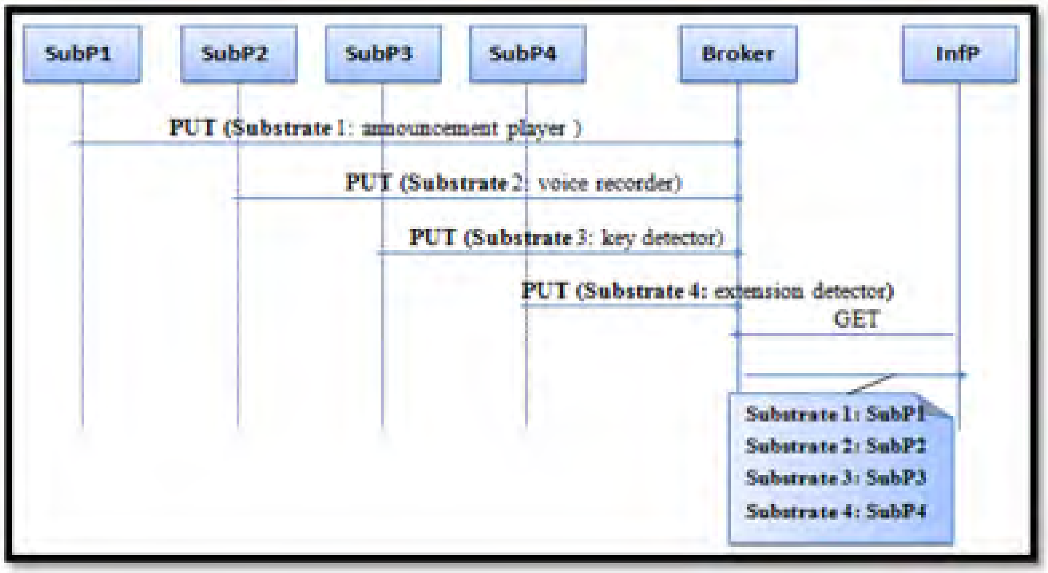

SubP1
SubP2
SubP3
SubP4
Broker
InfP
PUT (Substrate 1: announcement player)
PUT (Substrate 2: voice recorder)
PUT (Substrate 3: key detector)
PUT (Substrate 4: extension detector)
GET
Substrate 1: SubP1
Substrate 2: SubP2
Substrate 3: SubP3
Substrate 4: SubP4

Figure 2.6 Substrate publication and discover

The DSM description represents the substrate behaviour as a state machine and is included under the <doc> element of the WADL description. The use of DSM is motivated by the fact that it is used by the SIP Express Media Server (SEMS) used in our prototyping environment, and because it makes the substrates' composition easier. DSM enables a textual description of applications (substrates in our case) that can be directly executed by interpreters hosted by the SEMS (Fatna Belqasmi et al., Dec. 2011). Figure 2.7 shows a simplified WADL description of the 'Announcement Player' substrate. In Figure 2.6, IVR substrates are published to the repository using a PUT request. The substrate composition engine is the entity responsible in the creation phase. Next, the virtual composition engine of the infrastructure provider sends a GET request to broker to discover all the stored substrates by different substrate providers (Fatna Belqasmi et al., Dec. 2011).

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

Figure 2.7 WADL description of the ‘Announcement Player’

2.3.2.2 Activation

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

IVR service substrate. At the IVR substrate layer, when the management request handler gets an instantiation request, it forwards it to the dispatcher which forwards it to the appropriate IVR management engine (Figure 2.8) which creates the new SII (Fatna Belqasmi et al., Dec. 2011).

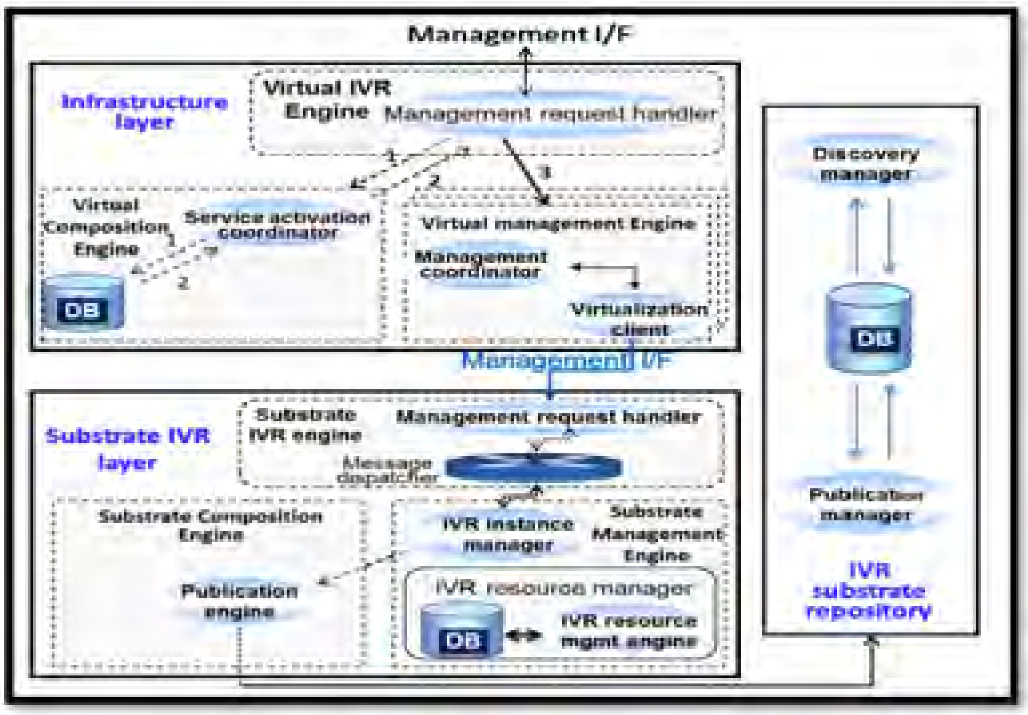


Figure 2.8 Activation Phase

Figure 2.9 describes the flow for this application activation. A new IVR substrate is instantiated by sending a POST request to the appropriate substrate along with the DMS description of the service instance to create and the identifier of the IVR service provider. When a substrate management engine receives an instantiation request, it checks resource availability then, it allocates the necessary resources to create the new SII. Then, the activated IVR is published in order to be used by other end users or other applications (Fatna Belqasmi et al., Dec. 2011).

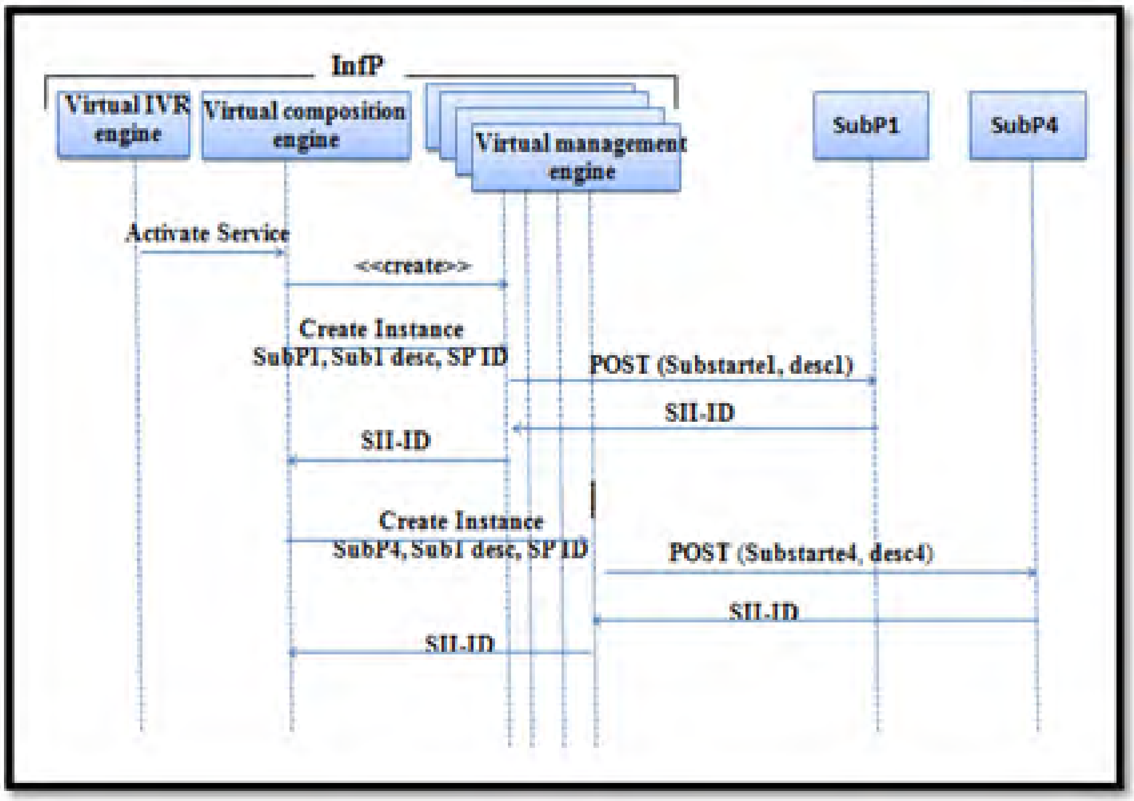


Figure 2.9 Service Activation

2.3.2.3 Management

The IVR instance manager receives a creation request then it forwards it to the IVR resource manager which will check if there are enough resources available to handle the coming request. The IVR resources manager is composed of these main engines: resource computational and instantiation scheduler. The resources computational engine is responsible for calculating and estimating the minimum of the required resources for each new IVR substrate instance. It includes two main entities as shown in the Figure 2.10: the resources optimization engine and The SLA evaluation engine.

The resources optimization engine computes the required resources based on a specific algorithm that will be described and discussed in the next chapter. It checks at first the availability of the physicals resources by sending a request to the database of the IVR resources manager. If there are enough resources, it runs an algorithm in order to estimate the appropriate combination of CPU, memory, bandwidth and disk space to support the coming request

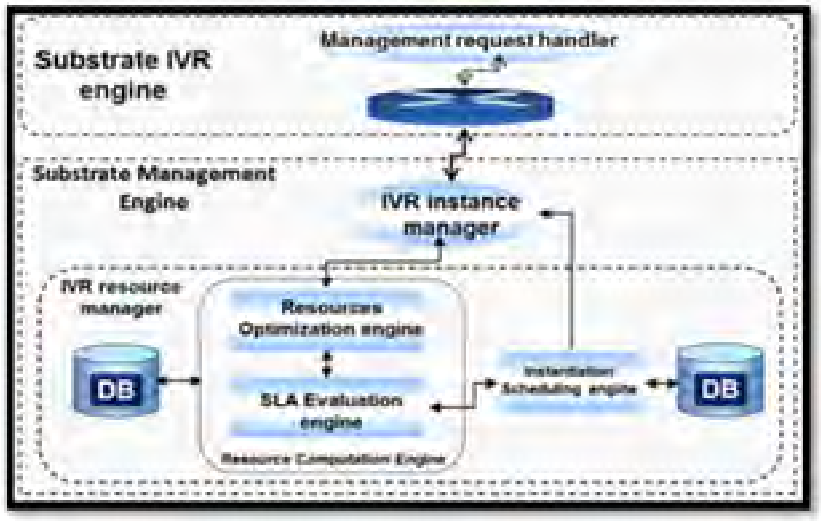

Substrate IVR engine
Management request handler
Substrate Management Engine
IVR instance manager
IVR resource manager
Resources Optimization engine
SLA Evaluation engine
Resource Computation Engine
Instantiation Scheduling engine
DB
DB

Figure 2.10 Management Phase

Then, the SLA evaluation engine receives a request from the resources optimization engine including the chosen solution to evaluate the computation fitness function. This evaluation engine compares the obtained results with the required SLA. If the chosen solution meets the required criteria and maximizes the utility of the substrate, it will be sent to the instantiation scheduler engine to create the IVR substrate instance. If this solution can't fulfill the required criteria the SLA evaluation engine should send a new request to the resource optimization engine to find new values of CPU, memory, disk space and bandwidth that satisfy the activation request criteria.

The instantiation scheduler engine receives the IVR creation requests from the resources computational engine to instantiate the IVR substrate instances. It has to optimize the allocation of processors by identifying the task assignment that can guarantee the maximum utilization of processors and the minimum execution time of tasks based on a fitness function that calculates and evaluates the task scheduling. It uses a task scheduling algorithm that will be described in the next chapter. This algorithm distributes the tasks across the processors and gives as a result a vector that contains the instantiation tasks and the processor for each task that will execute the IVR substrate creation request.

Lastly, the IVR resources manager sends the results of the two algorithms to the IVR instance manager. In other terms, it sends values of the minimum required resources that can handle the coming requests in one hand and guarantee a high level of QoS for the new created IVR substrate instances in the other hand. It also sends IVR creation tasks assignment across the processors that minimizes the execution time and improves the processors utilization. In Figure 2.11, the IVR instance manager sends the creation requests including this information to the server hypervisor to instantiate the virtual machines for the coming IVR applications. Once, the IVR substrate instance is created, a notification request is sent by the hypervisor to the IVR instance manager which forwards it to the publication engine to publish the new created instances information in the substrate layer.

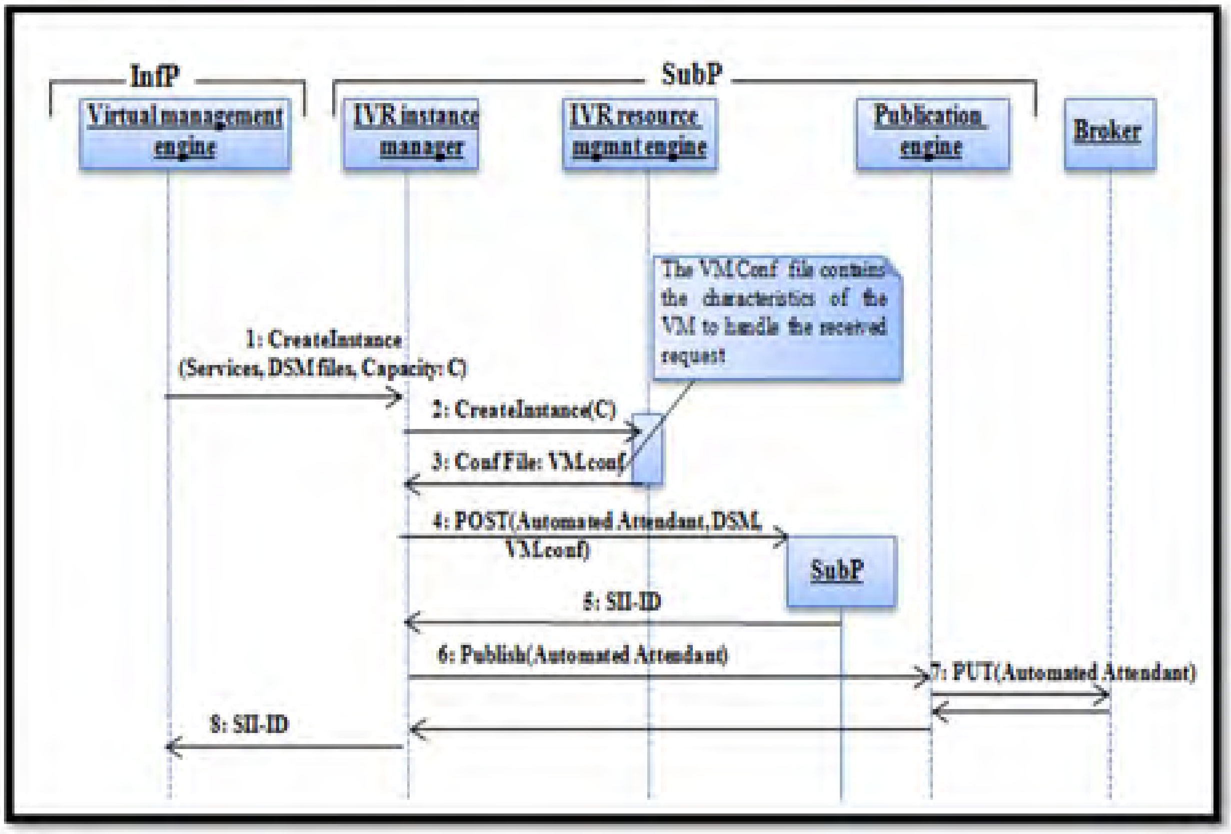

InfP
SubP
Virtual management engine
IVR instance manager
IVR resource mgmnt engine
Publication engine
Broker
The VMConf file contains the characteristics of the VM to handle the received request
1: CreateInstance (Services, DSM files, Capacity: C)
2: CreateInstance(C)
3: Conf File: VMconf
4: POST(Automated Attendant, DSM, VMconf)
SubP
5: SII-ID
6: Publish(Automated Attendant)
7: PUT(Automated Attendant)
8: SII-ID

Figure 2.11 Service Management

Figure 2.11 describes the flow for this management application. As shown in this figure, the virtual machine sends a creation request to the IVR instance manager in the substrate layer. The IVR instance manager forwards this request to the IVR resources manager to verify if there are enough physical resources on the server to handle the received request. Then, it executes the resources management algorithms in order to estimate the minimum of the required resources and to assign IVR creation tasks across the server processors to minimize the execution time and to improve processors performances. A configuration file which contains the characteristics of the required VM for the received request is sent by the IVR resources manager to the IVR instance manager. Moreover, the IVR instance manager will send the creation request to the server hypervisor that creates the instance and returns the SII-id of the created instance to the IVR instance manager. Finally, the instance manager will send a publication request to the virtual management engine which forwards it to the publication engine in order to publish the new created instance, the SII-id and the services that the new IVR can offer to the end users.

2.3.2.4 Execution

At the infrastructure layer, when the service request handler gets an IVR request to execute for instance play announcement and collect digit, it asks the service execution coordinator, for the IVR substrates that provide the virtual SIIs. Accordingly, the service execution coordinator retrieves the information which was stored in the DB when the virtual IVR was created and sends it to the virtual IVR engine. This includes the coordinates of the substrate, the type of the substrate, the instance ID, the substrate provider ID, the client address. Moreover, the virtual IVR engine creates a virtual service engine instance to communicate with each of the substrates, and instructs the different instances to execute the appropriate sub-requests, following the request plan. A request plan is a set of sub-requests and their execution sequence, along with the relevant substrates/SIIs that are required to answer an IVR request. The request plan is created by the virtual composition engine during the service creation phase. When a substrate service engine receives a service execution request, it forwards the request to the appropriate SII, which then executes the request and replies back to the virtual service engine (Figure 2.12). The flow for this execution application is given in Figure 2.13 (Fatna Belqasmi et al., Dec. 2011) (Fatna et al., June 2012).

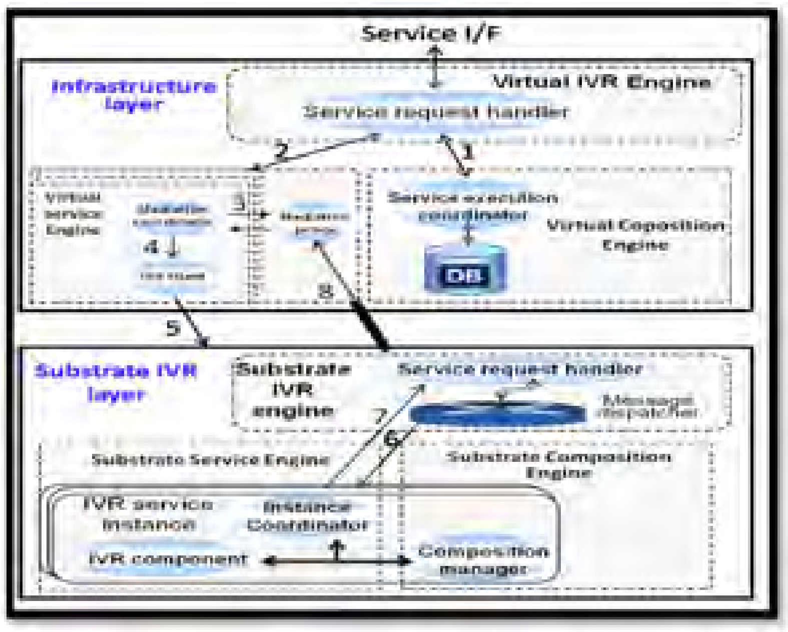


Figure 2.12 Execution Phase

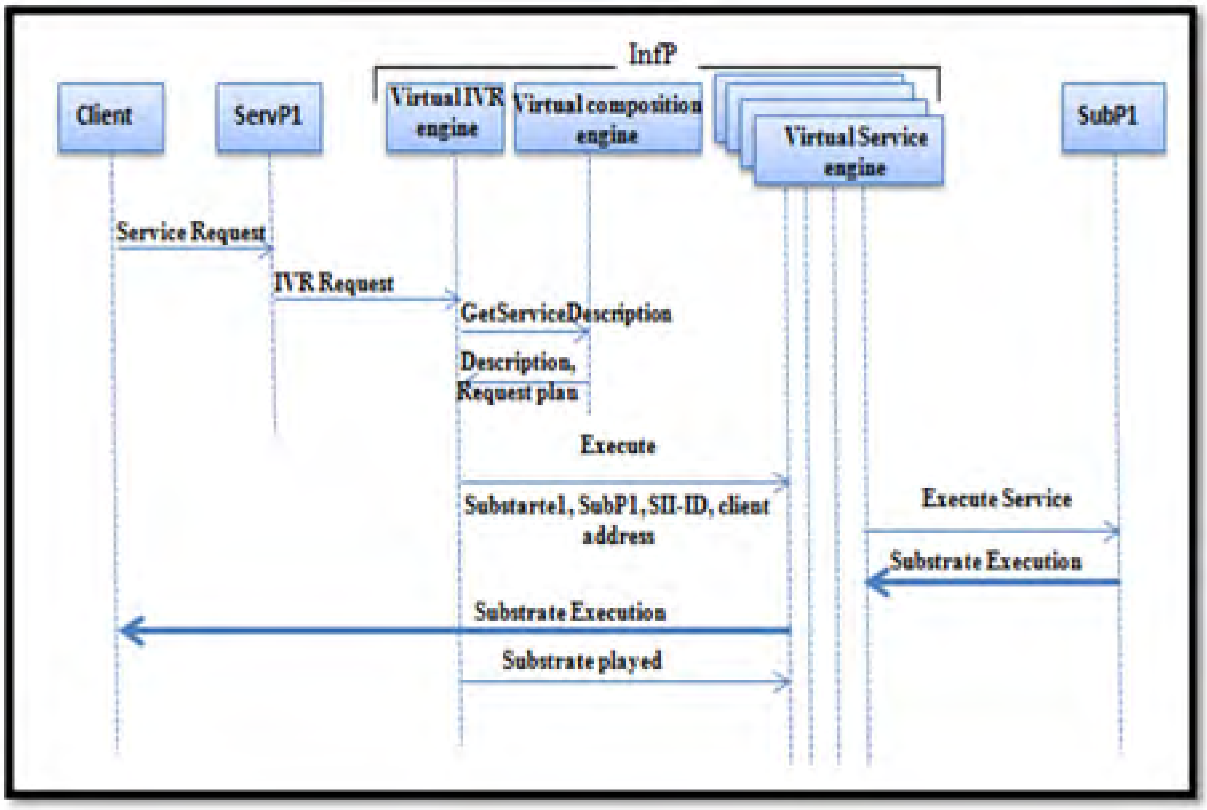


Figure 2.13 Service Execution

2.4 Conclusion

In this chapter, we first present our overall proposed architecture which is based on our business model that we described. We presented the main functional entities, planes, interfaces and the operational procedures of this architecture. We also described the overall software architecture including the layers, the main entities and the software operational procedures with a focus on the resources management architecture. In the next chapter, we will present our proposed algorithms to manage and control the use of the physical resources in the proposed virtualized architecture for IVR application.

CHAPTER 3

GENETIC-BASED ALGORITHMS FOR RESOURCES MANAGEMENT FOR VIRTUALIZED IVR APPLICATIONS

3.1 Introduction

In this chapter, we focus on resource management at the substrate layer. To instantiate new substrates, the substrate management engine should identify the needed resources, verify resource availability and then allocate the appropriate resources. We focus on two issues: computational resources sharing and task scheduling. In computational resource sharing, we deal with sharing existing computational resources (e.g. virtual machines, processors) between different IVR applications in an optimal way. The task scheduling relates to the assignment of the received instantiation requests. In this chapter, we propose IVR virtualization task scheduling and computational resource sharing (among different IVR applications) strategies based on genetic algorithms, in which different objectives are optimized. First, we will present our proposed resource management algorithm to estimate the required resources to create a new IVR instance. Then, we describe task scheduling algorithm to assign IVR creation tasks across processors.

3.2 Genetic-based Algorithms Objectives and Problem Statement

3.2.1 Genetic Algorithm Basics

Genetic algorithm (GA) is one of the most important algorithms used in optimization problem as it has been proven that it is a robust approach in heuristic searching (Leela, R. and Selvakumar, S., Jan. 2009). It is used to solve optimization problem in the industrial engineering systems especially machine learning and manufacturing which are complex and extremely hard to find optimal solutions in short time. In GA, a population of strings randomly generated from a set of potential solutions (represented by chromosomes) is used to create a second population, based on the fitness of each individual in the population and by applying different GA operators such as selection, crossover and mutation. The second population is then used by the algorithm to create a third one. The algorithm ends when a targeted fitness level is reached for the population (Pengfei Guo et al., 16-18 Oct. 2010) (Konfrst, Z., April 2004). The GA was introduced by John Holland in the early seventies (Leela, R. and Selvakumar, S., Jan. 2009), it starts with an initial population of chromosomes then it will apply operators on it to get different generations. These operators are basically the encoding, the evaluation, the selection, the crossover and mutation which are described with more details in the following part:

- **Encoding**: each chromosome should be coded as a binary or numerical string, the length of the string depends on the parameters number and their domain.

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

- **Evaluation**: Each generation is evaluated based on an objective function that aims to find the optimal solution. The fitness values are saved during all the iteration in order to be compared at the end of the algorithm execution and to get the best one.

- **Crossover**: the algorithm chooses two crossover points to selects genes from the parent chromosomes, these genes will be copied from one chromosome to another as shown in the following example:

Chromosome 1	01011 | 00100110110
	↕
Chromosome 2	11111 | 11000011110
Chromosome` 1	01011 | 11000011110
Chromosome` 2	11111 | 00100110110

- **Mutation**: it is performed after the crossover to switch randomly some genes from the obtained chromosomes to get more possible solution for the problem:

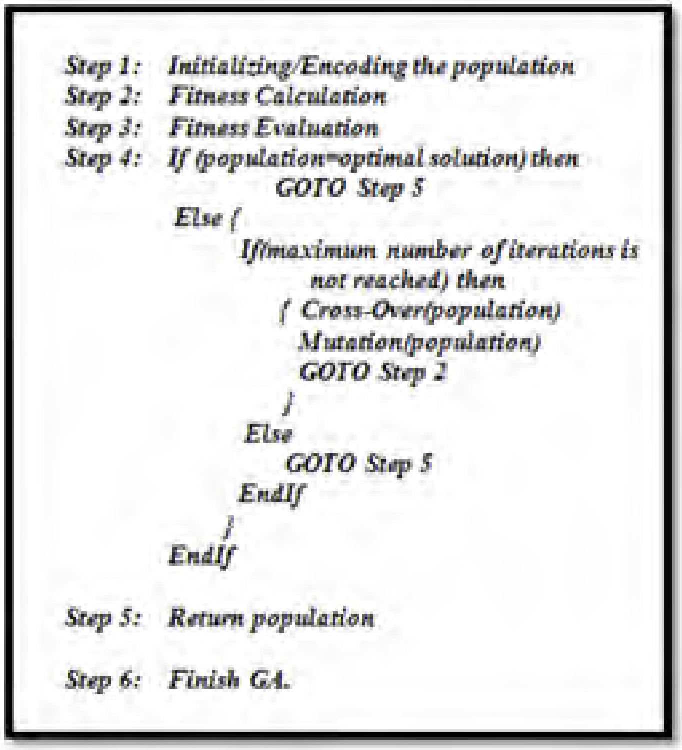
Chromosome 1	0101111000011110
Chromosome 2	1111100100110110
Chromosome` 1	0100111000011110
Chromosome` 2	1111101100110100

In this project, we propose using GA to optimize 1) the computational resource sharing; 2) the assignment of instantiation requests to different processors provided by the virtualization machine. For each algorithm, a specific fitness function and specific GA operators are used. In the computational resource sharing algorithm, a population is represented by the resources required by each SII to instantiate. In the task scheduling algorithm, a population is represented by the instantiation requests.

3.2.2 Genetic-based Algorithms Objectives

The computational resource sharing algorithm should allow the selection of the required resources for each SII (Substrate IVR Instance), while minimizing the amount of resources used as well as the resource allocation time and improving the utility of the assigned resources to get a high level of QoS. The task scheduling algorithm should minimize the execution time for the instantiation requests, by distributing the instantiation requests among the available processors as equally as possible. No processor should be underused while others are overloaded. These two algorithms are described in the next section. These algorithms will be used by the IVR instance manager and the substrate IVR engine, respectively.

These two algorithms are based on genetic algorithm (GA) that belongs to the stochastic search family as it is based on natural selection strategy. GA uses populations which are represented by set of potentials solutions or chromosomes to find the optimal solution for the proposed problem based on the fitness of each individual in the generated population. A specific fitness function and specific GA operators are used to look for the optimal solution. In the computational resource sharing algorithm, a population is represented by the resources required by each SII. In the task scheduling algorithm, a population is represented by the instantiation requests sequence. Figure 3.1 shows a GA algorithm procedure



```
Step 1:   Initializing/Encoding the population
Step 2:   Fitness Calculation
Step 3:   Fitness Evaluation
Step 4:   If (population=optimal solution) then
                    GOTO Step 5
           Else {
                 If(maximum number of iterations is
                        not reached) then
                     { Cross-Over(population)
                       Mutation(population)
                       GOTO Step 2
                     }
                  Else
                      GOTO Step 5
                 EndIf
               }
          EndIf

Step 5:   Return population

Step 6:   Finish GA.
```

Figure 3.1 GA algorithm procedure

In the next sections, we first discuss the computational resource sharing and then the task scheduling.

3.2.3 Problem Statement

We propose to use the following notations in the proposed algorithms:

- ***N*** is the expected number of users for a given IVR application.
- ***l*** is the expected call arrival rate for an ISS.
- t_n is the size in unit of time for the execution of a given instantiation request (task size).
- t_r is the time needed to compute the required resources for a given instantiation request.
- t_v is the time needed for the creation, configuration and activation of the appropriate virtual machine that will host a given SII ($t_n=t_v+t_r$).
- ***m*** is the number of processors that can be used to handle the instantiation requests.
- (CPU_r, M_r, B_r, D_r) represents the required resource for a given SII, in terms of CPU, memory, bandwidth and disk space, respectively.
- (CPU_c, M_c, B_c, D_c) represents the available capacities (i.e., the capabilities of the virtualization machine), in terms of CPU, memory, bandwidth and disk space, respectively.
- (t_{cpu}, t_M, t_B, t_D) represent the percentage of resource usage for a given SII:
    - t_{cpu} is the ratio of CPU_r over CPU_c.
    - t_M is the ratio of M_r over M_c.
    - t_B is the ratio of B_r over B_c.
    - t_D is the ratio D_r over D_c.

The computational resource sharing algorithm should allow selecting the required resources (CPU_r,M_r,B_r,D_r) for each SIIc, while minimizing the amount of resources used, minimizing the resource allocation time, and maximizing the satisfactory factor of the SII using a specific amount of resources. The task scheduling algorithm should minimize the instantiation requests' execution time (i.e. t_n), by sharing the instantiation requests among the available processors.

3.3 Computational Resource Sharing Algorithm

Each processor performs resource computation of each instantiation request. As a first step in the definition of the computational resource sharing algorithm for IVR applications, we performed a set of experimental measurements to quantify the resources used by a given number of SIIs. This was done using the prototype from our previous work. The measurements were then used as input to define the load measurement mathematical models and the resource computation algorithm to calculate the required resources for each instantiation request. We also defined a resource computation fitness function.

3.3.1 Load Measurement

Load measurement allows the quantification of the SII resource usage according to the number of users accessing the IVR substrate. It is performed to identify the required resources (CPU_r, M_r, B_r, D_r) for each SII. The experimental tests were executed on a system providing a set of SIIs, and that had the following capacity: CPU_c= 1 GHz, M_c= 512 MB, D_c= 20 GB and B_c=1 Gbps bit rate. Then, we measured the used resources (CPU_r, M_r, B_r, D_r) according to different call arrival rates. The call arrival rates were from 0 to 60 call/min. Results are given in figures, Figure 3.2, Figure 3.3, Figure 3.4 and Figure 3.5. From these observed data, we derive the functional model that describes the relationship between the number of users and the usage of each resources CPU, BW, Memory and Disk space and we get as a result the equations, (3.2), (3.3), (3.3) and (3.4). We started from the models given in (3.1) where y_{CPU} , y_M , y_B and y_D are respectively the CPU, memory, bandwidth, and disk space consumption in percentage according to call arrival rate (here the variable λ). We describe the given model for each resource type and we measure their consumption in percentage according to the call arrival rate. Then, we compute the *R-square* (coefficient of determination R^2) to assess the accuracy of the model and how well it fit the measured data. The closer the value of R^2 is to 1, the better the linear regression models the data.

$$\begin{cases} y_{CPU,\ M,\ D} = a_1\ \lambda^5 + a_2\ \lambda^4 + a_3\ \lambda^3 + a_4\ \lambda^2 + a_5\ \lambda + a_6 \\ y_B = a_1\lambda^4 + a_2\ \lambda^3 + a_3\ \lambda^2 + a_4\ \lambda + a_5 \end{cases} \tag{3.1}$$

- **CPU Usage :**

$$y_{CPU} \begin{cases} a_1 = 85\times10^{-14}, a_2 = -25.26\times10^{-10}, \\ a_3 = 28.91\times10^{-7}, a_4 = -14.48\times10^{-4}, \\ a_5 = 34.69\times10^{-2}, a_6 = 8.38 \\ R^2 = 0.9964 \end{cases} \tag{3.2}$$

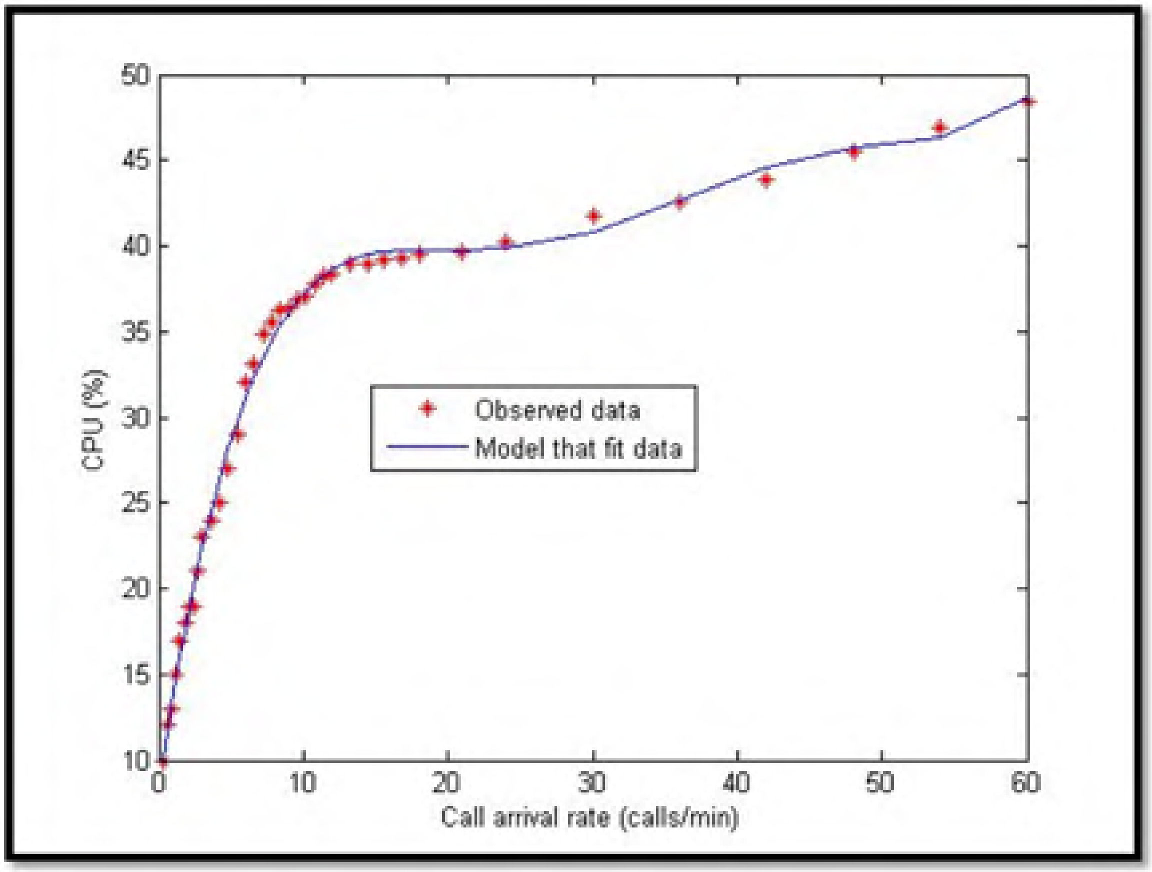


Figure 3.2 CPU Usage

- **Memory Usage :**

$$y_M \begin{cases} a_1 = 13.30\times10^{-14}, a_2 = -361.93\times10^{-12}, \\ a_3 = 3670.03\times10^{-10}, a_4 = -17367.23\times10^{-8}, \\ a_5 = 489.26\times10^{-4}, a_6 = 12.21 \\ R^2 = 0.9958 \end{cases} \tag{3.3}$$

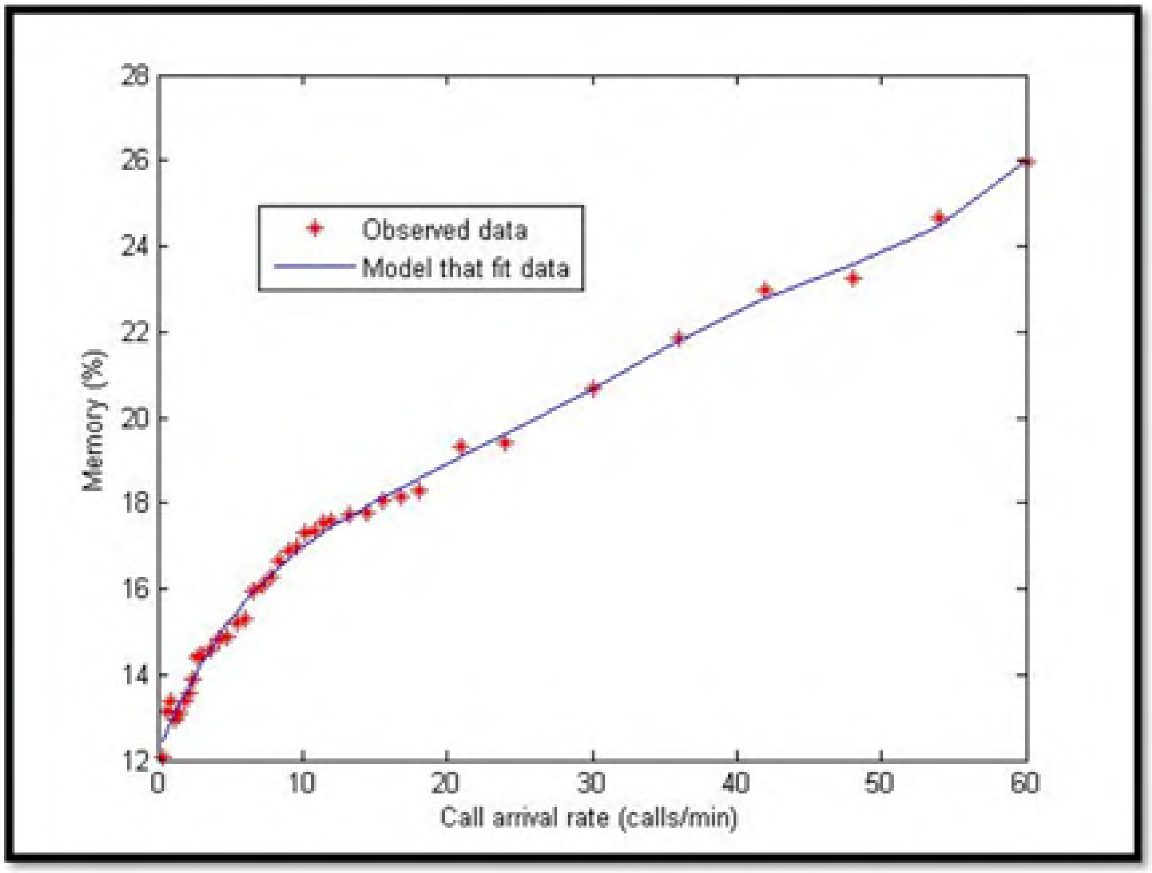


Figure 3.3 Memory Usage

- **BW Usage :**

$$y_B \begin{cases} a_1 = 16.92\times10^{-12}, a_2 = 73.49\times10^{-10}, \\ a_3 = -48.85\times10^{-6}, a_4 = 4.82\times10^{-2}, \\ a_5 = 11.75 \\ R^2 = 09843 \end{cases} \tag{3.4}$$

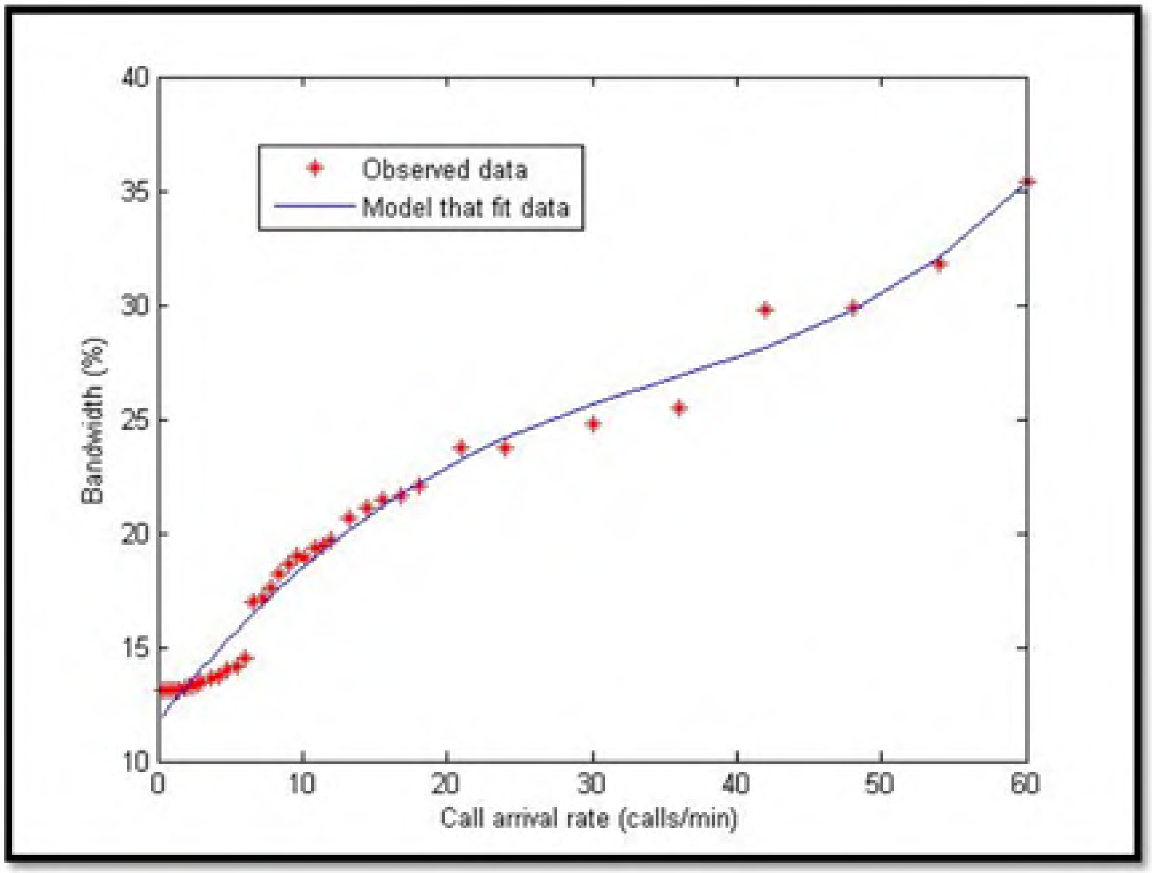


Figure 3.4 Bandwidth Usage

- **Disk space Usage :**

$$y_D \begin{cases} a_1 = -0.10\times10^{-14}, a_2 = 3.08\times10^{-12}, \\ a_3 = -19.10\times10^{-10}, a_4 = 8.14\times10^{-8}, \\ a_5 = 3.04\times10^{-4}, a_6 = 0.07 \\ R^2 = 0.9965 \end{cases} \tag{3.5}$$

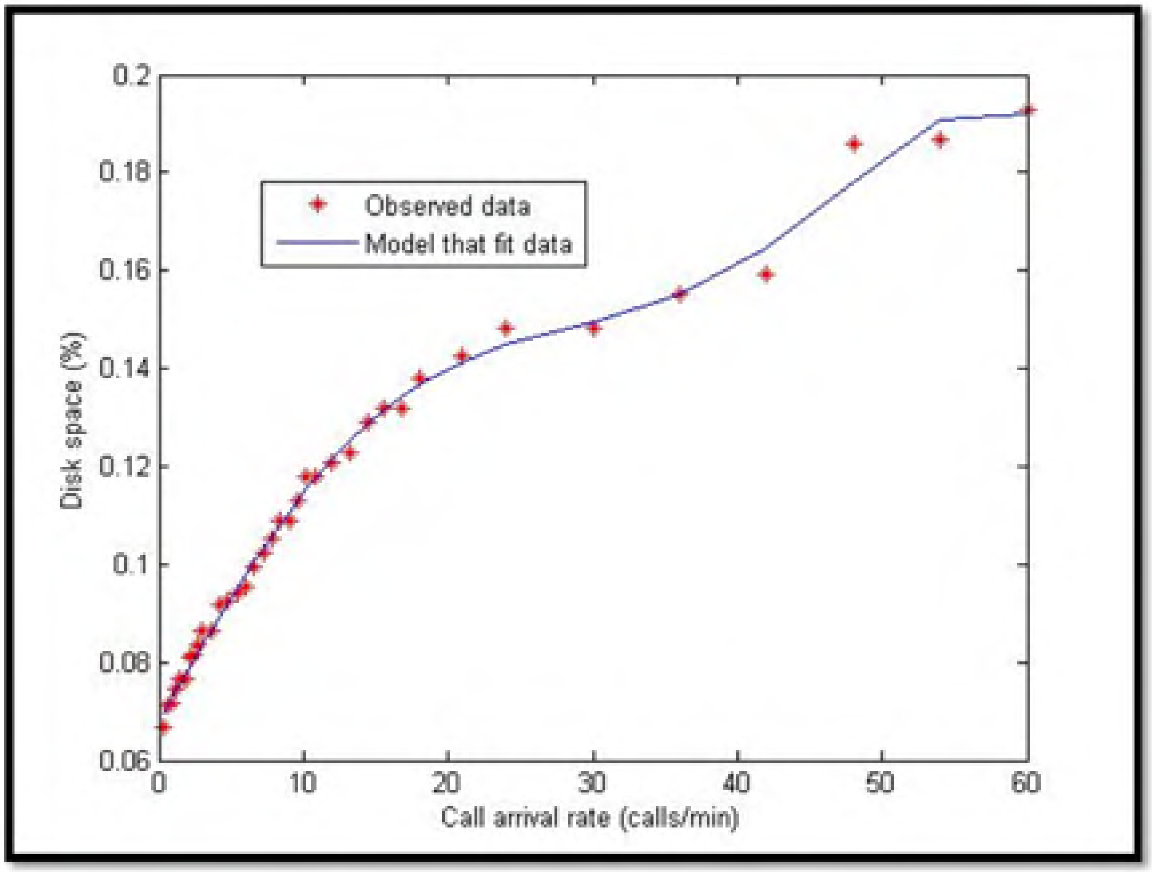


Figure 3.5 Disk Space Usage

3.3.2 Resources Computation

Resource computation can find the approximate values of the required resources for each SII (CPU_r, B_r, M_r, D_r). It is performed using a centralized GA-based method with the sliding window technique where two dimensional strings are used to represent the resource computation for each task in each processor. The resources computations for each task in each processor are presented using two dimensional strings: one string identifies the resource combination type provided by the virtualization machine and the second identifies the required resources for each task as shown in the Figure 3.6. In this figure, each resource in the second string is identified by the resource values C_i^j where i is the type of resource (CPU (1), memory (2), bandwidth (3) or disk space (4)) and j is the allocated resources to the instantiation request. The resource combination type *R1* for instance refers to an assignment of *CPU*=1 GHz, *B*=100 Mbps, *M*=256 MB*, D*=1 GB, as defined in Table 3.1.

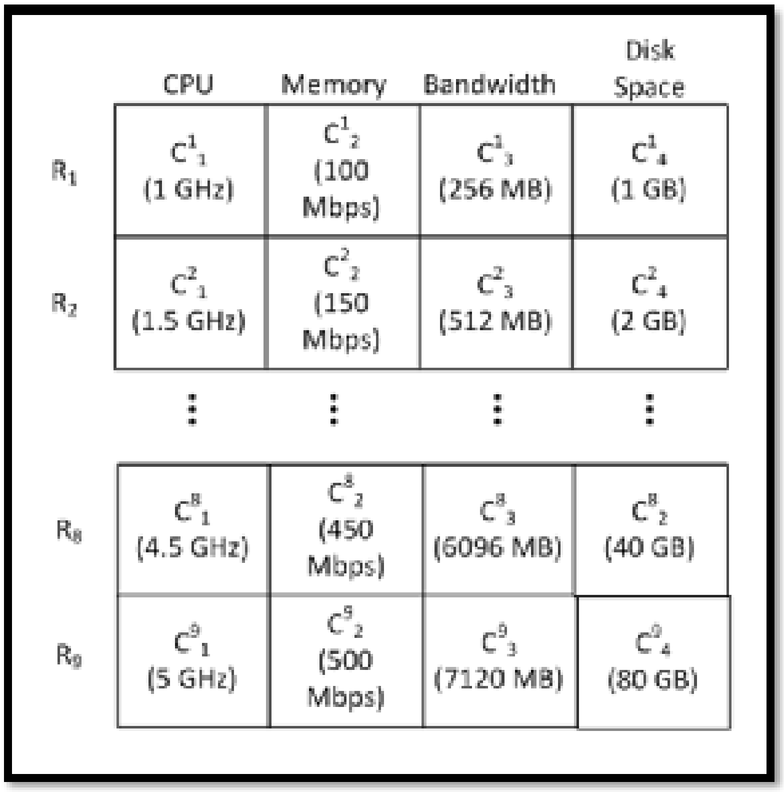


Figure 3.6 Resources Sharing

Table 3.1 Example of resource combination types

Type	Capacity			
	CPU (GHz)	RAM (MB)	BW (Mbps)	Disk Space (GB)
1	1	256	100	1
2	1.5	512	150	2
3	2	1024	200	4
4	2.5	2048	250	8
5	3	3072	300	10
6	3.5	4096	350	20
7	4	5072	400	30
8	4.5	6096	450	40
9	5	7120	500	80

The resources to be allocated to each instantiation request are identified using the resource computation selection, crossover and mutation methods described in the following sub-section.

3.3.3 Resources Computation Fitness Function

In this work, we assume that the service quality parameters required by each SII are described by the IVR application provider using a Service Level Agreement (SLA). In this report, we only consider the satisfactory factor of the IVR application as SLA parameter. This satisfactory factor can be defined as the resources used by a certain number of users over the allocated SII resources. This parameter can control application status so that no SII is under or over loaded and as result, it guarantees the pay as you use access. To evaluate this parameter for each SII, we define the resource computation fitness function so that our objective is to maximize the satisfactory factor of each SII (τ_{cpu}, τ_M , τ_B , τ_D) which are given by the following equations (3.6)

$$\begin{cases} \tau_{cpu} = \dfrac{y_{cpu}}{C_1^j}, & j = 1,\dots, m \\ \tau_M = \dfrac{y_M}{C_2^j}, & j = 1,\dots, m \\ \tau_B = \dfrac{y_B}{C_3^j}, & j = 1,\dots, m \\ \tau_D = \dfrac{y_D}{C_4^j}, & j = 1,\dots, m \end{cases} \tag{3.6}$$

These satisfactory factors are used as fitness function for the GA resources computing algorithm to evaluate if the resources are under or over loaded. The closer the satisfactory factor values are to 1, better is the resources are allocated. If the satisfactory factors are less than 0.75 or greater than 1 the resource are respectively under or over loaded This algorithm aims also to minimize the time t_r to compute the required resources (CPU_r, M_r, B_r, D_r) for each ISS. In fact, these results can limit the number of the possible combination to calculate the required resources for each instance. The resource computation selection, crossover and mutation method proposed in the following subsection satisfies this objective.

3.3.4 Resources Computation Crossover and Mutation

The resources computing algorithm is based on simple crossover and mutation operators to find the appropriate resources combination (*CPU$_r$, M$_r$, B$_r$, D$_r$*). Knowing the expected call arrival rate λ, for each SII, the required resources for each task are estimated using the proposed models in equations (3.2), (3.3), (3.4) and (3.5) . For instance, if the call arrival rate for a given SII is λ =30 calls/min, then the expected resource usage is given in (3.7):

$$\begin{cases} y_{CPU} = 49.89\,\% \times 1\,GHz = 0.4989\ GHz \\ y_M = 20.6659\ \% \times 512\ MB = 105.8094\ MB \\ y_B = 25.31\ \% \times 1\,Gbps = 0.2531\ Gbps \\ y_D = 0.1629\ \% \times 20\ GB = 0.0326\ GB \end{cases} \tag{3.7}$$

Furthermore, the expected resource usages for small IVR systems are given in the Table 3.2. We suppose that a small size IVR system has a call rate λ no greater than 60 calls/min.

Table 3.2 Required resources according to call arrival rate

λ calls/min	Required resources			
	y_{CPU} *(GHz)*	y_M *(MB)*	y_B *(Gbps)*	y_D *(GB)*
15	0.41	92.35	0.21	0.02
30	0.50	105.81	0.26	0.03
45	0.76	118.17	0.27	0.05
54	0.99	123.80	0.32	0.08
60	1.22	130.74	0.35	0.12

For call rates more than 60 calls/min, the algorithm uses the derived model to estimate the right values for each instance. Then, it calculates the satisfactory factors for the required resources and checks if they satisfy the criteria of the SLA. For instance, an instantiation request with λ= 60 calls/min requires 1.22 GHZ of CPU, 130.74 MB of memory, 0.35Gbps of bandwidth and 0.12 GB of disk space. For this request, the string with the closest values is selected (i.e., *R1* CPU, memory, bandwidth and disk space values). We compute the fitness values and we find that: τ_{cpu}=1.222, τ_M=0.51, τ_B=3.5 and τ_D=0.12. Nevertheless, these values should be between 0.75 and 1, so if these values are less than 0.75 the allocated resources should be reduced and if they are greater than 1, the resources should be increased. For that reason, we propose to use the sliding window to apply crossover and mutation across the possible solutions so, in the proposed example the value of the CPU of string *R1* (1 GHZ) is swapped with the CPU value of string *R2* (1.5 GHZ) while we maintain the same values of memory, bandwidth and the disk space as they are the smallest values that can satisfy the SLA conditions. The population derived from this mutation process will have a satisfactory factor τ_{CPU} = 0.81. This new population will be selected to represent the required resources for the received instantiation request that guarantee the best resource usage according to the resource combination types provided by the SII substrate.

3.4 Task Scheduling Algorithm

The substrate IVR engine receives a set of instantiations tasks to create new IVRs in the substrate layer. These requests should be assigned to processors to balance the load across them. Therefore, we propose to adapt the task scheduling algorithm proposed in (Y. A. Zomaya and Y. H. Teh, Sept.2001) and (Albert Y. Zomaya et al., 1999) in order to distribute the coming requests and minimize the required time to execute them. In addition, we propose to use the roulette wheel to initialize the first population in the centralized GA-based algorithm on which the GA will be applied. The roulette wheel is based on distributing randomly the number of the assigned tasks for each processor then, it specifies for each processor the selected tasks. At each time, the tasks that are within the roulette are reordered using the GA selection, crossover and mutation methods described in section 3.4.3. In the following sections we describe the representation of the tasks, the fitness function used to calculate the utility of the processors and lastly, we define the selection crossover and mutation used in the GA algorithm.

3.4.1 Processors and Task's Representation

We propose to represent the processors in the scheduling algorithm using two dimensional strings: one string identifies the processors and the other represents the assigned tasks in each processor. Each task is characterized by its size or its execution time $t_n = t_v + t_r$, preceded by the task number n (e.g. $2(t_v + t_r)$).

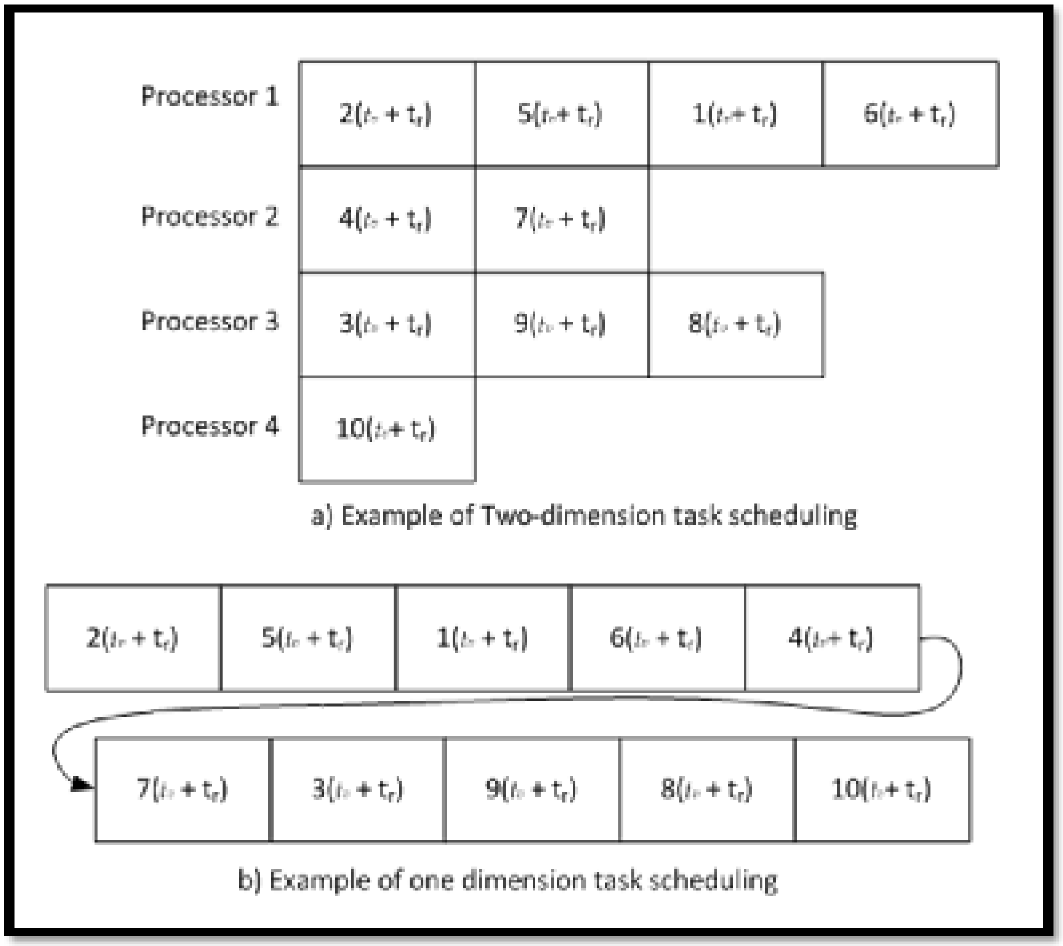


Figure 3.7 Task Scheduling

For the example presented in Figure 3.7, the processors' string will include the list of available processors. The GA starts by converting the strings from two dimensional to one string as shown in this figure. The t_v value is the same for all processors as we noticed in the experimental measurements that we did to implement our proposed prototype to create the IVR instance. However, the t_r has not a fixed value as it changes from a task to another, its value is computed using our proposed computational resources sharing algorithm. This is one of the differences between the original scheduling algorithm and the adapted one. In the original algorithm (proposed in (Y. A. Zomaya and Y. H. Teh, Sept.2001) and (Albert Y. Zomaya et al., 1999), the t_n value for each individual in a GA population is supposed to be known in advance.

3.4.2 Task Scheduling Fitness Function

We propose a fitness function to evaluate the distribution of tasks across processors that will handle their execution. This function enables the selection of tasks assignment that maximizes processors utilization and minimizes the execution time of the tasks. In (Y. A. Zomaya and Y. H. Teh, Sept.2001), the following objectives apply: 1) a minimization of the largest task completion time (i.e. Maxspan) across processors to guarantee that assignment tasks will be executed in shorter time possible (M. D. Kidwell and D. J. Cook, 1994); 2) increase the average processor utilization based on the Maxspan value; and 3) optimization of the number of tasks in each processor's queue, in order to ensure proper load balancing across the processors. We propose to combine the first and the second objectives by defining the TaksSpan as the difference between the largest task completion time and the smallest task completion time among all the processors in the system. In this algorithm, we define, the TaskSpan variable as the difference between the largest task completion time and the smallest task completion time among all the processors in the system rather than the MaxSpan value (as used in (Y. A. Zomaya and Y. H. Teh, Sept.2001) and (M. D. Kidwell and D. J. Cook, 1994), because guaranteeing minimum TaskSpan value not only ensures shorter task completion time but also a well-load balancing between all processors. The TaskSpan is calculated as follow in (3.8):

$$TaskSpan = \max_{i=1,\dots,m}\left(\sum_{j=1}^{n}(t_v + t_{rj})\right) - \min_{i=1,\dots,m}\left(\sum_{j=1}^{n}(t_v + t_{rj})\right) \tag{3.8}$$

Where n is the number of tasks in each processor queue, we can explain this approach to balance the tasks between the processor using an example given in Figure 3.8.

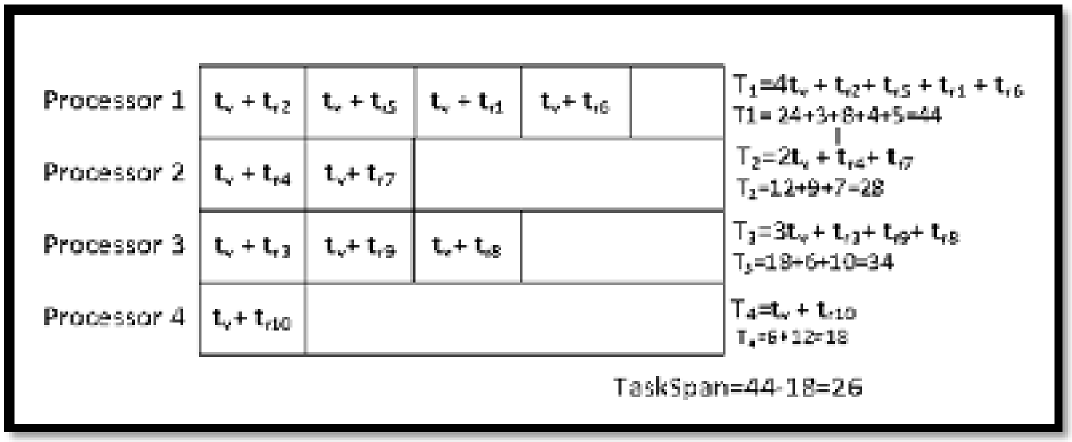


Figure 3.8 TaskSpan Calculation: Example

We assume in this example that t_v is equal to 6 units of time and the times t_r to compute the required resources for tasks 1 to 10 are respectively 4, 3, 8, 9, 8, 5, 7, 10, 6 and 12. Then, the processor 1 will execute tasks 2, 5, 1 and 6 within 44 units of time. Tasks 4 and 7 will be executed on processor 2 within 28 units of time, while tasks 3, 9, 8 and 10 will be executed within 34 and 18 units of time, respectively. Therefore, the TaskSpan for this task schedule example is 26 based on the proposed formula(3.9):

$$TaskSpan = \max_{i=1,...,4}(44,28,34,18) - \min_{i=1,...4}(44,28,34,18) = 26 \quad 3.9)$$

The second objective that we propose to define is the average processor utilization value which is the product of all processor utilization levels. In (Y. A. Zomaya and Y. H. Teh, Sept.2001), this value is the sum of all processor utilization levels by the total number of processors. Each processor utilization is calculated by dividing the tasks completion time in this processor by the MaxSpan value. The higher the average processor utilization, the better the load balancing across the processors. However, this objective doesn't guarantee that the load is well balanced across processor. Therefore, we propose to define the processor utilization factor *U* of all processors as the product of all processor utilizations. This factor is given by (3.10):

$$U = \prod_{i=1}^{m} \left(\frac{\sum_{j=1}^{n} (t_v + t_{rj})}{MaxSpan} \right) \tag{3.10}$$

Using the example given figure 3.8, this will lead to (3.11):

$$\begin{cases} P_1 : \frac{\sum_{j=1}^{n} (t_v + t_{rj})}{MaxSpan} = \frac{24+3+8+4+5}{44} = \frac{44}{44} = 1 \\ P_2 : \frac{\sum_{j=1}^{n} (t_v + t_{rj})}{MaxSpan} = \frac{12+9+7}{44} = \frac{28}{44} = 0.64 \\ P_3 : \frac{\sum_{j=1}^{n} (t_v + t_{rj})}{MaxSpan} = \frac{18+6+10}{44} = \frac{34}{44} = 0.77 \\ P_4 : \frac{\sum_{j=1}^{n} (t_v + t_{rj})}{MaxSpan} = \frac{6+12}{44} = \frac{18}{44} = 0.41 \end{cases} \tag{3.11}$$

Therefore, the *U* for this task schedule will be 0.2 and the average utilization value as defined in Y. A. Zomaya and Y. H. Teh, Sept.2001) will be 0.70. However, if we assign task 6 to processor 4 instead of to processor 1 in order to better balance the load in term of task completion time, this lead to (3.12):

$$\begin{cases} P_1 : \dfrac{\sum_{j=1}^{n}(t_v + t_{rj})}{MaxSpan} = \dfrac{18+3+8+4}{44} = \dfrac{33}{44} = 0.75 \\ P_2 : \dfrac{\sum_{j=1}^{n}(t_v + t_{rj})}{MaxSpan} = \dfrac{12+9+7}{44} = \dfrac{28}{44} = 0.64 \\ P_3 : \dfrac{\sum_{j=1}^{n}(t_v + t_{rj})}{MaxSpan} = \dfrac{18+6+10}{44} = \dfrac{34}{44} = 0.77 \\ P_4 : \dfrac{\sum_{j=1}^{n}(t_v + t_{rj})}{MaxSpan} = \dfrac{12+12+5}{44} = \dfrac{29}{44} = 0.66 \end{cases} \tag{3.12}$$

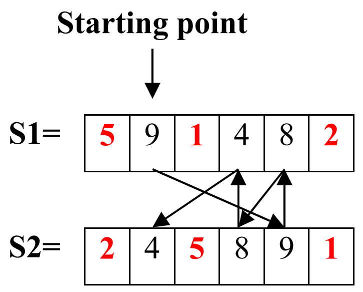
The utilization factor is then equal to 0.24, but the average utilization value as defined in (Y. A. Zomaya and Y. H. Teh, Sept.2001) will remain unchanged 0.70. Therefore, the greater the utilization factor, the better the load balancing. TaskSpan and utilization factor *U* are the two main objectives used by the fitness function of the GA task scheduling algorithm we propose. The higher is the fitness function *f*, better the task scheduling is. This function is defined as follow in (3.13):

$$f = \frac{U}{TaskSpan} \tag{3.13}$$

3.4.3 Task Scheduling Selection Crossover and Mutation

We propose to reuse the selection, the crossover and the mutation methods described in (Y. A. Zomaya and Y. H. Teh, Sept.2001). The Selection, crossover and mutation are based on the roulette wheel method (D. E. Goldberg, 1989). In this method, the selection of strings in a population is based on their fitness values. These values are used to assign to each string a probability of being selected.. These probabilities are computed by dividing the fitness of each string by the sum of the fitness values of the current set of strings in the population. The slots of the roulette wheel are created by adding the probability of the current string to the probability of the previous string. They are assigned until the value of 1 is reached. Then, the strings are selected randomly by generating a random numbers between 0 and 1. To perform the crossover operation, the selected strings are then converted from two dimensions to one. We use these two dimensions strings to balance the number of tasks across the processors. For instance, for 12 tasks and 8 processors, this procedure ensures that each processor will have at least one task and no more than 2 tasks. Hence, 8 tasks are allocated to 8 processors and 4 tasks are randomly assigned to 4 processors. This will allow the GA to converge for a fixed number of generation cycles (in our case 10 cycles).

The crossover operator is based on the cycle crossover method (Y. A. Zomaya and Y. H. Teh, Sept.2001) (Albert Y. Zomaya et al., 1999). First, crossover starts by selecting two random positions from two one dimension string S1 and S2. These positions should be between 1 and the length of the strings *S1* and *S2*. Let us assume that this starting point is $S_{1,\,n}$ which denotes the task at the position *n* in string 1. This task is marked as finished, and its corresponding task at $S_{2,\,n}$ is then also marked off as finished. The task in S_1 whose position is the value of $S_{2,\,n}$ is marked as finished and its corresponding task in S_2 is then marked off as finished as well. This process ends when the starting position $S_{1,\,n}$ is reached once again. Then the remaining tasks $S_{1,\,n}$ that were not marked off are swapped with their corresponding tasks in S_2 (ex. $S_{1,4}$ is swapped with $S_{2,4}$). When all tasks in the two strings are crossed over, they are reordered and converted to a two dimensional form to compute their new fitness values. An example of crossover is given in the following figure with the starting point is 2:

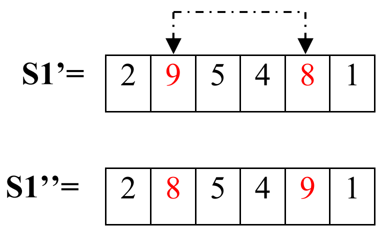


In this example, the tasks that are not marked have the positions: 1, 3 and 6. These tasks should be swapped between S_1 and S_2. This leads to the given results S_1' and S_2':

S1'=	**2**	9	**5**	4	8	**1**
S2'=	**5**	4	**1**	8	9	**2**

Then, the mutation operator is applied on the string S$_1$' by selecting two random positions n_1 and n_2 to swap tasks on these positions as shown in the following example:

Swap Mutation



Based on the new allocation of tasks given in S$_1$'', new fitness values are then computed to check if it's the optimal distribution across the processors in order to improve their performance and minimize the required execution time.

Third GA operator is based on swap mutation. It randomly selects two tasks and then swaps them. Each task is taken on randomly selected processors. These processors should be different to ensure that the two selected tasks are not the same. New fitness values are computed using the population derived from this swapping mutation process.

3.5 Conclusion

In this chapter, we discussed the proposed algorithms to compute the required resources for the coming request and to distribute the IVR creation tasks across the processors which will execute them using GA-based algorithms. The first algorithm concerns computational resource sharing, whereas the second relates to the scheduling of the IVR application instantiation requests. Both algorithms are GA-based and they both consider several objectives to optimize the resource usage and sharing at the substrate layer. The scheduling algorithm maximizes resources utilization while minimizing the execution time of tasks. The computational resource sharing algorithm minimizes the substrate resource utilization and the resource allocation time while maximizing the satisfactory factor of the IVR applications. Before discussing the performance measurements of these two algorithms, we propose to describe the developed prototype in the following chapter

CHAPTER 4

PROPOSED PROTOTYPE

4.1 Introduction

In this chapter, we propose to describe the prototype used to implement a virtualized architecture for IVR applications. We will describe also the implemented scenario to the proposed architecture and algorithms.

4.2 Architectural Prototype Design

To implement the proposed architecture, we use a proof of concept that includes sub-sets of the service, the composition and the management planes. We present some of the assumptions, and the architecture of the prototype and the software tools used to implement it.

4.2.1 General Assumptions

We assume that the infrastructure layer is composed of five substrates that are announcement player, voice recorder, key detector, extension detector and call transfer. We further assume that the substrates are supplied by substrates providers SubP1, SubP2, SubP3, SubP4 and SubP5 respectively. We further assume that we have one infrastructure provider InfP, one platform provider PP, and two service providers ServP1 and ServP2. The substrates are described using Donkey State Machine DSM that represents the substrate behavior as a state machine. The use of DSM is motivated by the fact that it is used by the SIP Express Media Server (SEMS) used in our prototyping environment, and because it eases the composition of the substrates. DSM enables a textual description of applications that can be directly executed by interpreters hosted by the SEMS.

4.2.2 Prototype Architecture

Figure 4.1 gives a general overview about the architecture of the proposed prototype. Our architecture is based on RESTFul web services. The service provider (ServP) can discover all the available IVR substrates using the GET method. The service provider can compose its new service by selecting some or all the discovered services displayed on the platform provider (PP) GUI. Then, the service will be activated and the required resources for this new substrate will be reserved. Using the PUT method the new service will be published into the broker. As a result, the ServP can be able to provide the service to the clients and other application to execute it.

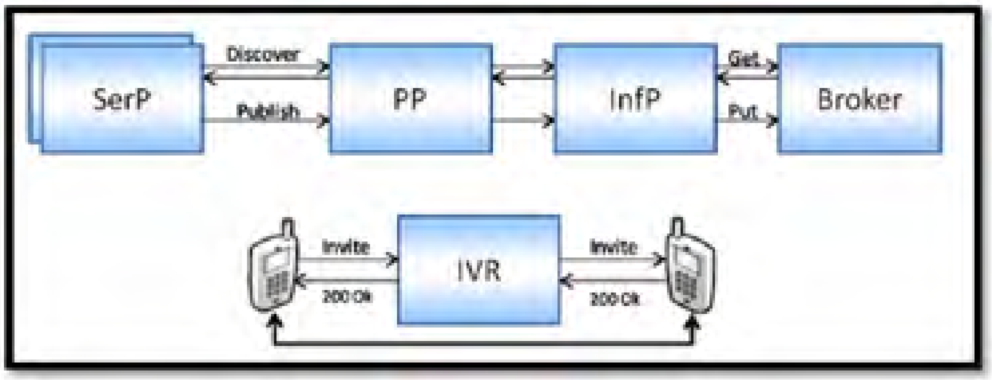


Figure 4.1 General Prototype architecture

Based on the general assumptions we described above, we will present a case study which covers the application development steps including mainly the publication, and the discovery phases. For the publication and the discovery phases, Figure 4.2 depicts the interactions between the InfP, SubPs and the broker. At first, the five substrates are published into the broker so that PP and InfP can discover them later. The exchanged requests in this step are Restful (PUT and GET requests). Each substrate provider publishes its services to the broker using a PUT request with an attached WADL description of the substrate. Next, the platform provider sends a GET request to discover the list of published services. The obtained list is then available to the platform provider through the GUI.

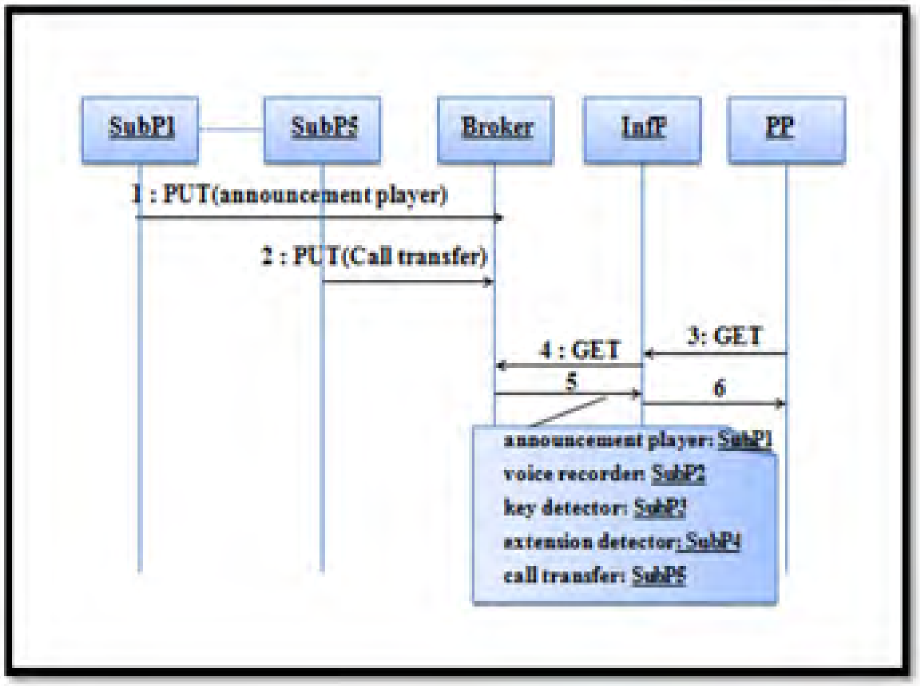


Figure 4.2 Substrate Discovery and Publication

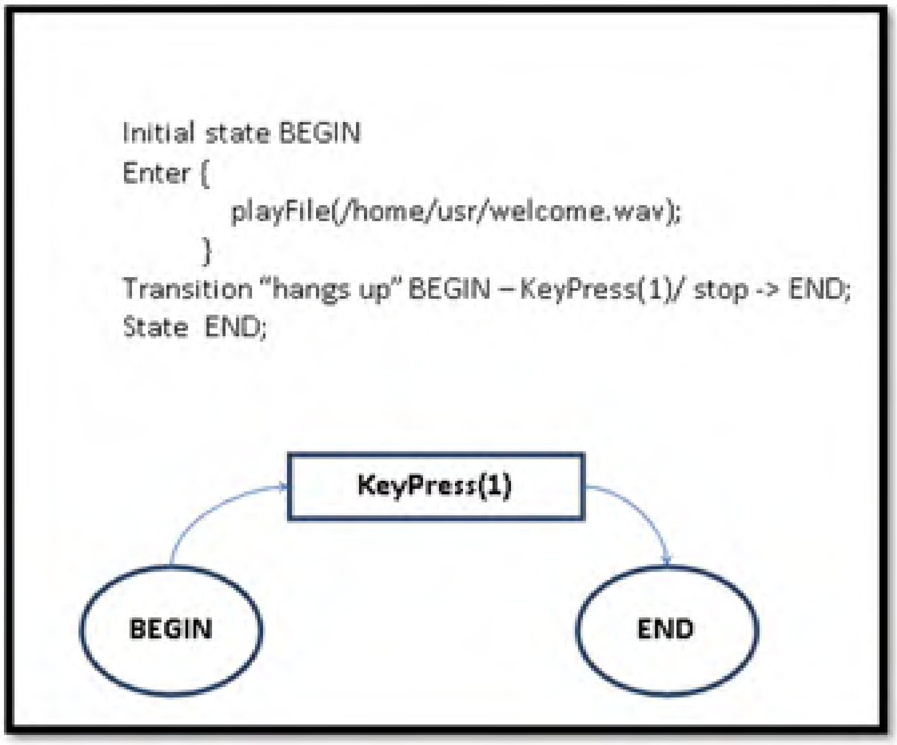
4.2.3 Architectural Environmental Settings

To implement the proposed prototype, we used several software tools which were used to create, manage IVR substrates and execute IVR services. This section is dedicated to describe the architectural environment settings used in the test scenarios that will be discussed in the next section.

4.2.3.1 SEMS

The Donkey State Machine (DSM) module is used to implement IVR substrates which are deployed using the Sip Express Media Server (SEMS). SEMS is a free, open source and scalable server for VoIP services. It handles the media processing for the applications using IVR services. It supports the basic call and audio functions and many types of plug-ins may extend its system. It offers also conferencing and voicemail services. Developers can extend it by creating their own plug-ins based on SEMS framework API in C++, python or the DSM.

DSM is a powerful service scripting development platform. Actually, it offers a textual description of applications which will be executed by the DSM module as application in SEMS. Service logic is considered as state machine; it contains states and transitions between the states. There are conditions between the states; they are checked when an event occurs and if the conditions matches then the transitions are made. The transitions are associated with a set of actions which are executed on entering the state and on leaving the state as illustrated in the Figure 4.3. An example of DSM file is given in this figure with two states BEGIN and END. The first state allows to play a file and then when 1 is pressed by a caller, an event is occurred, the transition is made and the action will be executed which is Stop in this case.



```
Initial state BEGIN
Enter {
        playFile(/home/usr/welcome.wav);
      }
Transition "hangs up" BEGIN – KeyPress(1)/ stop -> END;
State  END;
```

Figure 4.3 DSM Example

4.2.3.2 Interfaces and Repository

Interfaces in the implemented prototype are deployed using Jersey which is an open source reference implementation of JSR 311 to build RESTFul web services. The interfaces are deployed on a Glassfish Server and have a socket communication with the SEMS server. Glassfish server is an open source application server to deploy RESTFul web services. The substrate broker or the repository is based on Jersey and Glassfish Servers.

4.2.3.3 Java

In the service plane, we choose Java to implement the graphical user interface for IVR providers, we select Helios version Eclipse IDE (Integrated Development Environment) for java. Eclipse is an open source IDE that offers featured functions to build java applications and deploy them in Glassfish Server. It supports also Jersey APIs that we use to build the RESTFul web services.

4.3 Management Prototype Design

4.3.1 Prototype Architecture

The management phase is described in Figure 4.4 which shows the activation and the creation steps of a new substrate called Automated Attend which is composed from the five substrates mentioned in sub-section 4.2.1. The activation request is sent before the creation request which launches the resources management algorithms in the IVR resource management engine. First, it executes the resource computation algorithm to get the values of the required CPU, RAM, BW and disk space for the new coming request, then it executes the scheduling algorithm to balance the load across processors to optimize their performances and minimize the execution time. As a result, the management engine sends a configuration file that has “conf” as an extension to the IVR instance manager. This file contains the characteristics of the virtual machines that will host the new IVR substrate.

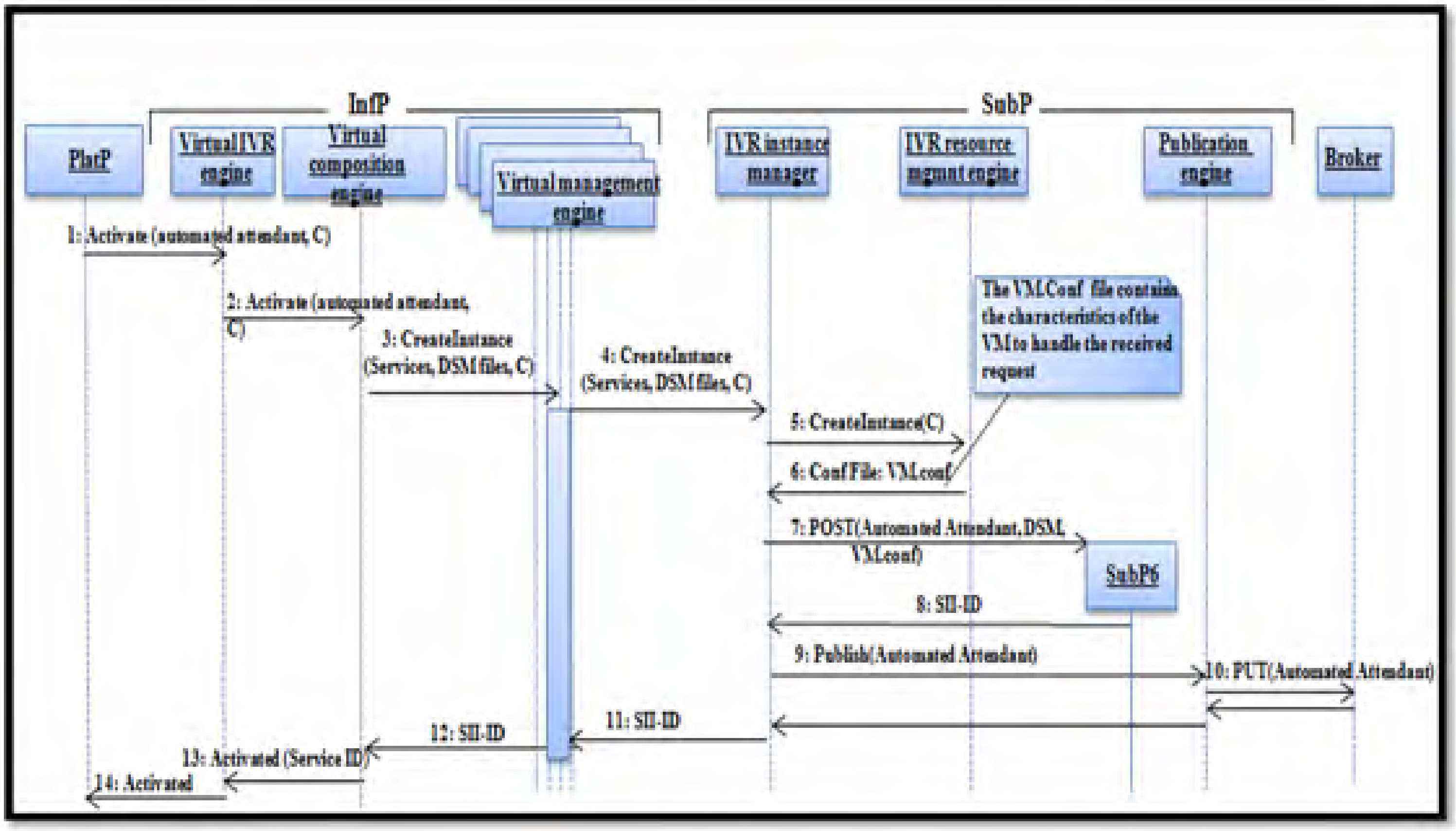

InfP
SubP
PlatP
Virtual IVR engine
Virtual composition engine
Virtual management engine
IVR instance manager
IVR resource mgmnt engine
Publication engine
Broker
1: Activate (automated attendant, C)
2: Activate (automated attendant, C)
3: CreateInstance (Services, DSM files, C)
4: CreateInstance (Services, DSM files, C)
The VM.Conf file contains the characteristics of the VM to handle the received request
5: CreateInstance(C)
6: Conf File: VM.conf
7: POST(Automated Attendant, DSM, VM.conf)
SubP6
8: SII-ID
9: Publish(Automated Attendant)
10: PUT(Automated Attendant)
11: SII-ID
12: SII-ID
13: Activated (Service ID)
14: Activated

Figure 4.4 Substrate Activation and Management

Next, the IVR instance manager sends the creation request attached with the configuration file as an argument to the hypervisor in order to instantiate the VM. In our prototype we are using XEN server to host the VMs. Figure 4.5 shows an example of VM.conf file based on python language. Finally, the IVR resource manager sends a new request to the Publication engine to make the new service visible for others providers by sending a RESTFul Post request to the broker. This request contains the description of the new created substrate.

```
#  -*- mode: python; -*-
# Python configuration setup for 'xm create'.
# This script sets the parameters used when a domain is created using
'xm create'.
# You use a separate script for each domain you want to create, or
# you can set the parameters for the domain on the xm command line.
# Kernel image file.
kernel = "/boot/vmlinuz-2.6.32.40"
ramdisk="/boot/initrd.img-2.6.32.40"
#extra = "rhgb console=ttyS0 "
#extra = "text ks=http://localserver/minimal-ks.cfg"
memory = 1024
name = "VM12-IVR2"
vcpus = 1
vif                                                                    =
['vifname=IVR2,ip=10.194.32.201,mac=00:16:3e:02:03:05,bridge=br0-
xen']
netmask = "255.255.255.0"
gateway = "10.194.32.1"
hostname = "xen-ivr-2.logti.etsmtl.ca"
# Set root device.
root = "/dev/xvdb1 ro"
disk = ['tap:aio:/virtual_machines/Xen/IVR-2/ivr-2.img,xvdb1,w']
#uuid = "162910c8-2a0c-0333-2349-049e8e32ba90"
#bootloader = "/usr/bin/pygrub"
# default rules on reboot
on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'restart'
# interface console
# serial = 'pty'
```

Figure 4.5 Example of Configuration File

4.3.2 Algorithmic Environmental Settings

4.3.2.1 Virtualization Server

XEN server was chosen to set the virtualization environment. It is a powerful open source standard which is characterized by its efficiency and offers secure feature set in such environments. In addition, it is a cloud-proven virtualization platform that consists of all the required capacity to create and manage virtual infrastructure. Furthermore, it supports multiple guest operating systems including Windows, Linux, Solaris, etc. It has a software virtualization layer called XEN Hypervisor as illustrated in Figure 4.6. This hypervisor is responsible for managing the virtual machines known as the server domains and resources assignment between different domains. The Domain0 is the privileged domain that has direct access to the hardware resources. The hypervisor has to optimize the resources utilization and scheduling the processes running on it.

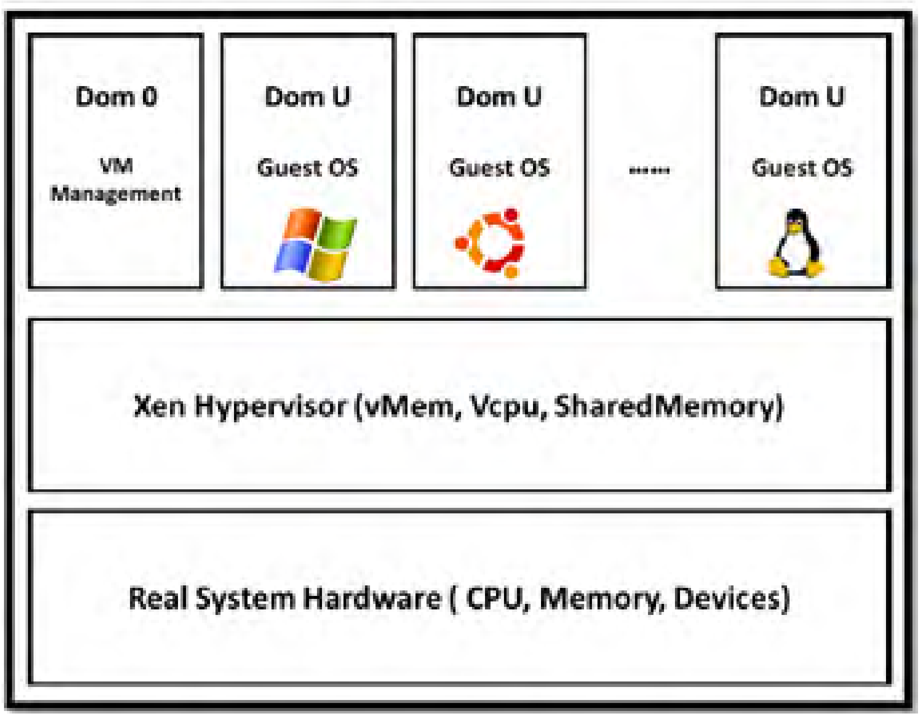

Dom 0
VM Management
Dom U
Guest OS
Dom U
Guest OS
......
Dom U
Guest OS
Xen Hypervisor (vMem, Vcpu, SharedMemory)
Real System Hardware (CPU, Memory, Devices)

Figure 4.6 XEN Architecture

4.3.2.2 Quick Test Professional

HP Quick Test Professional (QTP) is powerful software used to automate and test applications and networking environments. It includes scripting features that detect the occurred events in the graphical user interface and save them into a Visual Basic script to test it later. Furthermore, it can manipulate and control the same objects specified for the test while executing the script. It uses record and playback methods to test the specified script. We use QTP in our prototype to get some measurements about the use of CPU, memory, BW and disk space by recording a testing scenario and playing it under different testing conditions to get the required results.

4.3.2.3 Matlab

For the management plane, we choose Matlab which is a programming environment to develop and implement the proposed algorithms and analyze the computation results. Matlab is powerful tool for computations problem. It can be used in several domains such as signal and image processing, communications, financial modeling and analysis, etc. We use it in the management prototype to test and get the results of the two proposed algorithms.

4.4 Testing Scenarios

4.4.1 Architectural Testing Scenario

We implemented the scenario where the first service provider creates and provisions an automated attendant service for a bank. In this section, we will describe the steps to implement and develop this service. In the proposed example, when a client is calling a bank or a specific enterprise, the IVR application will play a record file in which it includes the welcome message using “announcement player” and asks the client to enter the required extension using “extension detector”. Then, the application will transfer the call to the specified extension using “call transfer” and if the entered extension is invalid the IVR service plays another record file to ask the client to enter the correct one. If the extension is valid and the callee doesn’t reply the application record the caller voice to play it later using “voice recorder”. Figure 4.7 presents the GUI offered by the platform provider to the service provider. Service providers will use this GUI to develop and manage IVR applications.

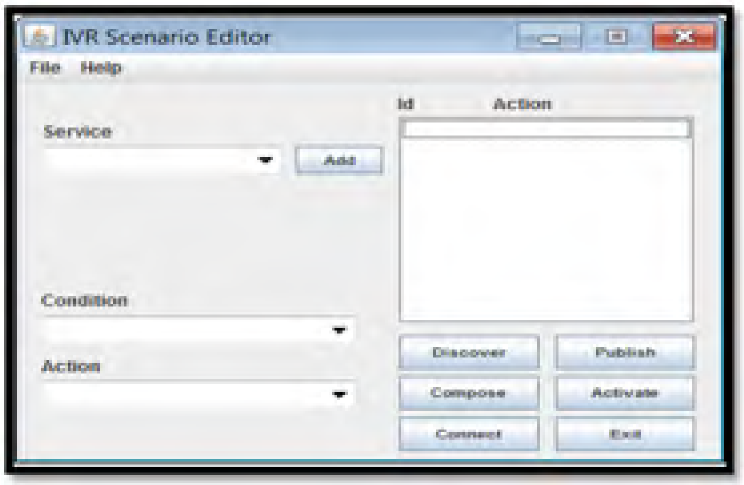


Figure 4.7 Platform Provider GUI

First of all, the service provider pushes the “Connect button” in order to establish a connection with the platform provider. If the platform provider is connected to the server, it means that he is able to develop and manage the IVR application and build new ones. Using the “Discover button”, he can get the list of the published services in the broker and the URI of the existing substrates. Next, the service provider can select and compose new services from the existing substrates as shown in Figure 4.8. In addition, he can add a condition and an action, this information is important to specify the relationship between the composed services in the execution phase.

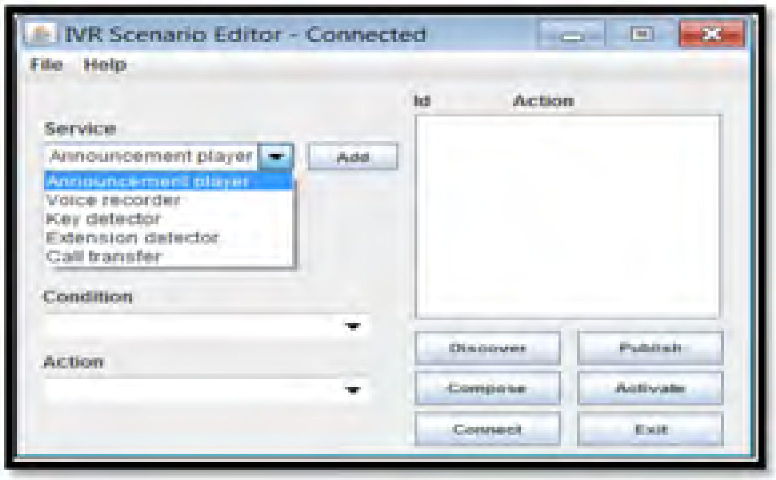
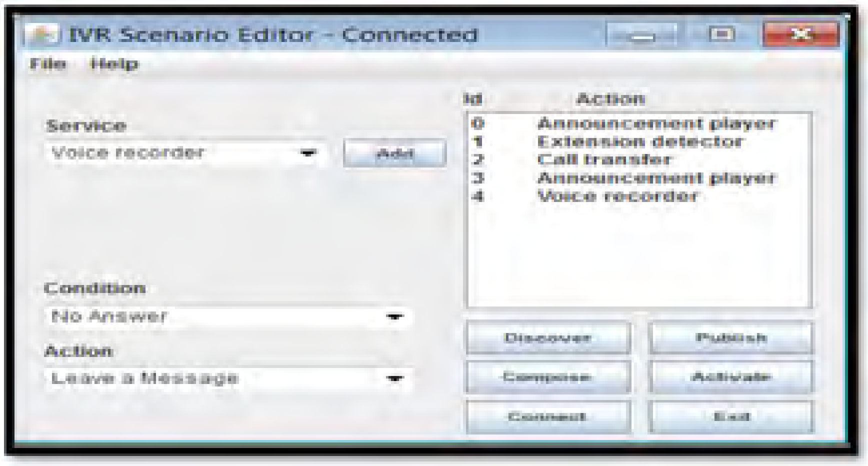


Figure 4.8 Platform provider discover

The GUI allows the service provider to create its composed service by choosing the substrates to compose and then ordering them using this graphical tool. The substrates selected in Figure 4.9 allow implementing an automated attendant service.



Service: Voice recorder

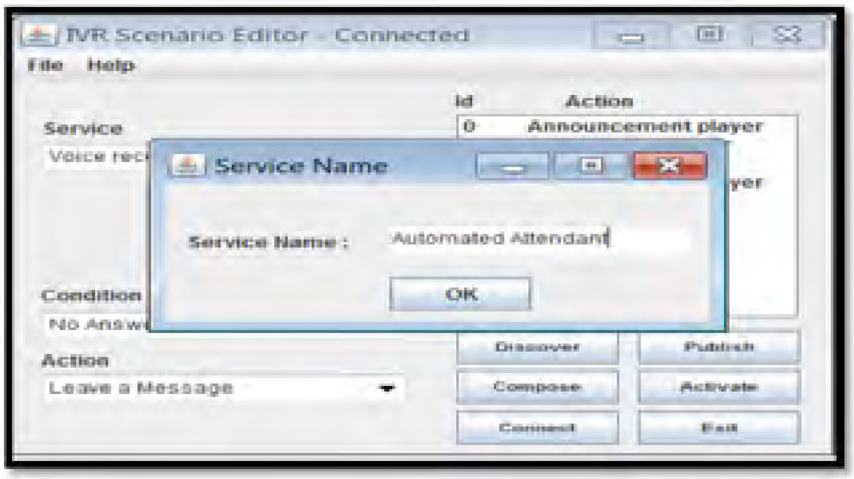
Id	Action
0	Announcement player
1	Extension detector
2	Call transfer
3	Announcement player
4	Voice recorder

Condition: No Answer

Action: Leave a Message

Figure 4.9 Platform provider compose

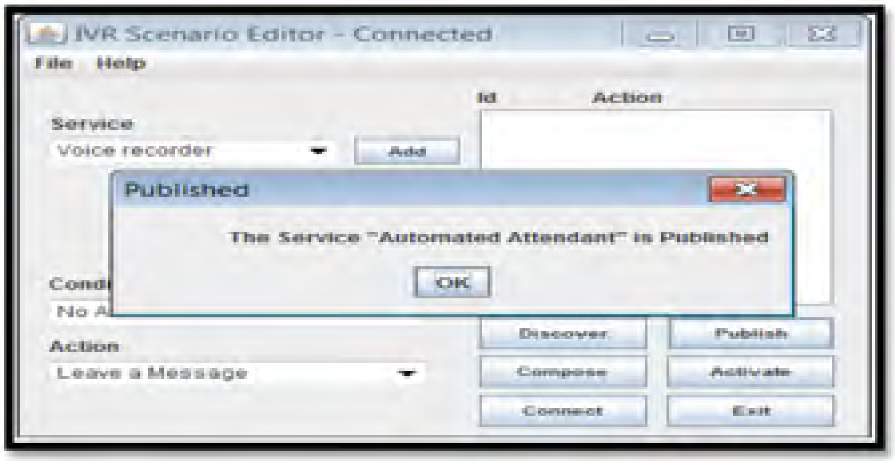
The proposed scenario for the test is composed of the 5 substrates listed in Figure 4.9. When the “Compose button” is pushed, the service provider should give a name to the new composed service. Figure 4.10 shows the creation of the automated attendant service.



Service Name : Automated Attendant

Figure 4.10 Platform provider create

Then, the new composed service is activated (Figure 4.11) after computing resources are calculated using the algorithms and generating the configuration file of the new VM. Lastly, the substrate instance will be created and the new composed service is published into the broker.



Published

The Service "Automated Attendant" is Published

OK

Figure 4.11 Platform provider activate and publish

Later, if a new service provider discovers the published services, he will find that the automated attendant service belongs to the existing list in the broker as shown in Figure 4.12.

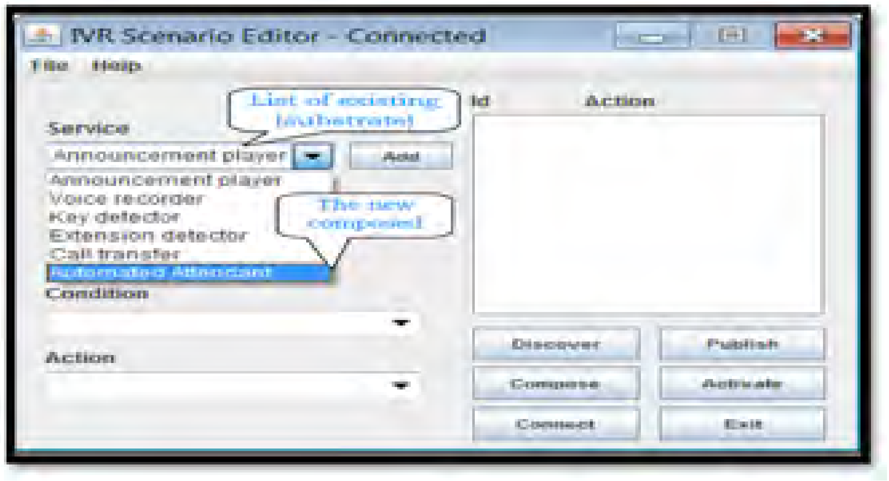


Figure 4.12 Platform Provider re-discover

4.4.2 Management of the testing Scenario

In this subsection, we will describe the two proposed algorithms and their implemented functions. First, we will discuss the computational resources algorithm which allows selecting resources for each SII while minimizing the amount of the required resources, minimizing the resource allocation time and maximizing the satisfactory factor of each new IVR instance. The following functions describe in details the first algorithms including the *main()* function and the appropriate functions.

Main (): Computational Resources Calculation Program for IVR applications using the Genetic Algorithm (GA) and the Sliding Window technique:

- **Inputs:**
  - **n:** Number of virtual machines to create
  - **Lambda []:Call arrival rate**
- **Outputs:**
  - **CPU[] :** Required CPU for each new VM using the CPU equation
  - **RAM[] :** Required RAM for each new VM using the RAM equation
  - **BW[] :** Required BW for each new VM using the BW equation
  - **Disk[] :** Required Disk for each new VM using the Disk equation

```
1. Print "Enter the number of tasks:"
2. Read n // n= number of tasks sent from the instance engine and received by the
   Resource manager
3. Nbmaxcpu = nbmaxram= nbmaxbw= nbmaxdisk = maximum of the resources that can
   support a VM in our proposed model
4. for index =1 to n do
       Print "Enter the call arrival rate for each IVR
       Read Lambda(index)
   endfor
   //Calculate the required resources using the equations given by the load measurement
   functions and the Sliding Window technique
5. Function 1(): Compute the required resources using the equations given by the load
   measurement functions
6. Function2(): Compute the Initial Satisfactory Factors of the resources using Sliding
   Window technique
7. Function3(): Find the Resource type based on the Resource Type table and the Sliding
   Window technique
```

Function 1(): Compute the required resources using the equations given by the load measurement functions

- **Inputs:**
  - **n:** Number of virtual machines to create
  - **Lambda[]:Call arrival rate**
- **Outputs:**
  - **CPU[]** : Required CPU for each new VM using the CPU equation
  - **RAM[]** : Required RAM for each new VM using the RAM equation
  - **BW[]** : Required BW for each new VM using the BW equation
  - **Disk[]** : Required Disk for each new VM using the Disk equation

```
1. for i = 1 to n do

  a. CPU Equation

  if(Lambda(i)>nbmaxcpu) then

      CPU(index)=  $(85*10^{-14} * Lambda(i)^5 - 25.26 * 10^{-10}*Lambda(i)^4 + 28.91 * 10^{-7} * Lambda(i)^3 - 14.48 * 10^{-4} * Lambda(i)^2 + 34.69 * 10^{-2} * Lambda(i) + 8.38) * ( Lambda(i)/nbmaxcpu)$

  else

      CPU(index)=  $85*10^{-14} * Lambda(i)^5 - 25.26 * 10^{-10}*Lambda(i)^4 + 28.91 * 10^{-7} *Lambda(i)^3 - 14.48 * 10^{-4} * Lambda(i)^2 + 34.69 * 10^{-2} * Lambda(i) + 8.38$

  endif

  b. RAM Equation

    if( Lambda(i)>nbmaxram) then

     RAM(index)=  $(13.30*10^{-14}* Lambda(i)^5-361.93*10^{-12}*Lambda(i)^4+ 3670.03*10^{-10}*Lambda(i)^3-17367.23*10^{-8}*Lambda(i)^2+489.26*10^{-4} *Lambda(i) + 12.21) * ( Lambda(i)/nbmaxram)$

    else

     RAM(index)=  $(13.30*10^{-14}*Lambda(i)^5-361.93*10^{-12}*Lambda(i)^4 + 3670.03*10^{-10}*Lambda(i)^3-17367.23*10^{-8}*Lambda(i)^2+489.26*10^{-4} *Lambda(i) + 12.21)$

    endif
```

```
c. BW Equation

  if( Lambda(i)>nbmaxbw) then

    BW(index)= $(16.92*10^{-12}*Lambda(i)^5-73.49*10^{-10}*Lambda(i)^4-48.85*10^{-6}*Lambda(i)^3+4.82*10^{-2}*Lambda(i)^2+11.75*Lambda(i))*( Lambda(i)/nbmaxbw)$

  else

    BW(index)= $(16.92*10^{-12}*Lambda(i)^5 - 73.49*10^{-10}*Lambda(i)^4 - 48.85*10^{-6}*Lambda(i)^3 + 4.82*10^{-2}*Lambda(i)^2 + 11.75*Lambda(i))$

  endif

d. DiskSpace Equation

  if( Lambda(i)>nbmaxdisk) then

    Disk(index)= $(-0.10*10^{-14}*Lambda(i)^5+3.08*10^{-12}*Lambda(i)^4-19.10*10^{-10}*Lambda(i)^3+8.14*10^{-8}*Lambda(i)^2+3.04*10^{-4}*Lambda(i)+ 0.07)*( Lambda(i)/nbmaxdisk)$

  else
    Disk(index)= $(-0.10*10^{-14}*Lambda(i)^5+3.08*10^{-12}*Lambda(i)^4-19.10*10^{-10}*Lambda(i)^3+8.14*10^{-8}*Lambda(i)^2+3.04*10^{-4}*Lambda(i)+ 0.07)$

  endif

endfor
```

Function2(): Calculate the Initial Satisfactory Factors of the resources using Sliding Window technique

- **Inputs:**
  - **n:** Number of virtual machines to create
  - **CPU[]** **:** Required CPU for each new VM using the CPU equation
  - **RAM[] :** Required RAM for each new VM using the RAM equation
  - **BW[]** **:** Required BW for each new VM using the BW equation
  - **Disk[]** **:** Required Disk for each new VM using the Disk equation

- **Outputs:**
  - **TempM [][]:** matrix that contains the initial values of the required resources
  - **TempMSatis[][]:** matrix that contains the initial values of the Satisfactory factors of each resource CPU[], RAM[], BW[] and Disk[] using the formula: (Eq. 3.6)

```
1.  For i=  0 to (n-1) do
        //Calculate the initial values of the required resources
        TempM(2*i,1)=floor(CPU(i));
        TempM(2*i+1,1)=floor(CPU(i))+1
        TempM(2*i,2)=floor(RAM(i))
        TempM(2*i+1,2)=floor(RAM(i))+1
        TempM(2*i,3)=floor(BW(i))
        TempM(2*i+1,3)=floor(BW(i))+1
        TempM(2*i,4)=floor(Disk(i))
        TempM(2*i+1,4)=floor(Disk(i))+1

        //Calculate the Satisfactory factor of the initial values

        TempMSatis(2*i,1) =  floor(CPU(i)) / CPU(i)
        TempMSatis(2*i+1,1) = (floor(CPU(i))+1) / CPU(i)
        TempMSatis(2*i,2) =  floor(RAM(i)) / RAM(i)
        TempMSatis(2*i+1,2) = (floor(RAM(i))+1) / RAM(i)
        TempMSatis(2*i,3) =  floor(BW(i)) / BW(i)
        TempMSatis(2*i+1,3) = (floor(BW(i))+1) / BW(i)
        TempMSatis(2*i,4) =  floor(Disk(i)) / Disk(i)
        TempMSatis(2*i+1,4) = (floor(Disk(i))+1) / Disk(i)

    endfor
```

Function3(): Find the Resource type based on the Resource Type table and the Sliding Window technique

- **Inputs:**
  - **n:** Number of virtual machines to create
  - **CPU[] :** Required CPU for each new VM using the CPU equation
  - **RAM[] :** Required RAM for each new VM using the RAM equation
  - **BW[] :** Required BW for each new VM using the BW equation
  - **Disk[] :** Required Disk for each new VM using the Disk equation

- **Outputs:**
  - **TMax [][] :** Optimum value of the CPU, RAM, BW and Disk of each VM

- **Satisf [][] :** Optimum satisfactory factor of CPU, RAM, BW and Disk of each VM
- **Timearray[]:**Required time to estimate the minimized resources for each request.

```
1. TMax=[n,4] //Tmax contains the minimum values of CPU,RAM, BW and Disk that
   //ensure satisfacory value between 0.7 and 1 according to the SLA on the cloud.
2. Satisf=[n,4]  //Satisf contains the values of the new Satisfactory values of each //
   resource
3. for i =0 to (n-1) do
   inttime1=cputime   // initial time to start the computation of each VM
   k=i+1
      a) CPU satisfactory
      Function 3.1()
      b) RAM satisfactory
      Function 3.2()
      c) BW satisfactory
      Function 3.3()
      d) Disk satisfactory
      Function 3.4()
       inttime2=cputime //last time to end the computation of each VM
       timearray(k)=inttime2-inttime1
      //Calculate the required time to find the required resources for each VM
      end for
```

Function3.1(): Find the Resource type based on the Resource Type table and the Sliding Window technique

- **Inputs:**
  - **n:** Number of virtual machines to create
  - **k:** Index of the IVR creation request
  - **CPU[] :** Required CPU for each new VM using the CPU equation
  - **TempMSatis [][]:**initial values of the Satisfactory factors of each resources
  - **TempM [][]:**initial values of the resources
  - **Cputime :** starting execution time given by the system
- **Outputs:**
  - **TMax(k,1) :** Optimum value of the CPU to kth VM
  - **Satisf(k,1) :** Optimum satisfactory factor of the CPU of the kth VM
  - **Timearray[]:**Required time estimate the minimized resources for each request.

```
a) CPU satisfactory

1.   satiscpu=TempMSatis(2*k,1)
2.   xcpu=TempM(2*k,1)
3.   t1=cputime //t1 is used to optimize the execution time
4.   while((satiscpu>1)||(satiscpu<0.75) || ((cputime-t1)>0.02))do
       //Check if the satisfacory of the cpu is between 0.75 and 1
       if((cputime-t1)<0.02) then
       // check if the time to calculate the required cpu is minimized
         if((satiscpu>1)) then
           xcpu=xcpu-1
         endif
         if((satiscpu<0.75))then
           xcpu=xcpu+1
         endif
         satiscpu=xcpu/CPU(k) //calculate the new value of the CPU satsifactory
       endif
     endwhile
5.   TMax(k,1)=xcpu // the optimum value of the CPU
6.   Satisf(k,1)=satiscpu // the optimum value of the satisfactory factor of CPU
```

Function3.2(): Find the Resource type based on the Resource Type table and the Sliding Window technique

- **Inputs:**
  - **n:** Number of virtual machines to create
  - **k:** Index of the IVR creation request
  - **RAM[] :** Required RAM for each new VM using the RAM equation
  - **TempMSatis [][]:**Initial values of the Satisfactory factors of each resource
  - **TempM [][]:** Initial values of the resources
  - **Cputime :** starting execution time given by the system
- **Outputs:**
  - **TMax(k,2) :** Optimum value of the RAM to kth VM
  - **Satisf(k,2) :** Optimum satisfactory factor of the RAM of the kth VM
  - **Timearray[] :** Required time to estimate the minimized resources for each request.

```
b) RAM satisfactory

1. satisram=TempMSatis(2*k,2)
2. xram=TempM(2*k,2)
3. t2=cputime //t2 is used to optimize the execution time
4. while((satisram>1)||(satisram<0.75)|| ((cputime-t2)>0.02)) do
    if((cputime-t1)<0.02) then
      // check if the time to calculate the required RAM is minimized
      if((satisram>1)) then
        xram=xram-1
      endif
      if((satisram<0.75)) then
        xram=xram+1
      endif
      satisram=xram/RAM(k)
    endif
   endwhile // end of RAM while
5. TMax(k,2)=xram // Optimum value of the RAM
6. Satisf(k,2)=satisram // Optimum value of the satisfactory factor of RAM
```

Function3.3(): Find the Resource type based on the Resource Type table and the Sliding Window technique

- **Inputs:**
  - **n:** Number of virtual machines to create
  - **k:** the index of the IVR creation request
  - **BW[] :** Required BW for each new VM using the BW equation
  - **TempMSatis [][]:**Initial values of the Satisfactory factors of each resource
  - **TempM [][]:**Initial values of the resources
  - **Cputime :**Starting execution time given by the system
- **Outputs:**
  - **TMax(k,3) :** Optimum value of the BW to kth VM
  - **Satisf(k,3) :** Optimum satisfactory factor of the BW of the kth VM
  - **Timearray[]:** Required time to estimate the minimized resources for each request.

```
c) BW satisfactory

1.  satisbw=TempMSatis(2*k,3)
2.  xbw=TempM(2*k,3)
3.  t3=cputime
4.  while((satisbw>1)||(satisbw<0.75)|| ((cputime-t3)>0.02))
      if ((cputime-t1)<0.02) then
       // check if the time to calculate the required BW is minimized
        if((satisbw>1)) then
           xbw=xbw-1
        endif
        if((satisbw<0.8)) then
           xbw=xbw+1
        endif
        satisbw=xbw/BW(k)
        endif
    endwhile // end of the BW while
5.  TMax(k,3)=xbw
6.  Satisf(k,3)=satisbw
```

Function3.4(): Find the Resource type based on the Resource Type table and the Sliding Window technique

- **Inputs:**
  - **n: N**umber of virtual machines to create
  - **k:** index of the IVR creation request
  - **Disk[] :** Required RAM for each new VM using the RAM equation
  - **TempMSatis [][]:**Initial values of the Satisfactory factors of each resource
  - **TempM [][]:**Initial values of the resources
  - **Cputime :** Starting execution time given by the system
- **Outputs:**
  - **TMax(k,4)** : Optimum value of the Disk to kth VM
  - **Satisf(k,4) :** Optimum satisfactory factor of the Disk of the kth VM
  - **Timearray[]:** Required time to estimate the minimized resources for each request.

```
d) Disk satisfactory

    1. satisdisk=TempMSatis(2*k,4)
    2. xdisk=TempM(2*k,4)
    3. t4=cputime;
    4. while((satisdisk>1)||(satisdisk<0.8)|| ((cputime-t4)>0.02)) do
          if((cputime-t1)<0.02) then
          // check if the time to calculate the required Disk space is minimized
          if((satisdisk>1)) then
            xdisk=xdisk-1
          endif
          if((satisdisk<0.8)) then
            xdisk=xdisk+1
        endif
        satisdisk=xdisk/Disk(k)
      endif
     endwhile //end disk while
    5. TMax(k,4)=xdisk
    6. Satisf(k,4)=satisdisk
```

For the second algorithm, we will describe the implemented functions used to distribute the coming requests and minimize the required time to execute them. The task scheduling algorithm is responsible to assign the tasks to create the IVR to the appropriate processors so that there will not be a processor which is under loaded or overloaded.

Main (): Task Sceduling Calculation Program for IVR applications using the Genetic Algorithm (GA):

- **Inputs:**
    - **N:** Number of tasks to schedule
    - **I []:** Number of processor
- **Outputs:**
    - **MaxFitness:** Maximum value of fitness
    - **TTAsk[]:**Tasks assignment across the different processors

```
1.  Print "Enter the number of tasks:"
2.  Read n // n is  number of tasks
3.  Print "Enter the number of processor"
4.  Read i // i is number of processors
5.  Tv = 6 // Tv is the required time to create a virtual machine
6.  Tr = randint(1,n,[6,20]) // Tr is the time to compute the resources for tasks (1st
    algorithm), randint generates one vector of random number between 6 and 20.
7.  Create Initial Random Fitness based on this Function :
        RandomFit_Fun=(z) abs(-(1/2)*z^2 + 3*z + 6)
8.  Print "Pick random fitness values to start, example [4 1.5 3.1 4.4 6 4.8 3 1]:"
9.  Read y //y is a vector composed of different random fitness values
10. Generate the Fitness from the fitness values using the random Fitness function
        for j = 1 to i do
                x(j)=RandomFit_Fun(y(j))
        endfor
11. FitnessArray= [10,10]// create a matrix that contains the Fitness values of each
    process allocation
12. FitnessArray =0 // initialize the matrix by zeros
13. testprocessor = 0 //it is used to test if a processor is not selected.
14. for loop = 1 to 100 do
        Function 1(): Use of the Roulette Wheel to assign Task across the different
        processors
        Function 3(): CrossOver and Mutation between A and B and the Fitness
        Computation
         x=Sum1
        //use the values of Sum1 vector as the new probilities to assign the tasks across
        the processors instead of the roulette wheel technique
         for k= 1 to 100 do
            FitnessArray(loop,k)=f(k)// Copy the values of the obtained fitness from
            Function3()
         endfor
    endfor //end for loop from 1 to 100
15. MaxFitness=max(FitnessArray)//look for the best fitness after 100*100 cycles and
    find the appropriate task allocation (TTask vector)
```

Function 1(): Use of the Roulette Wheel to assign Task across the different processors

- **Inputs:**
  - **n :** Number of tasks to schedule
  - **I []:** Number of processor
- **Outputs:**
  - **TTAsk[]:** Number of tasks assigned to each processors
  - **A[]:**Assignment of task across the different processors

```
1. while (testprocessor==0) do
   Function(): Roulette Wheel()
   A=[]// Assignment of task among the different processors.
   index1=1 //index of vector A
   for j =1 to n do
     for k=1 to length(sl1) do
        if(sl1(k)==j)then
          A(index1)=k
          index1=index1+1
        endif
     endfor
   endfor
   TMatrix=zeros(i,n)// create a matrix TMatrix and initialize it by zeros
   for j =1 to n do
            TMatrix(sl1(j),j)=Tr(j)
   endfor
   for j = 1 to n do
             Print "For Task", j "Processor number", sl1(j)), "is selected"
   endfor
    //Vector d indicates the number of assigned task for each processor
    for j = 1 to n do
         d(sl1(j))=d(sl1(j))+1
    endfor
    indtask=1// index for the TTask vector
     //test if all processors are selected or not: testprocessor=0 or testprocessor=1

    for j= 1 to i do

         Print("Processor", j, "is selected times", d(j))

         TTask(indtask)=d(j)

         indtask=indtask+1

         if (d(j)==0) then // there is a processor which is not selected

            testprocessor=0

         endif

      endfor

   endWhile
```

Function 2(): The Roulette Wheel Technique

- **Inputs:**
  - **N:** Number of tasks to schedule
  - **I []:** Number of processor
  - **X[]:** The fitness values vector (Main()) using the random Fitness function
- **Outputs:**
  - **sl1[]:** Assignment of the tasks across the processors

```
1.  total=0 // total: the total summation of the generated Fitness
2.  for j = 1 to I do
      total=total+x(j)
    endfor
3.  k=0 // temporary variable
    // p(j) = Probability of the jth value used to calculate the cumulative  probability
    // cp(j) = cumulative probability of the jth value. The last value equal 1
4.  for j = 1 to I do
      p(j) = x (j)/total
      k    = k+p(j)
      cp(j) = k
    endfor
    //generating an n X n array of random numbers We only need 'n' random
    numbers so  we use only the first row of the n X n Array
5.  rn1=rand(n)
6.  for j= 1 to n do
7.    g1(j) = rn1(1,j)
    endfor
    // The selection process involves choosing the processor on the basis of which
    class the random number generated by the wheel belongs
8.  for j= 1 to n do
        for a= 1 to I do
            if (cp(a)>=g1(j)) then
                sl1(j)=a // Each task selects one processor
                break
            endif
        endfor
    endfor
```

Function 3(): CrossOver and Mutation between A and B and he Fitness Computation

- **Inputs:**
  - **n :** Number of tasks to schedule
  - **I []:** Number of processor
  - **A[]:**Assignment of task among the different processors
  - **TTAsk[]:**Number of tasks assigned to each processors
  - **Cp[]:** cumulative probability generated by the roulette wheel technique.
  - **Tv:** Required time to create a virtual machine
  - **Tr[]:**Time to compute the resources for tasks (1st algorithm), randint generates one vector of random number between 6 and 20.
- **Outputs:**
  - **f[]:** Obtained fitness values after 100 cycles.

```
1. B=[]
2. f=[]
3. Sum1=[]
4. for intx = 1 to 100 do
     B=A
     A1=[]
     A1(n)=0
     B1=[]
     B1(n)=0

     AandA1=0// then A = A1
     AandB=0// then A = B
     while((AandA1==0)|| (AandB==0)) do
        Function4(): Generate B for crossover (same procedure like A vector)
        Function5(): Crossover and Mutation between A and B vectors
     endwhile   // end of while((AandA1==0)|| (AandB==0))

     Function(6): Calculate the fitness for each processor using
                  TTask(nb of tasks  for each processor)
                  and A1 (Task Assignment)
   fitnesss=U/TaskSpan   //Fitness Function
   f(intx)=fitnesss    //Fitness Value in the FitnessArray(loop, intx)
   A=A1   //Copy A1 into A for the new cycle
endfor //end for intx from 1 to 100 do
```

Function 4(): The Roulette Wheel Technique

- **Inputs:**
  - **n:** Number of tasks to schedule
  - **I []:** Number of processor
  - **Cp[]:** Cumulative probability generated by the roulette wheel technique.
- **Outputs:**
  - **B[]:** Assignment of the tasks across the processors in vector B[]

```
//Generate B for crossover (same procedure like A vector)
        1. rn2=rand(n)
        2. for j= 1 to n do
        3. g2(j)=rn2(1,j)
          endfor
        4.  for j= 1 to n do
                for a = 1 to i do
                        if (cp(a)>=g2(j))then
                                        sl2(j)=a
                        break
                        endif
                endfor
        endfor
        5. index2=1
        6. for j = 1 to n do
          for k =1 to length(sl2)
            if(sl2(k)==j)then
            B(index2)=k
            index2=index2+1
            endif
          endfor
        endfor
```

Function5(): Crossover and Mutation between A and B vectors

- **Inputs:**
  - **n:** Number of tasks to schedule
  - **AandB:**
  - **A[]:** Assignment of the tasks across the processors in vector A[]
  - **B[]:** Assignment of the tasks across the processors in vector B[]
- **Outputs:**
  - **A1[]:** New assignment of the tasks across the processors after Crossover and Mutation
  - **B1[]:** New assignment of the tasks across the processors after Crossover and Mutation

```
if (AandB==1)then
    // 1. CroosOver Operator
         FirstIndexFound=0
         //chosing the starting point for crossover
         SP=1
         while(FirstIndexFound==0) do
             A1(SP)=A(SP)
             B1(SP)=B(SP)
             x=B(SP)
             SP=find(A==x)
             if(SP==1) then
                 FirstIndexFound=1
             endif
             endwhile
         for k = 1 to n do
             if(A1(k)==0)then
                 A1(k)=B(k)
                 B1(k)=A(k)
             endif
         endfor
  // 2. Mutation Operator
         TMutation=[]
         TMutation=randperm(n)
         ind1=TMutation(1)
         ind2=TMutation(2 )
         temp=A1(ind1)
         A1(ind1)=A1(ind2)
         A1(ind2)=temp
 endif  // end of if (AandB==1)
```

Function6(): Calculate the fitness for each processor using TTask(nb) of tasks for each processor) and A1 (Task Assignment):

- **Inputs:**
  - **n:** Number of tasks to schedule
  - **TTAsk[]:** Vector that contains the number of tasks assigned to each processors
  - **Tv:** Required time to create a virtual machine
  - **Tr[]:** Time to compute the resources for tasks (1st algorithm), randint generates one vector of random number between 6 and 20.
  - **A1[]:** Assignment of the tasks across the processors in vector A1[]
- **Outputs:**
  - **Utility:** Compute the value of the utility of the processors using Equation (3.10)
  - **TaskSpan:** Maximum of the execution time among the processors
  - **Sum1[]:** use the values of Sum1 vector as the new probabilities to assign the tasks across the processors instead of the roulette wheel technique

```
1. int=1
2. for k =1 to n do
     Sum1(k)=0
     for j = int to ((int+TTask(k))-1) do
        Sum1(k)=Sum1(k)+Tv+Tr(A1(j))
     endfor
     int=int+TTask(k)
   endfor
3. MaxSpan=max(Sum1)
4. MinSpan=min(Sum1)
5. TaskSpan=maxx-minx //TaskSpan Function
6. Utility=1
7. for l = 1 to nbtask do
     if(Sum1 (l)≠0)then
        Utility=Utility*(Sum1 (l)/MaxSpan)//Utility Function
     endif
   endfor
```

We propose to test the two algorithms while changing the parameters of the execution such as the call arrival rate for the first algorithm to calculate the required resources and the number of the coming request for the second algorithm to schedule them adequately across the different processors. 1) The call arrival rate values for the first algorithm were: 25, 50, 75, 100, 150, 200, 250, 300, 350, 350, 400, 450, 500, 500, 550 and 600 calls/min. 2) For the second algorithm, the test parameters are:

- **Changing the number of tasks:** 8 processors, Window size =10, Generation size = 10 and Number of tasks = from 0 to 1500.
- **Changing the window size:** 8 processors, Window size = from 10 to 60, Generation size = 10 and Number of tasks = 100.
- **Changing the generation size:** 8 processors, Window size = 10, Generation size = from 10 to 60 and Number of tasks = 100.

4.5 Conclusion

In this chapter, we have introduced the proposed prototype to implement our virtualized architecture for IVR applications. We proposed also a testing scenario to evaluate our prototype including the discovery, the publication, the activation and the creation phases. We described also the environmental settings to execute both the architecture and the algorithms behind. We will present in the next chapter the performances results and analyze them to prove the utility of the two proposed algorithms in our architecture that can handle IVR application in virtualized infrastructures.

CHAPTER 5

RESULTS AND ANALYSIS

5.1 Introduction

In the previous chapter, we described the proposed prototype to implement our architecture and the resource management algorithms. Furthermore, a scenario was discussed briefly to describe some aspects of the testing environment. In this chapter, we will present the performance results of the two algorithms and analyze them: the first section describes the computational resources sharing results whereas the second section discusses the instantiation request scheduling results.

5.2 Computational Resources Sharing

For the computational resource sharing algorithm, we propose to compare the required resources (CPU, memory, bandwidth and disk space) with the allocated ones so that we can later calculate the satisfactory factors of these resources. The required resources are estimated using the model we describe in chapter 3 while the allocated ones are calculated based on the proposed GA computational resource sharing algorithm according to the call arrival rate. The comparison will allow us to see if the example of resource combination types given in Table 3.2 and that which is usually used in cloud computing environment is suitable to well manage the available resources *(CPU_c, M_c, B_c, D_c)*. For the instantiation request scheduling algorithm, we compute the total completion times and average utilization of processors to compare the resource usage efficiency of the proposed fitness function with those of the dynamic and random algorithms.

The required and allocated CPU, memory and bandwidth were computed according to call arrival rate. As described in Table 3.2, the required resources for call arrival rates less than 60 calls/min are less than 1 GHZ of CPU 256 MB of memory, 350 Mbps of bandwidth and

1GB of disk space. Because the performance measurements are too small for small-sized IVR, we computed the required and allocated resources for large-sized IVR (i.e. for $\lambda \geq 60$ calls/min).

5.2.1 CPU Computation Results

For a call arrival rate $\lambda = 60$ calls/min, the required CPU was almost 1.2 GHz and the estimated resource was 1.5 GHZ which represents a typical resource combination type provided by the virtualization machine as illustrated in table 3.2. While changing the call rate arrival, the obtained results show that the required CPU and the allocated CPU increase in parallel as shown in Figure 5.1. We noticed also that the difference between the required CPU and the allocated valued is too small due to the fact the CPU combination types were predefined as 1.5 GHz, 2 GHz, 2,5 GHz, etc, while the estimated values were 1.3 GHz, 2,2 GHz, 2,6 GHz, etc, that is why the difference were too small.

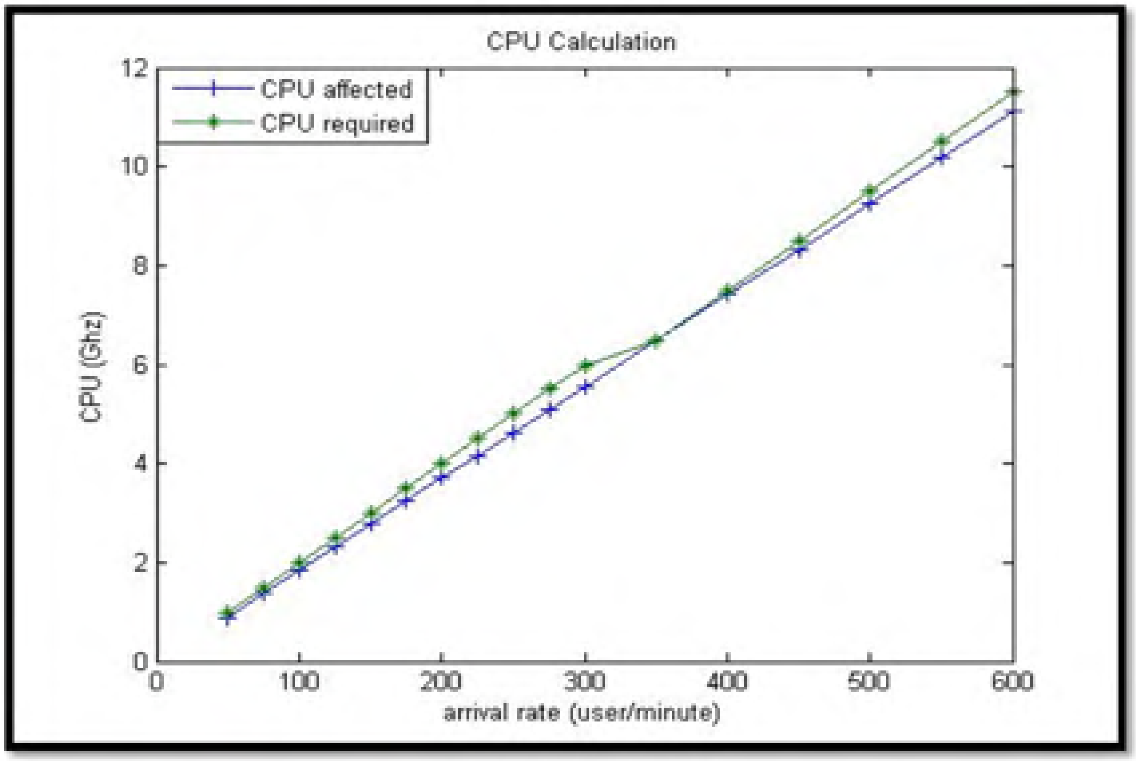


Figure 5.1 Required and allocated CPU

5.2.2 Memory Computation Results

For a call rate λ = 60 calls/min, the required memory was almost 131 MB and the estimated memory was 256 MB. Based on the given results in the Figure 5.2,, we noticed that the required and the estimated values are increasing simultaneously. Figure 5.2 shows that the required resources are less than the allocated ones. Therefore, a typical resource combination type provided by the virtualization machine should be improved to optimize memory usage and sharing among different requests.

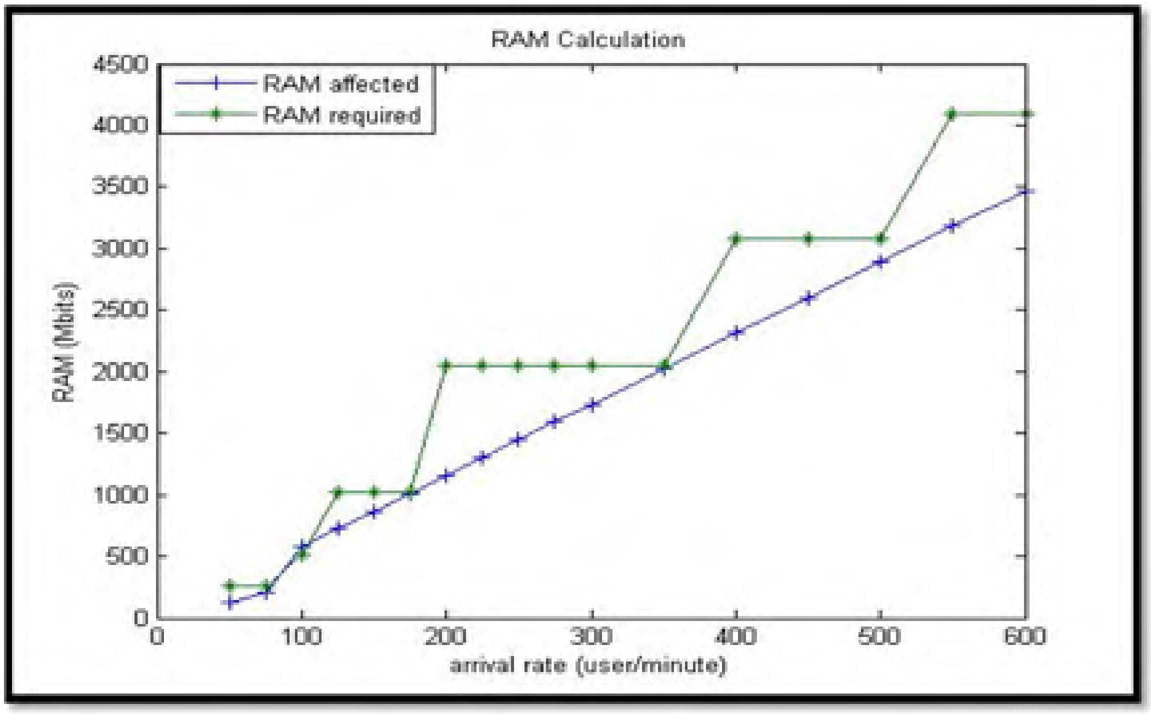


Figure 5.2 Required and allocated memory

5.2.3 Bandwidth Computation Results

The required bandwidth was almost 350 Mbps and the estimated resource was 350 Mbps for a call arrival rate λ= 60 call/min. We observed clearly in Figure 5.3 that the performance results of the allocated and estimated values are very close because of the size of the resources combination type were defined closer enough to required bandwidth (100 Mbps, 200Mbps, 300Mbps, etc.).

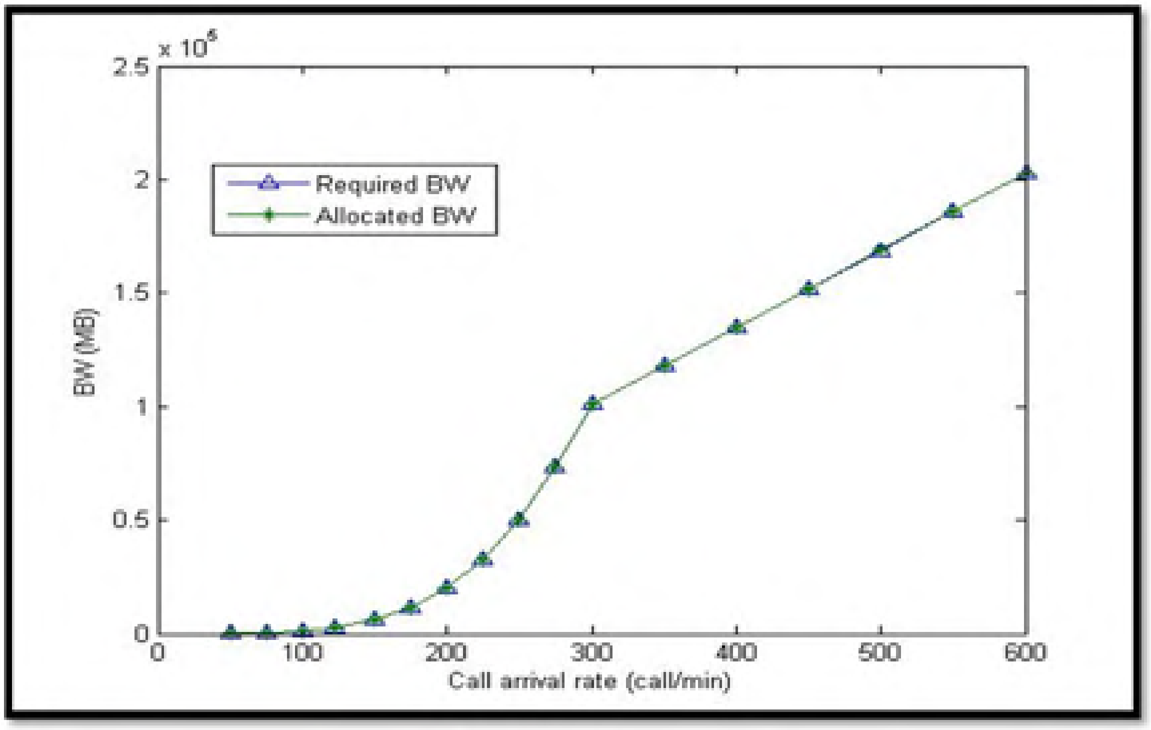


Figure 5.3 Required and allocated bandwidth

5.2.4 Disk Space Computation Results

Regarding the required disk space, the obtained results were very small even for higher call arrival rates (e.g., λ=600 calls/min) always staying under 1GB. This is due to the nature of the IVR applications we select to deploy them in our prototype, which need little disk space. The disk space measurements were therefore not included in this section because the estimated value was the same (i.e. 1 GB).

5.2.5 CPU, Memory and Bandwidth Satisfactory factor Results

As shown in Figure 5.1, Figure 5.2 and Figure 5.3, the required and allocated resources increased as the call arrival rates were increased. These performances were expected because each IVR call requires specific SII resources (CPU, memory and bandwidth) to be executed. In figure 5.1, the difference between the required CPU and the allocated value was small and according to the CPU satisfactory factor (Figure 5.4) the resource usage percentage was greater than 90%. For λ greater than 350calls/min, this percentage was more than 95%. The required and the allocated bandwidth were almost the same (Figure 5.3) and the bandwidth usage was almost 100% (Figure 5.4). The required and allocated memory measurements were different due to the fact that the sizes of the resource combination types were predefined (e.g., 256MB, 512MB and 1024MB).

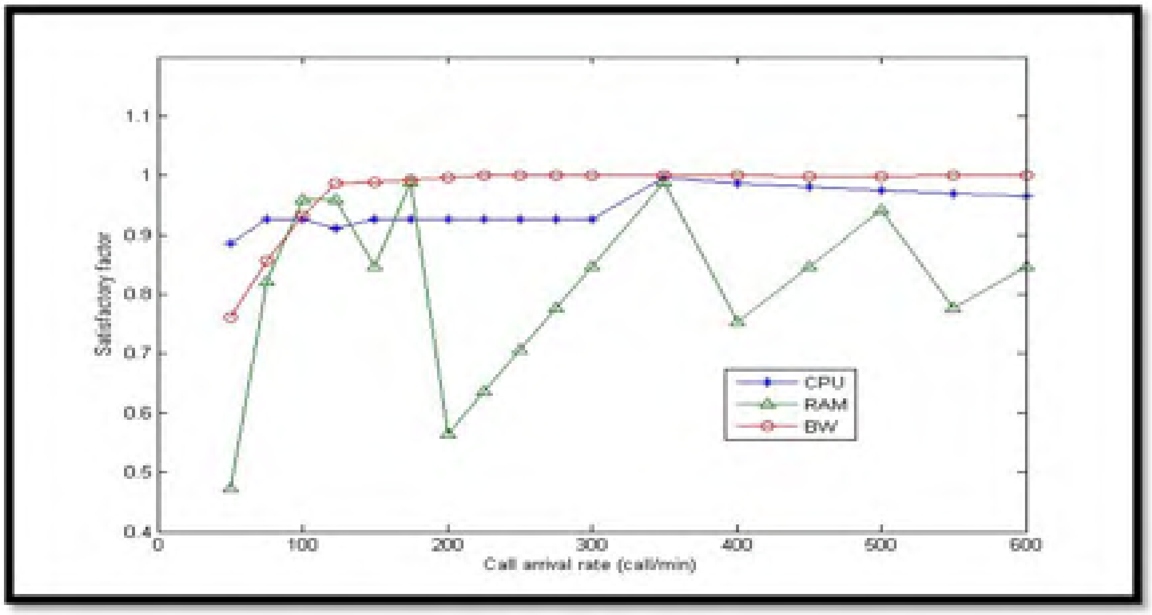


Figure 5.4 CPU, memory and bandwidth satisfactory factors

For instance, for λ=60 calls/min the required memory was almost 131 MB while the allocated memory was 256 MB. Only half the capacity memory was therefore used. For λ=100 calls/min, the required memory was almost 500 MB and the allocated memory was 512 MB, representing 97% memory usage. Therefore, for a large-sized IVR, the higher the call arrival rate, the higher the percentage of CPU and Bandwidth resource usage. Unlike the CPU and bandwidth performance improvement in terms of resource usage, Figure 5.3 and Figure 5.4 show that the memory usage was not that efficient. In fact, the memory resource needs for an IVR application are small. The difference between the required memory and the allocated value varies according to the call arrival rate (Figure 5.3). The percentage of the resource usage varies from 50 % to almost 100 % for call arrival rates less than 250 call/min and varies from 70% to 100% otherwise. As a conclusion, the resource combination types usually used in could environments do not allow for efficient resource usage. New combination types are therefore needed.

5.3 Instantiation Request Scheduling

5.3.1 Test Parameters

For the instantiation scheduling algorithm, we compute the total completion time and the average processor utilization in order to evaluate the resource usage efficiency of the proposed fitness functions. These values are calculated according to the number of tasks, the sizes of window and number of generation cycles. We will compare also the obtained results of the algorithm with those of dynamic and random scheduling algorithms. These two algorithms are based on selection, crossover and mutation methods. However, the random algorithm doesn’t balance the number of tasks across the processors. A set of tests were performed using the following default values: 100 instantiation requests, 8 processors, window size of 10 requests, generation cycles of 10, each request of 20 units of time, and population size of 10.

5.3.2 Changing the number of tasks

To compute the total completion time and the average processor utilization, the test parameters were set to the default values and we varied the number of tasks from 0 to 1500. As shown in , the total completion time for the three algorithms increased as the number of tasks increased. Hence, the higher the number of tasks to be scheduled, the longer the total completion time. Moreover, the instantiation request scheduling algorithm provided a better performance than the two other algorithms. In Figure 5.5, the average processors utilization is much higher for instantiation request scheduling algorithm than for the dynamic and the random algorithms. The means of these utilizations were 0.83, 0.74 and 0.58 for instantiation request Scheduling, dynamic and random algorithms respectively. These performance behaviors are due to the fact that the instantiation request scheduling algorithm provides a fitness function that guarantees a better processors utilization and the faster task execution times than those defined for the dynamic and random algorithms. Therefore, it requires more processing.

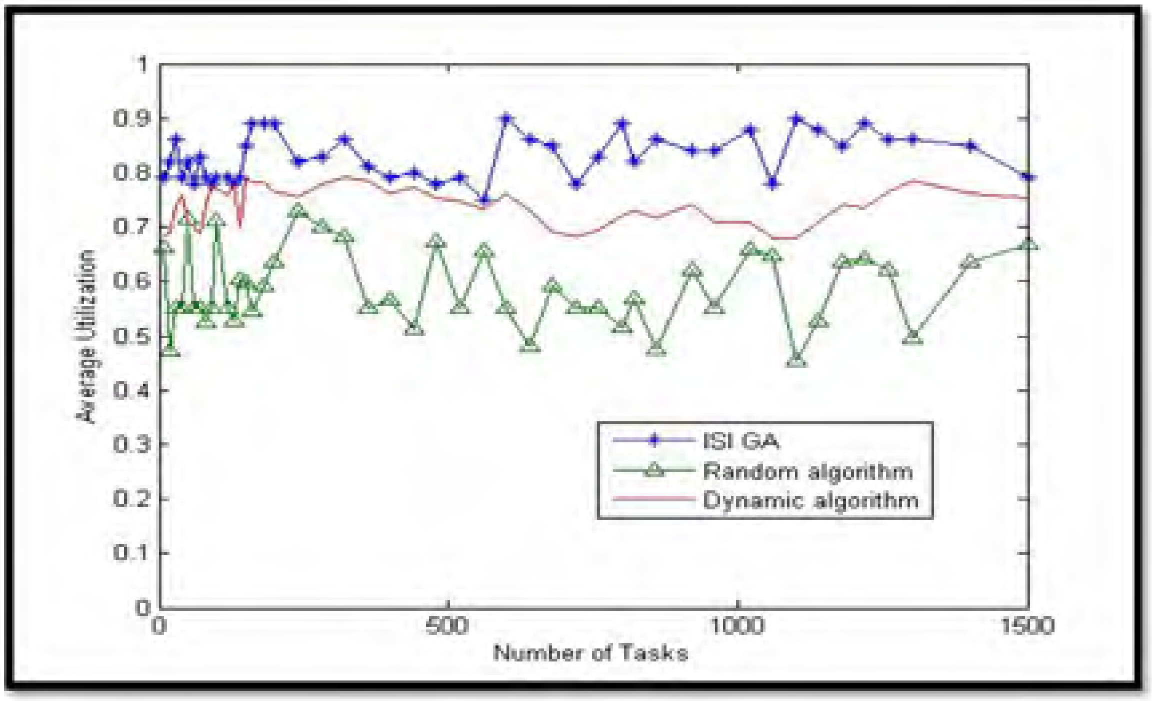


Figure 5.5 Utilization of processors according to number of tasks

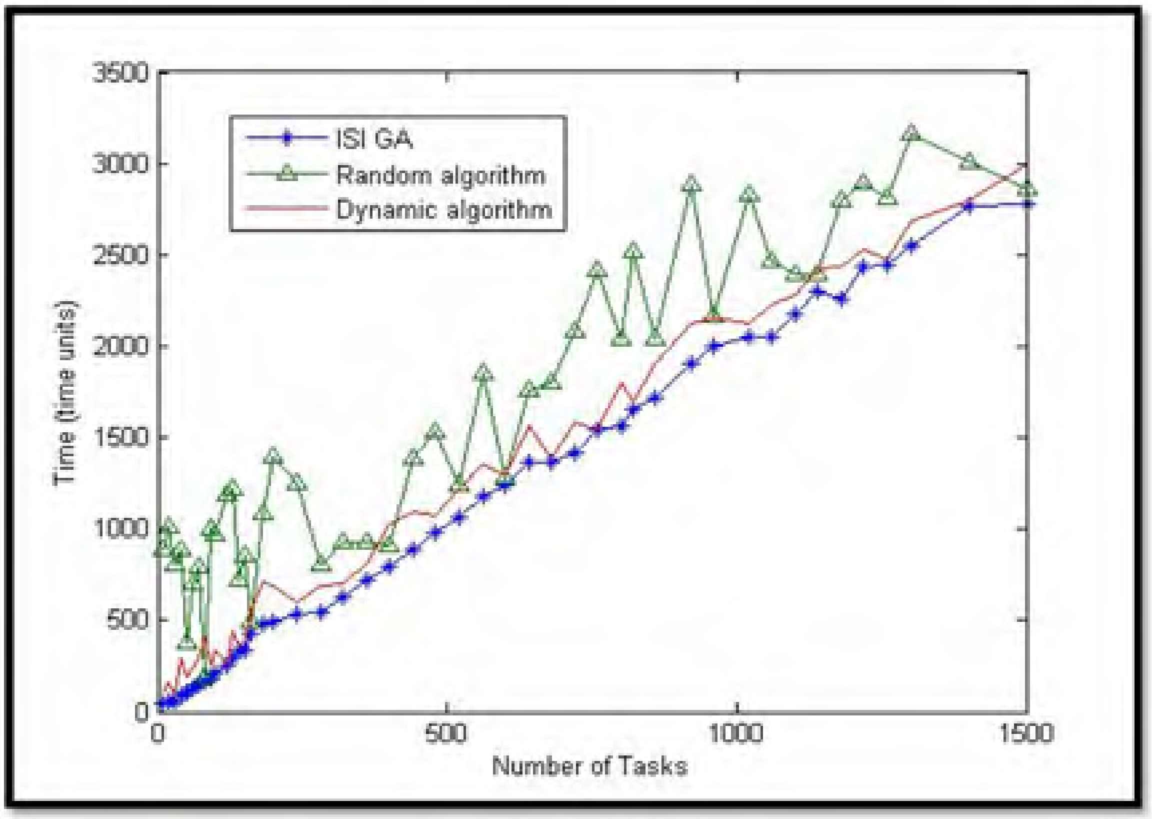


Figure 5.6 Completion time according to number of tasks

5.3.3 Changing the window size

We also computed the total completion time and the average processor utilization according to the window size. Figure 5.7 and Figure 5.8 both illustrate these performances for 100 tasks. The total completion time decreased as the window size was increased and the average processor utilization improved as the window size increased for instantiation request scheduling and dynamic algorithms. Moreover, the instantiation request scheduling algorithm outperformed the dynamic scheduling and random algorithms. This performance improvement shows that the increasing number of tasks to be scheduled was well handled by the 8 processors.

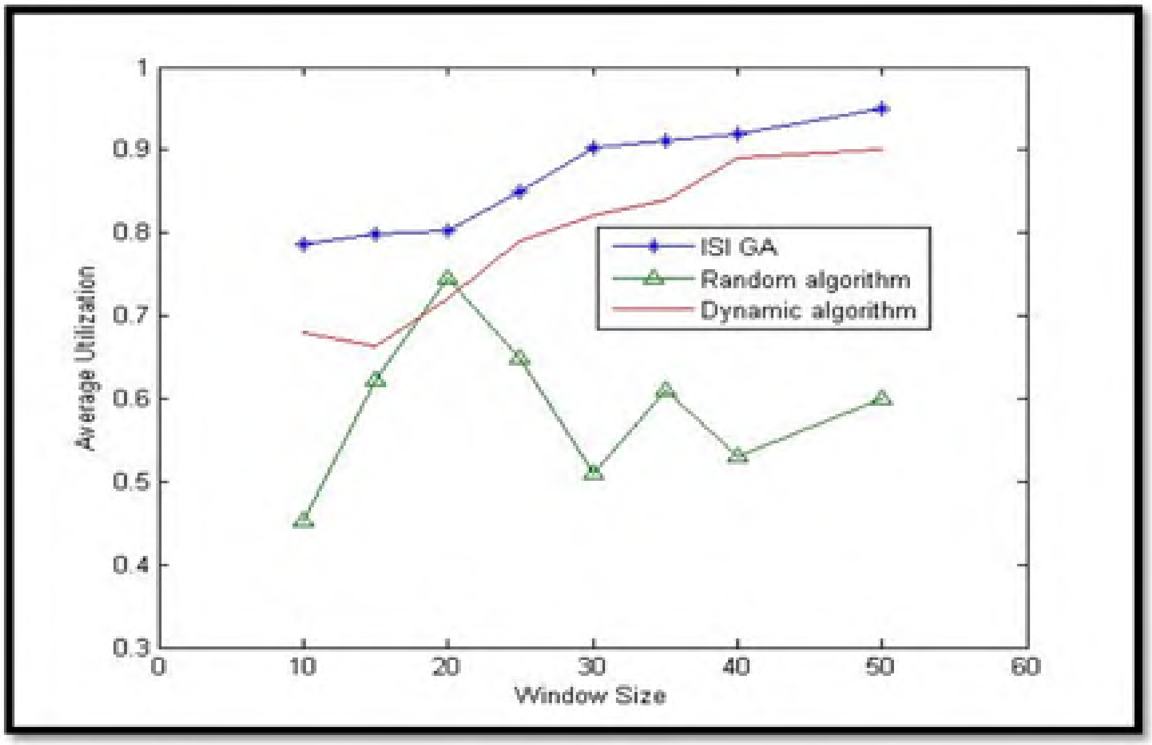


Figure 5.7 Average Utilization of processors according to window size

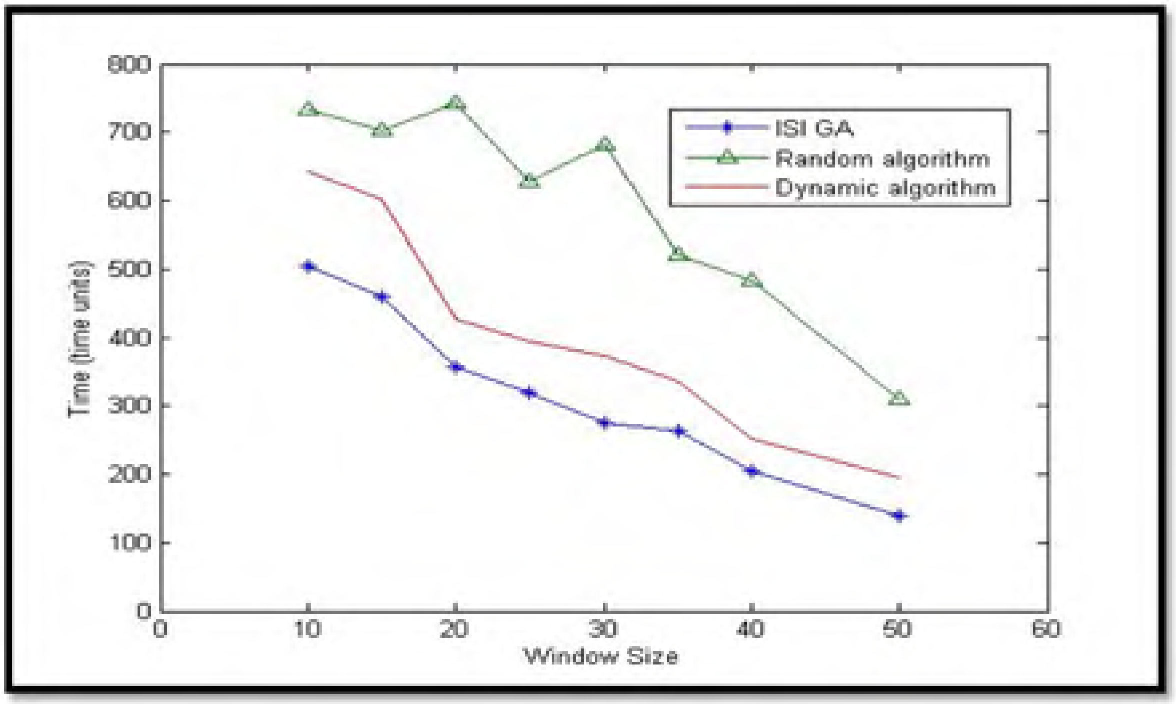


Figure 5.8 Completion time according to window size

5.3.4 Changing the number of generation

We analyzed the effect of the number of generation cycles on the instantiation request scheduling algorithm. We varied the number of generation from 0 to 60 for a 10 request window size and a task number of 100. Figure 5.9 and Figure 5.10 show the total completion time and average processor utilization according to the number of generation cycles. The total completion time and the average processor utilization decrease as the number of generation cycles increase. A significant reduction in completion time and improvement in processor utilization were noticed when varying the number of generation cycles from 10 to 30. These performances were expected because increasing the number of generation cycles improves the task assignment quality. Moreover, the instantiation request scheduling performed better than the dynamic and random algorithms.

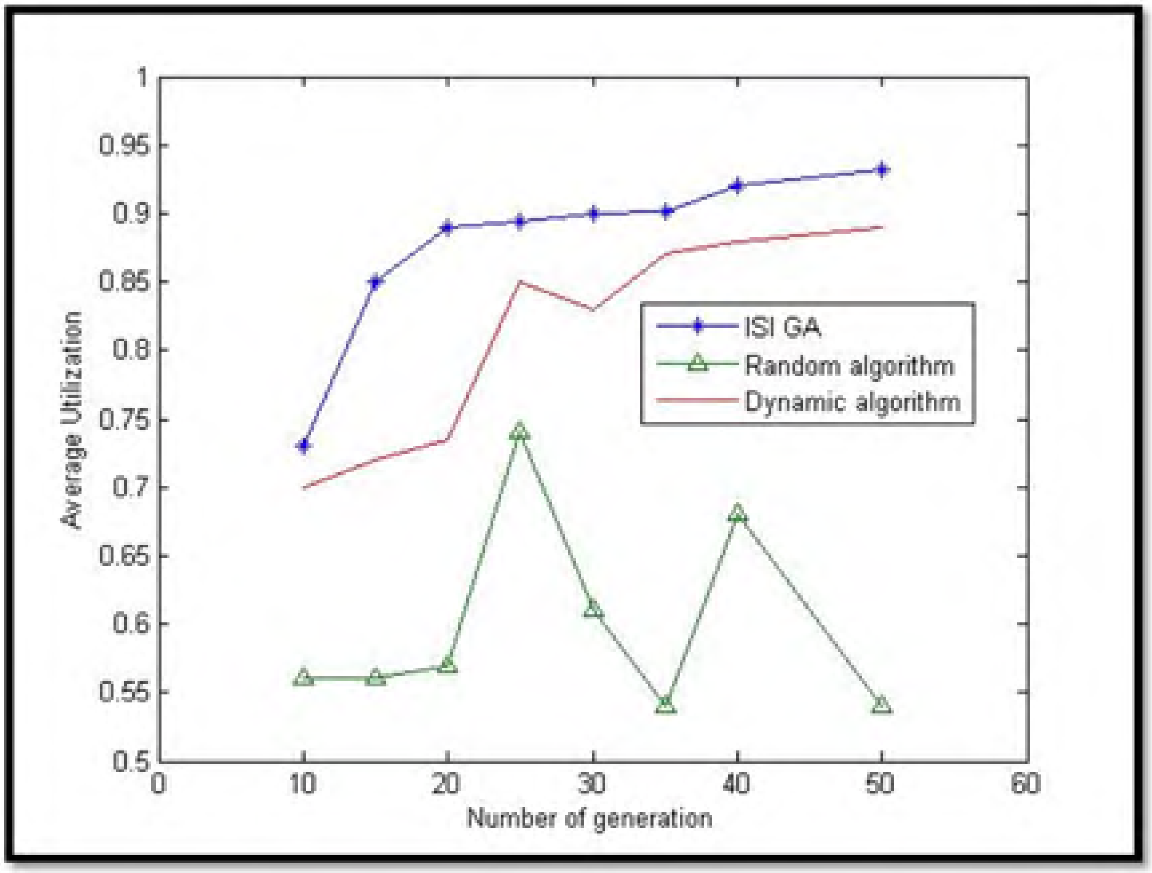


Figure 5.9 Average utilization according to number of generation

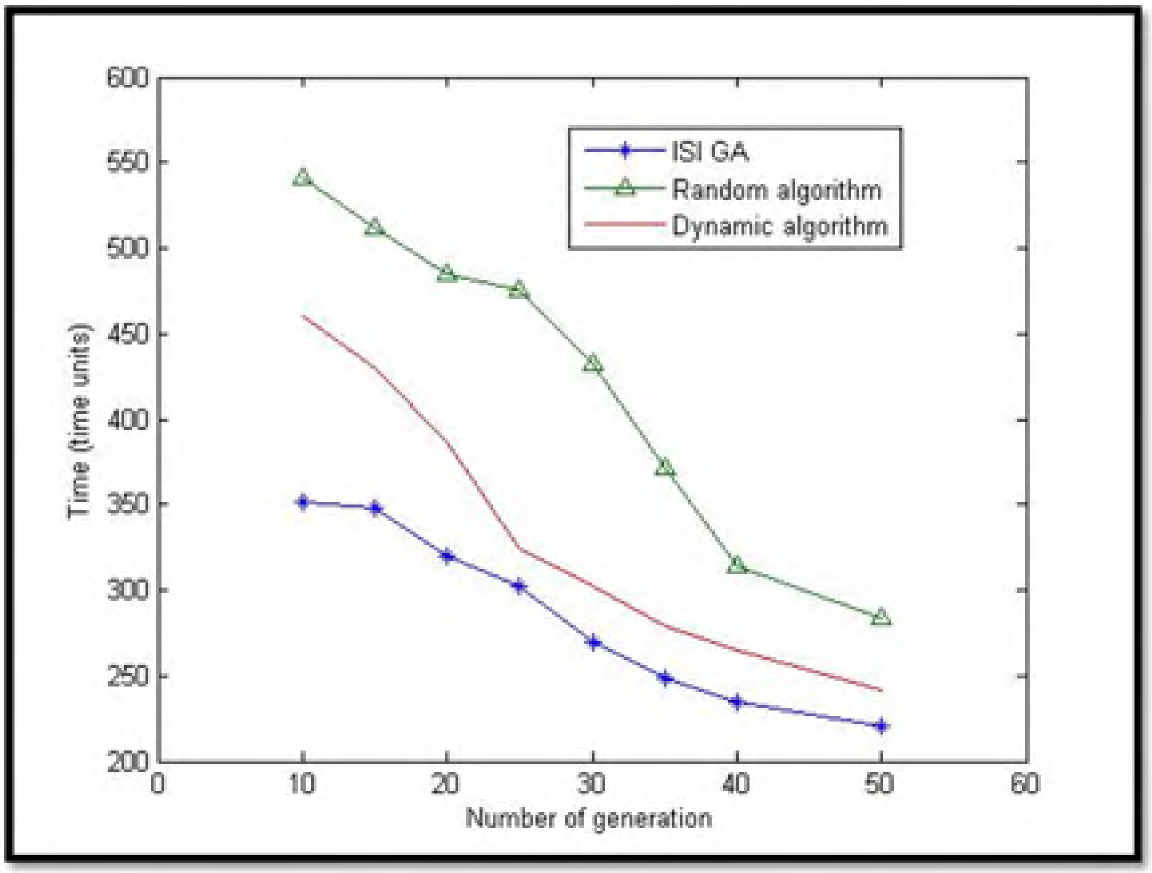


Figure 5.10 Completion time according to number of generation

5.4 Conclusion

In this chapter, we described the obtained results of the proposed resources management algorithms and compared them with dynamic and random algorithms results. The performance measurements conducted showed that the proposed algorithms are promising. Indeed, compared to dynamic and random algorithms, the proposed instantiation request task scheduling GA outperformed in terms of total completion time and average processors utilization. The computational resource sharing algorithm allows efficient CPU and bandwidth usage. However, due to the resource combination types used and because the memory resource needs are small for IVR applications, the memory resource usage was not that efficient

CONCLUSION

Interactive Voice Response (IVR) is a technology that enables interactions with automated information systems. IVR applications are ubiquitous. They are used in many industries, such as in telecommunications and banking. Nowadays, several IVR systems such as call centres allow users to remotely access IVR application as services. However, to the best of our knowledge none of them allow to share different IVR substrate components and to compose them, and none of these applications rely on virtualized infrastructures.

In this project, we proposed a new virtualized architecture for IVR applications. Virtualization is a technology that can ease the development of IVR applications and their deployment in cloud environment. IVR virtualization will enable efficient resource usage, by allowing IVR applications to share different IVR substrate components.

Therefore, we defined a new virtualized architecture for IVR applications and demonstrated its potential by developing a prototype with specific scenario. This architecture proposes IVR substrates that are virtualized and composed to build new IVR applications on the fly. It is based on a business model that defines different roles such as IVR substrate provider, infrastructure provider, platform provider and service providers. Based on this model, we proposed a virtualized architecture which eases the development and the management of new IVR applications. Our contributions are three folds: 1) a new business model; 2) a new virtualized architecture for IVR applications based on this business model; 3) a proof of concept to demonstrate the benefits of the proposed architecture.

The proposed model is composed of an IVR substrates provider, an IVR infrastructure provider, an IVR platform provider, an IVR service provider, a Broker, and a Connectivity provider. Based on this model, we proposed also a novel architecture for a virtualized IVR infrastructure that includes two layers, three planes and a repository. It enables easy development and management of the new IVR applications via its different components. The main functional entities of the architecture are the virtual IVR engine and the substrate IVR engine. The interactions between the two layers, the planes and the different entities are organized using three types of interfaces: service I/F, management I/F, and publication & discovery I/F. Then, we designed the proposed software architecture and we described its entities, the planes and the software operational procedures including the creation, activation, management and execution phases.

Moreover, we proposed IVR resource management and task scheduling strategies to guarantee efficient resource usage by IVR applications. These strategies are parts and parcels of IVR virtualization. The virtualization task scheduling and the computational resource sharing (among different IVR applications) strategies are based on genetic algorithms, in which different objectives are optimized. The algorithms used by both strategies are simulated and the performance measured and analysed. The task scheduling algorithm maximizes resources utilization while minimizing the execution time of tasks. The computational resource sharing algorithm minimizes the substrate resource utilization and the resource allocation time while maximizing the satisfactory factor of the IVR applications.

We implemented automated attending scenario to test the proposed prototype and the two proposed algorithms using different parameters. The performance results conducted showed that the proposed algorithms are promising as they can fulfill the objectives. The resources sharing algorithm can minimize the use of physical resources based on some statistics we get to estimate the necessary amount for each request. The second algorithm can balance the load between the processors to optimize their utility and maximize their fitness.

Our proposed virtualized IVR architecture has been published in a refereed conference and is under review by a refereed journal and the resources management strategies and algorithms are under review by second refereed journal.

Based on the obtained results, we conclude that our virtualized architecture is promising solution to develop IVR applications as services in cloud settings. Furthermore, it enables the development and management of new IVR based applications via a simplified platform.

Nevertheless, we still need some other modules to add them to the proposed architecture like the resources computation workload engine to compute the amount of traffic generated by each created IVR instance and to evaluate the satisfactory factor while the VM is overloaded or under loaded. This module enables to control the processors workload and to optimize their performance. This engine will allow dynamic adjustment of the resources across different VMs using migration techniques of virtual machines to guarantee quality of Service (QoS) continuously when the workload is fluctuating.

Also, we can add the resources negotiation engine that allows selecting new CPU, RAM, BW values so that we can change the allocated resources and adjust them to get better SLA. Furthermore, resource combination types should be defined to improve the memory resource usage. Finally, the proposed virtualized architecture, and resource management and task scheduling strategies should be generalized to support other applications such as audio presence, video conferencing and IPTV that can be offered in cloud settings.

BIBLIOGRAPHY

Abdullah, R. et al., Nov. 2011, “A model of knowledge management system for facilitating knowledge as a service (KaaS) in cloud computing environment”, Research and Innovation in Information Systems (ICRIIS), 2011 International Conference on , vol., no., pp.1-4.

Abarca, C., Peters, M., 1999, “TINA business model for UMTS: benefits and possible enhancements”, Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA '99, vol., no., pp.171-173.

Albert Y. Zomaya et al., August 1999, “Genetic Scheduling for parallel processor systems: Comparative studies and performance issues”, IEEE Transaction on parallel and distributed systems, Vol. 10, No. 8.

Amazon Elastic Computing Cloud, 2011, available at: < http://aws.amazon.com/ec2>, Accessed May 10 2011.

Arub Acharya et al., 2009, "Programmable presence virtualization for next-generation context-based applications", Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference. pp.1-10.

Atel, P.B. and Marwala T., Oct. 2008, "Interactive Voice Response field classifiers," Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on , vol., no., pp.3425-3430.

Barcomb, K.E. et al., Nov. 2011, “A case for DoD application of public cloud computing services”, Military Communications Conference, 2011 - MILCOM 2011 , vol., no., pp.1888-1893.

Bo Peng Hammad, et al., Nov 2011-Dec. 1 2011, “A Network Virtualization Framework for IP Infrastructure Provisioning”, Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on , vol., no., pp.679-684.

Briscoe, G. and Marinos A., June 2009, “Digital ecosystems in the clouds: Towards community cloud computing”, Digital Ecosystems and Technologies, 2009. DEST '09. 3rd IEEE International Conference on , vol., no., pp.103-108.

Cheng-Jen Tang and Miau-Ru Dai , Dec. 2011, “Dynamic computing resource adjustment for enhancing energy efficiency of cloud service data centers”, System Integration (SII), 2011 IEEE/SICE International Symposium on , vol., no., pp.1159-1164.

Cloud Computing Use Cases White Paper, 2 July 2010, Version 4.0, available at: < http://opencloudmanifesto.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf>

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

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

D. Dutta and R.C. Joshi, February 2011, "A Genetic –Algorithm Approach to Cost-Based Multi-QoS Job Scheduling in Cloud Computing Environment", ACM International Conference and Workshop on Emerging Trends in Technology (ICWET 2011)-TCET, Mumbai, India, 422-427.

D. E. Goldberg, 1989, "Genetic algorithms in search, optimization, and machine learning," Reading, Mass. Addison-Wesley.

E. Barrett, E. Howley and J. Duggan, 2011, "A learning architecture for scheduling workflow applications in the cloud," 9th IEEE European Conference on Web Services, pp. 83-90.

El Barachi, M., Kara, N., Dssouli, R., 13-15 Dec. 2010, "Towards a service-oriented network virtualization architecture", Kaleidoscope: Beyond the Internet - Innovations for Future Networks and Services, 2010 ITU-T , vol., no., pp.1-7.

Euiyoul Ryu et al., July 2010 , "MyAV: An all-round virtual machine monitor for mobile environments", Industrial Informatics (INDIN), 2010 8th IEEE International Conference on , vol., no., pp.657-662.

Fatna Belqasmi, Christian Azar, Mbarka Soualhia, Nadjia Kara and Roch Glitho, December 2011 ,"A Virtualized Infrastructure for Interactive Voice Response Applications in the Cloud," ITU-T Kaleidoscope 2011 the fully networked human - Innovations for future networks and services.

Fatna Belqasmi, Christian Azar, Mbarka Soualhia, Nadjia Kara and Roch Glitho, September 2012 (accepted with revision), "A case study of a virtualized Infrastructure and its accompanying platform for IVR applications in Clouds," IEEE Communication Magazine.

Fatna Belqasmi, Nadjia Kara, Roch Glitho, Mbarka Soualhia and Christian Azar, June 2012 (submitted), "Genetic-based algorithms for resource management for virtualized IVR application," IEEE Transactions on Network and Service Management.

Fatna Belqasmi et al., 2011, “A novel virtualized presence service for future Internet”, IEEE International Conference on Communications ICC2011.

Fu Wen and Li Xiang, Dec. 2011, “The study on data security in Cloud Computing based on Virtualization”, IT in Medicine and Education (ITME), 2011 International Symposium on , vol.2, no., pp.257-261.

F. Xhafa et al., 2008, ”Genetic Algorithm Based Schedulers for Grid Computing Systems”, International Journal of Innovative Computing, Information and Control, Vol. 3, No.5, pp.1-19.

Google App Engine, 2011, available at: <http://code.google.com/appengine>, Accessed May 10 2011.

Haibo Zhao and Doshi, P., July 2009, “Towards Automated RESTful Web Service Composition”, Web Services, 2009. ICWS 2009. IEEE International Conference on , vol., no., pp.189-196.

Iang Li et al., Dec. 2009, “Adaptive Management of Virtualized Resources in Cloud Computing Using Feedback Control”, Information Science and Engineering (ICISE), 2009 1st International Conference on , vol., no., pp.99-102.

Jia Yu et al., 2008, “Workflow Scheduling Algorithms for Grid Computing”, available at < http://www.cloudbus.org/papers/MHS-Springer-Jia2008.pdf>

Joanna Kolodie et al., May 2012, “Multi-level hierarchic genetic-based scheduling of independentjobs in dynamic heterogeneous grid environment Multi-level hierarchic genetic-based scheduling of independentjobs in dynamic heterogeneous grid environment”, Journal No 214, pp 1-19.

Junchao Li et al., 2010, “Study on Service-Oriented Cloud Conferencing”. Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on, vol.6. pp.21-25.

Junghan Kim et al, July 2009, “Design and Implementation of Networking Virtualization for Cluster File Systems”, Computational Science and Its Applications, 2009. ICCSA '09. International Conference on , vol., no., pp.79-83.

J. Yu and R. Buyya., 2006, “Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms,” Scientific Programming Journal, Vol. 14, No.3, p.217-230.

Jun-wei Ge et al., Oct. 2010,"Research on Storage Virtualization Structure in Cloud Storage Environment," Multimedia Technology (ICMT), 2010 International Conference on , vol., no., pp.1-4.

Karunamurthy R., July 2007, "A business model for dynamic composition of telecommunication web services," Communications Magazine, IEEE , vol.45, no.7, pp.36-43.

Khan A. et al., Jan. 2012,"Network virtualization: a hypervisor for the Internet?", Communications Magazine, IEEE , vol.50, no.1, pp.136-143.

Konfrst, Z., April 2004, "Parallel genetic algorithms: advances, computing trends, applications and perspectives", Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International , vol., no., pp. 162.

Kyong-I Ku et al., Feb. 2010,"Method for distribution, execution and management of the customized application based on software virtualization", Advanced Communication Technology (ICACT), 2010 The 12th International Conference on , vol.1, no., pp.493-496.

Kyte, I. et al., June 2012 , "Enhanced side-channel analysis method to detect hardware virtualization based rootkits", Internet Security (WorldCIS), 2012 World Congress on , vol., no., pp.192-201.

Lamb, J. , Nov. 2011, "Green IT and use of private cloud computing in South Africa", Emerging Technologies for a Smarter World (CEWIT), 2011 8th International Conference & Expo on , vol., no., pp.1-6.

Leela R. and Selvakumar S.,Jan. 2009, "Genetic Algorithm approach to Dynamic Multi Constraint Multi Path QoS Routing Algorithm for IP networks (GA-DMCMPRA)", Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International , vol., no., pp.1-6.

Li Wenrui et al., June 2012, "A Framework for Self-Healing Service Compositions in Cloud Computing Environments", Web Services (ICWS), 2012 IEEE 19th International Conference on , vol., no., pp.690-691.

L. M. Vaquero et al., Jan. 2009, "A Break in the Clouds: Towards a Cloud Definition", ACM SIGCOMM Computer Communication Review, vol. 39, no. 1.

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

M. D. Kidwell and D. J. Cook, April 1994, "Genetic Algorithm for Dynamic Task scheduling", Proc. IEEE 14th Annual International Phoenix conference on Computers and communications, pp. 61-67.

Moses, J. et al, May 2011, “Shared Resource Monitoring and Throughput Optimization in Cloud-Computing Datacenters”, Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International , vol., no., pp.1024-1033.

O. Morariu, C. Morariu and T. Borangiu, “A genetic algorithm for workload scheduling in cloud based e-learning,” in Proceedings of the 2th International Worshop on Cloud Computing Platforms, pp. 1-6.

Pandey, S. and Nepal, S., June 2012, "Modeling Availability in Clouds for Mobile Computing," Mobile Services (MS), 2012 IEEE First International Conference on , vol., no., pp.80-87.

Pengfei Guo, Oct. 2010, “The enhanced genetic algorithms for the optimization design”, Biomedical Engineering and Informatics (BMEI), 2010 3rd International Conference on , vol.7, no., pp.2990-2994.

Phooi Yee Lau et al., 2010. “Pay-as-you-use on-demand cloud service: An IPTV case”, Electronics and Information Engineering (ICEIE), 2010 International Conference On , vol.1. pp.V1-272-V1-276.

Prangchumpol, D., Nov. 2009, “Server virtualization by user behaviour model using a data mining technique — A preliminary study”, Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for , vol., no., pp.1-5.

Prodan, R. et al, 2012, “Evaluating High-Performance Computing on Google App Engine”, Software, IEEE , vol.29, no.2, pp.52-58.

Rock, M. and Goscinski, A., July 201” , “Execution of Compute Intensive Applications on Hybrid Clouds (Case Study with mpiBLAST)”,Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on , vol., no., pp.995-1000.

Qiang Li et al, Dec. 2009, “Adaptive Management of Virtualized Resources in Cloud Computing Using Feedback Control”, Information Science and Engineering (ICISE), 2009 1st International Conference on , vol., no., pp.99-102.

Qiao L. Iyer et al., April 2005, “SVL: storage virtualization engine leveraging DBMS technology”, Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on , vol., no., pp. 1048- 1059.

Salesforce CRM, 2011, available at: <http://www.salesforce.com/platform>, Accessed May 20 2011.

Shaikh, F.B., Haider, S, Dec. 2011, “Security threats in cloud computing”, Internet Technology and Secured Transactions (ICITST), 2011 International Conference for , vol., no., pp.214-219, 11-14.

Sichao Yang and Hajek, B., August 2007, “VCG-Kelly Mechanisms for Allocation of Divisible Goods: Adapting VCG Mechanisms to One-Dimensional Signals”, Selected Areas in Communications, IEEE Journal on , vol.25, no.6, pp.1237-1243.

Singh, D. et al., May 2012, “High Availability of Clouds: Failover Strategies for Cloud Computing Using Integrated Checkpointing Algorithms”, Communication Systems and Network Technologies (CSNT), 2012 International Conference on , vol., no., pp.698-703.

Skejic, Emir, May 2010, “Virtualization of hardware resources as a method of power savings in data center”, MIPRO, 2010 Proceedings of the 33rd International Convention , vol., no., pp.636-640.

S. Prabhu, 2011, ”Multi-Objective Optimization based on genetic algorithm in Grid Scheduling ”, International Journal of Advanced Research in Technology, Vol. 1, No. 1, pp. 54-58.

S. Tayal, 2011, ”Tasks scheduling optimization for the cloud computing systems”, International Journal of Advanced Engineering Sciences and Technologies, Vol. 5., No. 2, pp. 111-115.

Sungjune Hong et al., Nov. 2005 , “The semantic PARLAY for 4G network”, Mobile Technology, Applications and Systems, 2005 2nd International Conference on , vol., no., pp.1-5.

S. Xu, et al., 2010, “Design of Hierarchical and Configurable IVR System”, 2010 Second International Conference on Computational Intelligence and Natural Computing Proceedings (CINC).

Trihandoyo, A. et al., May 1995, “A real-time speech recognition architecture for a multi-channel interactive voice response system”, Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on , vol.4, no., pp.2687-2690 vol.4.

Van Halteren et al., 1999, “Value added Web: integrating WWW with a TINA service management platform”, Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA '99 , vol., no., pp.14-23.

VMware ESX Server, 2011 , available at: <http://www.vmware.com/products/esx>, Accessed May 10 2011.

Voith, T. et al, Sept. 2010, “A Path Supervision Framework A Key for Service Monitoring in Infrastructure as a Service (IaaS) Platforms”, Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on , vol., no., pp.127-130.

Uhlig, R. et al., May 2005, “Intel virtualization technology”, Computer , vol.38, no.5, pp. 48-56.

Wei Chen et al., Nov. 2008 , “A Novel Hardware Assisted Full Virtualization Technique”, Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for , vol., no., pp.1292-1297.

W3C Member Submission, 2009, “Web Application Description Language”, available at: < http://www.w3.org/Submission/wadl/>. Accessed May 20 2011.

Xianmin Wei, Sept. 2011, “Application of Server Virtualization Technology in Enterprise Information”, Internet Computing & Information Services (ICICIS), 2011 International Conference on , vol., no., pp.25-28.

Xianxian Li, May 2011, “VMInsight: Hardware Virtualization-Based Process Security Monitoring System”, Network Computing and Information Security (NCIS), 2011 International Conference on , vol.1, no., pp.62-66, 14-15

Yang Gao, Yusi Ding, Hongyu Zhang, 20-22 Sept. 2009, “Multi-Objective Optimization for Dynamic Job-Shop Scheduling in Manufacturing Grid”, Management and Service Science, 2009. MASS '09. International Conference on , vol., no., pp.1-4.

Y. A. Zomaya and Y. H. Teh, Sept.2001, “Observation on using genetic algorithms for dynamic load-balancing”, IEEE Transaction on parallel and distributed systems, Vol. 12, No. 9.

Yudong Guo et al., May 2010, “A cooperative model virtual-machine monitor based on multi-core platform”, Future Computer and Communication (ICFCC), 2010 2nd International Conference on , vol.1, no., pp.V1-802-V1-807.

Zeadally, S. et al.., Dec. 2011, “Internet Protocol Television (IPTV): Architecture, Trends, and Challenges”, Systems Journal, IEEE , vol.5, no.4, pp.518-527.

Zehua Zhang and Xuejie Zhang, Nov. 2009, “Realization of open cloud computing federation based on mobile agent”, Intelligent Computing and Intelligent Systems, 2009, ICIS 2009, IEEE International Conference on , vol.3, no., pp.642-646.

Zhengxiong Hou et al., Sept. 2010, “ASAAS: Application Software as a Service for High Performance Cloud Computing”, High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on , vol., no., pp.156-163.

Zhen Ye et al., 2011, “Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing”, 322-334.