An Infrastructure for Robotic Applications as Cloud Computing Services

Carla Mouradian

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science
at Concordia University

Montréal, Québec, Canada

February 2014

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Carla Mouradian

Entitled: "An Infrastructure for Robotic Applications as Cloud Computing Services"

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

Master of Applied Science

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

Signed by the final examining committee:

____ Chair
Dr. M. Mehmet-Ali

____ Examiner, External
Dr. O. Ormandjieva (CSE) To the Program

____ Examiner
Dr. A. Agarwal

____ Supervisor
Dr. R. Glitho

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

____________20_____ ____
Dr. C. W. Trueman
Interim Dean, Faculty of Engineering and
Computer Science

ABSTRACT

An Infrastructure for Robotic Applications as Cloud Computing Services

Carla Mouradian

Robotic applications are becoming ubiquitous. They are widely used in several areas (e.g., healthcare, disaster management, and manufacturing). However, their provisioning still faces several challenges such as cost efficiency. Cloud computing is an emerging paradigm that may aid in tackling these challenges. It has three main facets: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Virtualization is a technique that allows the abstraction of actual physical computing resources into logical units; it enables efficient usage of resources by multiple users. Its role is a key to resource efficiency. Virtualization can be performed at both node and network level.

This thesis focuses on the IaaS aspects of robotic applications as cloud computing services. It starts by defining a set of requirements on the infrastructure for cost efficient robotic applications provisioning. It then reviews the state of the art. After pinpointing the shortcoming of the state of the art, it proposes an architecture that enables cost efficiency through virtualization and dynamic task delegation to robots, including robots that might belong to other clouds. Overlays and RESTful Web services are used as cornerstones. The virtualization in the IaaS is achieved by providing a coalition formation algorithm, which is the cooperation between several robots to perform a task that either cannot be solved individually or can be solved more efficiently as a group. Forming the effective coalitions is another big challenge. We adapted heuristic-based Multi Objective- Particle Swarm Optimization (MO-PSO) algorithm to solve this specific problem.

As a proof of concept, a prototype is built using LEGO Mindstorms NXT as the robotic platform, and JXTA as the overlay middleware and the prototype architecture is presented along with the implemented scenario (i.e., wildfire suppression). Performance measurements have also been made to evaluate viability. To evaluate the effectiveness of our algorithm, WEBOTS simulation software is used.

Acknowledgments

I would like to express my sincere thanks to Dr. Roch Glitho, my supervisor who helped me make this master of thesis possible. I thank Dr. Glitho for his continuous assistance, guidance and insightful advices over the last one year. I would also like to thank Dr. Fatna Belqasmi for her indispensable advices and support on different aspects of my project.

I would like to take this opportunity to thank my colleagues at Concordia's lab for their help, cooperation, and encouragement. I would like to express my deepest appreciation to Fatima Zahra Errounda for her continual support and for accompanying me on this journey. I am truly lucky to count you amongst my friends.

I am grateful to Dr. Agarwal and Dr. Ormandjieva for serving as members of my thesis committee.

I am grateful to Dr. Roch Glitho and Concordia University for their financial support and for giving me the opportunity to work in research and follow my dreams.

And finally, massive thanks to my husband for his support and sacrifice, thanks to my family for believing in me, and thanks to my friends who have showed me unceasing encouragement and support.

Table of Contents

List of Figures

List of Tables

Acronyms and abbreviations

NIST	National Institute of Standards and Technology
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
ISO	International Organization for Standardization
WSN	Wireless Sensor Networks
MOPSO	Multi-Objective Particle Swarm Optimization
SLA	Service-Level Agreement
VM	Virtual Machines
HaaS	Hardware as a Service
OS	Operating System
VMM	Virtual Machine Monitor
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
UGV	Unmanned Ground Vehicles

UAV	Unmanned Aerial Vehicles
CT	Computed Tomography
USAF	United States Air Force
CIA	Central Intelligence Agency
SOA	Service Oriented Architectures
SOAP	Simple Object Access Protocol
MRDS	Microsoft Robotic Development Studio
DSS	Decentralized Software Services
WSDL	Web Service Definition Language
XML	Extensible Mark-up Language
RTOS	Real-time Operating System
GPSO	General Purpose Operating System
CPU	Central processing unit
HTTP	Hypertext Transfer Protocol
ROS	Robot Operating System
RSNP	Robot Service Network Protocol
NP	Nondeterministic Polynomial Time

GA	Genetic Algorithm
NSGA	Non-Dominated Sorting Genetic Algorithm
SPEA	Strength Pareto Evolutionary Algorithm
PTCFA	Polynomial Time Coalition Formation Algorithm
OCFA	Optimal Coalition Formation Algorithm
PSO	Particle Swarm Optimization
PAES	Pareto Archived Evolution Strategy
P2P	Peer-to-Peer
REST	REpresentational State Transfer
JSON	JavaScript Object Notation
URI	Uniform Resource Identifier
ACO	Ant Colony Optimization
API	Application Programming Interface
VPL	Visual Programming Language
HTML	Hyper Text Mark-up Language
GAE	Google Apps Engine
URL	Uniform Resource Locator

Chapter 1

1 Introduction

This chapter starts with definitions for the key concepts related to our research such as cloud computing, robots, and virtualization. Then the motivation, the problem statement, and the thesis contributions are discussed. The last section introduces the thesis organization.

1.1 Definitions

In the following subsections we will give some definitions which are relevant to this thesis research domain. The definitions include Cloud Computing definition, Robots definition, and Virtualization definition.

1.1.1 Cloud Computing

There are several definitions for cloud computing, according to National Institute of Standards and Technology (NIST), "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction*". These resources can be networks, servers, storage, applications, and services [1]. Another definition of the cloud concept is, clouds are "*large pool of easily usable and accessible virtualized resources that can be dynamically reconfigured to adjust to a variable load (scale), allowing for an optimum resources utilization"* [2].

Cloud computing has three key facets: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The IaaS is the actual dynamic pool of physical and virtualized computing resources used by applications. The PaaS is built on the infrastructure layer and it is the middle bridge between hardware and application. It provides the software platform to develop and deploy applications. The SaaS is another alternative for the applications running on PC; it is the highest level of the hierarchy and consists of the actual cloud applications. Users can use these applications on pay-per-use basis.

1.1.2 Robots

According to International Organization for Standardization (ISO) 8373 [3], a robot is an actuated programmable mechanism that can perform intended tasks by moving in its environment. This definition combines both industrial and service robots. ISO first defined the robot as industrial robot: automatically controlled, reprogrammable multipurpose manipulator, programmable in three or more axes which may be either fixed in place or mobile for use in industrial automation applications. Then by the time that service robots emerged at the markets, they defined service robot as a robot that performs useful tasks for humans or equipment.

Robots can be used in a plurality of applications, in industry they can be used in environments that are dangerous for humans, such as search and rescue operations in disaster management and wildfire suppression operations in forests. In healthcare, they can be used in assisting nurses in taking care of patients, and they also can be used in assisting doctors in performing precise and complicated surgeries.

1.1.3 Virtualization

Virtualization allows the abstraction of actual physical computing resources into logical units, enabling their efficient usage by multiple independent users [4]. Its role is a key to resource efficiency. Virtualization can be performed at both node and network levels.

In this thesis we define robot Node Level Virtualization as the mechanisms that enable multiple applications to reside in and run concurrently on a single robot, analogous to the definition given in reference [5] for wireless sensor networks (WSN). On the other hand, we define robot Network Level Virtualization as the dynamic formation of subsets of robot nodes, with each subset dedicated to a certain application at a given time. This is also analogous to the definitions used in the WSN world [6]. The thesis deals only with Network Level Virtualization.

1.2 Motivation and Problem Statement

Robotic applications are ubiquities; they are used widely in different domains, but provisioning them as cloud computing services in cost efficient manner is a difficult task. For example in dynamic environments, there is no prediction of the size and the location of the event that may happen (e.g. wildfire); also some tasks that either cannot be solved individually or can be solved more efficiently as a group may need the collaboration between several robots, forming and dedicating the effective coalition dynamically with the correct number and capabilities of the robots is very critical. Furthermore, in some cases the capabilities and the number of the robots belonging to one cloud may not be sufficient for a given task which results in not completing a task or completing it in a non-efficient manner.

Delegating some tasks to robots belonging to other clouds can help in finding the most appropriate robots for a given task, since they may have the required capabilities. Also Network Level Virtualization can help in dynamic formation and dedication of subsets of robots for a task. By delegating some tasks to other clouds and performing Network Level Virtualization using an appropriate algorithm, cost efficient robotic application provisioning at the infrastructure level can be achieved.

1.3 Thesis Contributions

The thesis contributions are as follows:

- Set of requirements on the infrastructure for cost efficient robotic applications provisioning as cloud computing services.
- Set of requirements on the algorithm to perform the effective coalition for each task as part of the robot Network Level Virtualization performed by the infrastructure.
- Review of the state of the art with an evaluation based on our sets of requirements.
- Architecture for an infrastructure that enables cost efficient robotic application provisioning.
- Adaptation of heuristic-based Multi Objective- Particle Swarm Optimization (MOPSO) algorithm to form the effective coalition as part of the robot network level virtualization performed by the infrastructure.
- Implementation architecture, a proof of concept prototype, and performance evaluation.

- A simulation using WEBOTS software.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 presents the background concepts about to the key concepts used in this thesis.

Chapter 3 introduces the scenarios and the requirements derived from these scenarios, followed by the evaluation of the state of the art based on the requirements.

Chapter 4 describes the proposed architecture for an infrastructure that enables cost efficient robotic application provisioning. It discusses the architectural principles, the functional entities and the interfaces.

Chapter 5 describes the algorithm used for Network Level Virtualization. It discusses the assumptions and presents the algorithm in detail along with the Pseudocode.

Chapter 6 describes the implementation architecture and technologies used for the proof-of-concept prototype. Then it discusses the performance measurements done to evaluate the architecture and discusses the simulation results to evaluate the algorithm.

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

Chapter 2

2 Background

This chapter presents the background information that is relevant to this thesis research domain. The background information covers three topics: Cloud Computing, Virtualization, and Robotic Applications.

2.1 Cloud Computing

This subsection present a general overview of cloud computing. We start with a brief definition of cloud computing. Then, we present the cloud layers, followed by a subsection that discusses the different cloud types. Finally, we explain the cloud computing advantages.

2.1.1 Cloud Computing Definition

Cloud computing recently has taken attention and reached popularity. It has been defined using different definitions [2]. The main reason for existence of different definitions is that cloud computing is not a new technology, but rather a new operations model that brings together set of existing technologies (such as virtualization and utility-based pricing) to run business in a different way [7]. NIST has defined cloud computing as "*model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" [1].

Yet, in [2], after gathering most of available definition they tried to find an integrative definition, they defined cloud computing as a "*large pool of easily usable and accessible virtualized resources that can be dynamically reconfigured to adjust to a variable load (scale), allowing for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the infrastructure provider by means of customized SLAs*".

2.1.2 Cloud Layers

Cloud computing services are divided into three layers as shown in Figure2-1: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

2.1.2.1 Infrastructure-as-a-Service (IaaS)

This IaaS provides virtualized resources on demand, such as computation, storage, and communication [8]. This layer is the lowest level of abstraction where users have access to the underlying infrastructure through the Virtual Machines (VMs). Users at this level request computing resources such as processing power, memory, and storage from the IaaS provider and use the resources to deploy and run their applications [9]. Sometimes the IaaS is also called Hardware-as-a-Service (HaaS). Examples of IaaS: Amazon Web Services' EC2, and S3.

2.1.2.2 Platform-as-a-Service (PaaS)

This layer provides a higher level of abstraction to make a cloud easily programmable. It offers a platform on which developers can create, deploy and develop applications without knowing how much memory or processors need their applications [8]. The applications are developed using different programming languages. Users do not have access or control over the underlying infrastructure [9]. Example of PaaS is Google App Engine, which lets the developer to run a web application on Google's infrastructure [10] and Microsoft Azure.

2.1.2.3 Software-as-a-Service (SaaS)

This is the highest level of cloud architecture. Services provided by this layer can be accessed by end users through Web portals. It is an alternative to applications that run locally on a PC. Users do not need to install or run any software; they just need a web-browser to access software developed by others [8]. Just like the PaaS, users do not have control over the underlying infrastructure. Example of Software-as-a-Service is word processor and Google Docs.

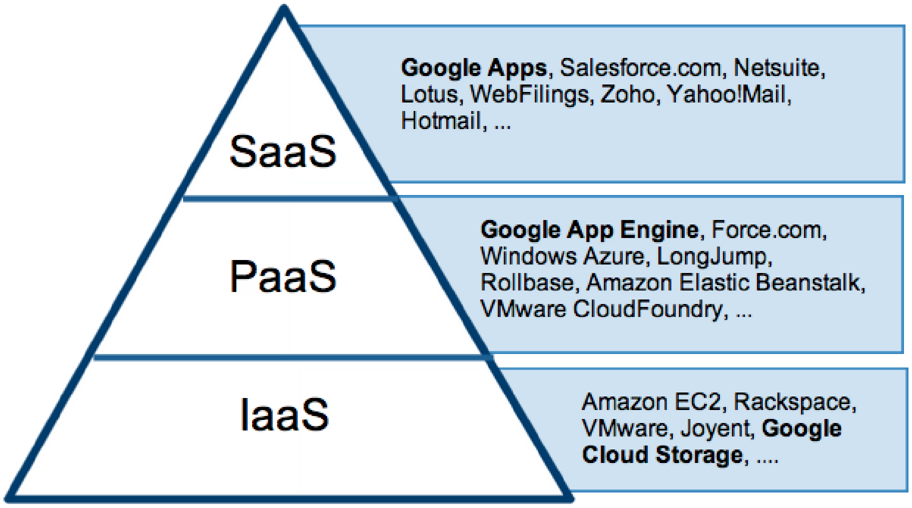

SaaS
Google Apps, Salesforce.com, Netsuite, Lotus, WebFilings, Zoho, Yahoo!Mail, Hotmail, ...
PaaS
Google App Engine, Force.com, Windows Azure, LongJump, Rollbase, Amazon Elastic Beanstalk, VMware CloudFoundry, ...
IaaS
Amazon EC2, Rackspace, VMware, Joyent, **Google Cloud Storage**, ...

Figure 2-1 Cloud Computing Layers [11]

2.1.3 Types of Clouds

There are different types of clouds depending on who owns and uses them. Cloud can be classified as public cloud, private cloud, community cloud, or hybrid cloud [9][7].

- **Private Cloud:** A private cloud is used by a specific organization; it is not available for public. It allows users to interact with the local data centers while having the same advantages of public cloud. This type of clouds provides performance, reliability and security [7].

- **Public Cloud:** A public cloud is available for the public as pay-per-use manner. It is usually owned by big corporations such as Amazon, Google, or Microsoft. This type of clouds lacks some control over data, network and security settings [7].

- **Community Cloud:** A community cloud is a cloud shared by several organizations, and it is setup for their specific requirements.

- **Hybrid Cloud:** A hybrid cloud is combination of public, private, and community cloud. It combines the advantage of both public and private clouds. It also allows cloud bursting to take place, which means a private cloud can burst-out to a public cloud when it requires more resources [9]. The main benefit of hybrid clouds is that it provides more flexibility than both public and private clouds [7].

2.1.4 Cloud Computing Advantages and Features

Cloud computing has several advantages and features. Some of these features include:

- **Per-Usage Billing:** The pricing in cloud computing is flexible and it is on pay-per-use basis. It may vary based on the services given, a provider for example can charge a user based on the usage of a virtual machine per hour, or based on the number of clients using the service [7]. Resources are released and the user doesn't pay anything as soon as the resources are not needed anymore.

- **On-demand Self-Service:** The resources can be acquired at any time whenever the user needs, without the need of human interaction between the user and cloud provider [9].

- **Elasticity:** Cloud computing gives the impression of having infinite computing resources on demand; therefore it provides resources to users in any quantity at any time [8]. A user can acquire more resources whenever he needs, and these resources are released once they are not required anymore [9].

- **Scalability:** Cloud providers provide large amount of resources. A cloud can easily expand its services to a large scales based on the user needs. It can handle rapid increase of service demands.

- **Resource Pooling:** It is also called multi-tenancy, where resources are pooled so they can be shared by multiple users. For example a physical server can host several Virtual machines belonging to different users [9].

- **Easy Access:** A cloud is generally web-based. This makes the cloud accessible easily by users through variety of devices**.**

2.2 Virtualization

In this subsection we first define virtualization, and then we discuss different types of virtualization (e.g., Network Level Virtualization, Node Level Virtualization). After that we discuss the different approaches of virtualization including full virtualization, para-virtualization, etc. And finally, briefly we give some of the benefits of virtualization.

2.2.1 Definition of Virtualization

Virtualization allows reduction in cost and complexity through the abstraction or emulation of actual physical computing resources into logical, shareable units, enabling their efficient usage by multiple independent users [4]. Its role is a key to resource efficiency. In other words, virtualization is the concept of having different logical views of a physical machine, each of which can be used to interact with a user simultaneously. Virtualization increases the utilization rate of the underlying physical hardware.

Before virtualization, a physical machine can have one operating system which supports one or more application programs, while using virtualization a physical machine may have several virtual machines, each virtual machine may be running a different operating system. Any problem or failure in one of the virtual machines doesn’t affect the other virtual machines [12]. A virtual machine is the software implementation of a machine (i.e., computer) that executes program just like a physical machine. The Virtual Machine Monitor or the Hypervisor is the software that creates virtual machine environment in a machine. The guest operating system is the operating system that runs on the virtual machine. The host operating system is the operating system that runs on the physical machine.

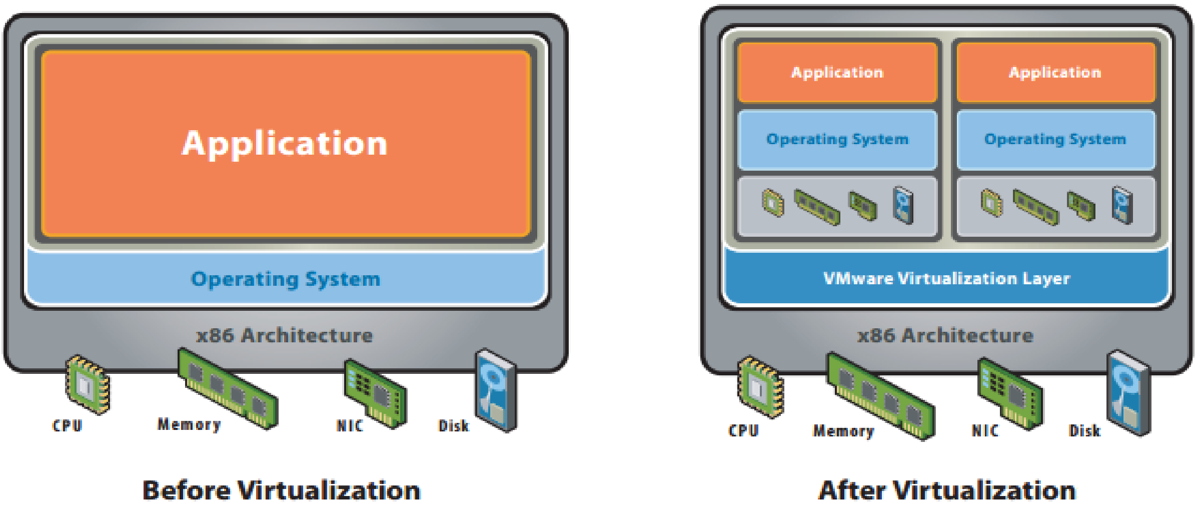


Figure 2-2 Before and After Virtualization [12]

Figure 2-2, shows the difference before virtualization and after. Before virtualization:

- Single OS image per machine.
- Software and hardware tightly coupled.
- Running multiple applications on same machine often creates conflict.
- Underutilized resources.
- Inflexible and costly infrastructure.

While after Virtualization:

- Hardware-independence of operating system and applications.
- Virtual machines can be provisioned to any system.

- Can manage OS and application as a single unit by encapsulating them into virtual machines.

2.2.2 Virtual Machine Monitor or Hypervisor Types

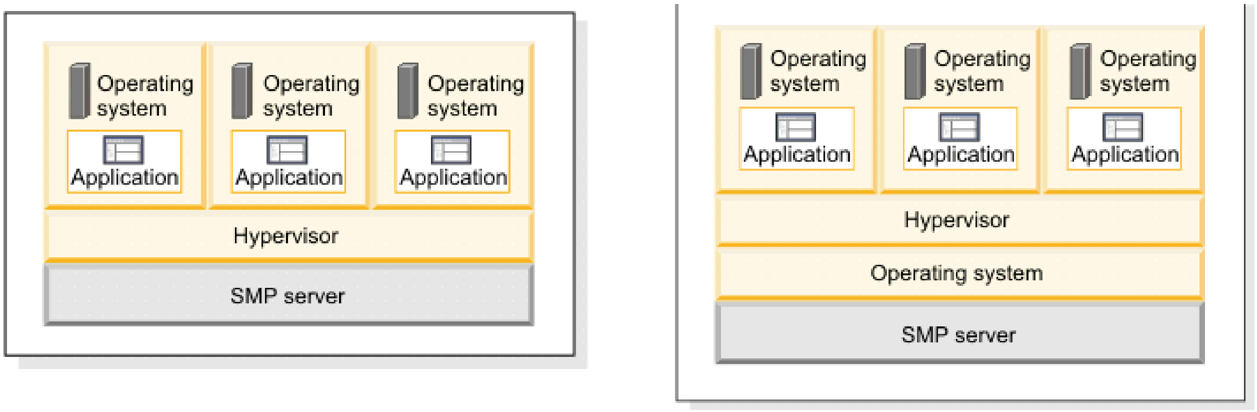
Virtual Machine Monitor or the Hypervisor is a layer of software that runs on hardware and allows multiple operating systems to run and share the underlying hardware or physical host. The VMM presents to each gust OS a set of virtual platform interfaces that constitutes a virtual machine [13]. The VMM provides abstraction of the virtual machine to guest OSes and takes complete control of virtualized resources [14].

There are two types of hypervisor [15]:

- **Type 1:** Hypervisor runs directly on the hardware to control the hardware and monitors the Guest OSes. This type can achieve high performance and high virtualization efficiency.

- **Type 2:** Hypervisor runs on the host operating system that provides virtualization services and monitors the Guest OSes. This type is characterized by ease of construction and installation.

Figure 2-3 shows the two types of hypervisor



a. Type 1 Hypervisor

b. Type 2 Hypervisor

Figure 2-3 The Two Types of Hypervisor [15]

2.2.3 Different Types of Virtualization

There are different types of Virtualization: Network Virtualization, Node Level Virtualization, Storage Virtualization, Desktop Virtualization, Database Virtualization, etc. In this thesis we define robot virtualization as Node and Network Level virtualization, therefore we will give background information on Node and Network Level virtualization.

2.2.3.1 Network Virtualization

Network virtualization is allowing the coexistence of multiple virtual networks on the same physical substrate. Each virtual network in Network Virtualization is a collection of virtual nodes and virtual links. In Network Virtualization there are the Infrastructure Providers and the Service Providers. The Infrastructure Provider is responsible of the physical infrastructure, and the Service Provider provides the virtual networks [16].

Network virtualization can be classified into three types, Virtual Local Area Network (VLAN), Virtual Private Network (VPN), and the Overlay Networks. In VLANs group of computers appears as if they are connected to the same physical network, while they are actually located on different geographical areas. The main benefit is that if a computer is moved to another location, it can still stay on the same VLAN. The VPN extends the private network across the public network. It uses the internet to provide remote offices or users with secure access to their organization’s network. Lastly, the Overlay Network is a virtual network which is created on top of a physical network, without making any changes to the underlying network. Figure 2-4 shows Network Level Virtualization, where there is the physical network on the bottom, and virtual networks created on top of it.

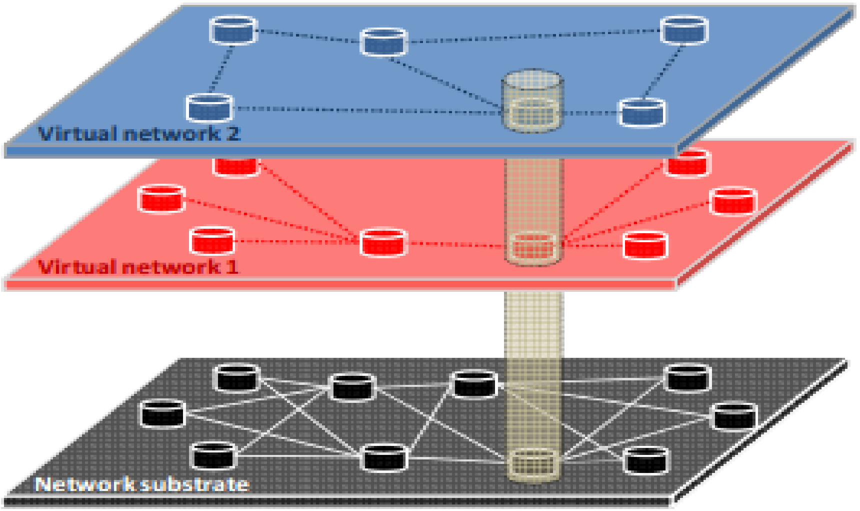


Figure 2-4 Network Virtualization Model [17]

2.2.3.2 Node-Level Virtualization

Node virtualization is achieved by isolation and partitioning of hardware resources. The physical resources of physical node (CPU, memory, storage capacity, and link bandwidth) is partitioned into slices and allocated to the virtual nodes based on their requirements [17].

2.2.4 Robot Virtualization

In this thesis we define robot virtualization in both Node and Network Level. Robot Node Level Virtualization is defined as the mechanisms that enable multiple applications to reside in and run concurrently on a single robot, analogous to the definition given in reference [5] for wireless sensor networks (WSN). It allows the robot to become multi-purpose device which is capable of executing more than one application at the same time. The execution of new applications is done without disturbing the old ones.

On the other hand, we define robot Network Level Virtualization as the dynamic formation of subsets of robot nodes, with each subset dedicated to a certain application at a given time. This is also analogous to the definitions used in the WSN world [6]. Enabling the dynamic formation of subset of robot nodes dedicated to specific application ensures the resource efficiency. For example in dynamic events, since the event can move around, it is better to use only some of the robots by creating virtual network of the robots and to include or exclude robots from that virtual network depending on the event movement. The other robots can be available for another application.

2.2.5 Levels of Virtualization

There are three different types of virtualization concept [18]:

- Full Virtualization
- Para-Virtualization
- Hardware-Assisted Virtualization

2.2.5.1 Full Virtualization

In full virtualization a guest OS is fully decoupled from the underlying hardware by the virtualization layer. It is unaware that it is being virtualized hence doesn't need any modification and can be installed above the hypervisor. The hypervisor provides hardware resources to each guest OS [19]. In full virtualization whenever the guest OS calls a sensitive instruction, the hypervisor emulates the instructions behavior and return the proper result.

Full virtualization provides best isolation and security for virtual machines and simplifies migration and portability as the same guest OS instance can run virtualized or on native hardware. Examples of full virtualization products are VMware's virtualization products and Microsoft Virtual Server [18].

2.2.5.2 Para-Virtualization

In Para-virtualization the OS needs to be modified for the hypervisor. It refers to communication between the guest OS and the hypervisor to improve performance and efficiency [18]. In Para-virtualization the guest OS is acting like a normal user application running on a regular OS, the only difference is that the guest OS is running on the hypervisor. In Para-virtualization the guest

OS is modified in order to make hypercalls instead of containing sensitive instructions. Para-Virtualization is much easier to implement than full virtualization. The open source Xen project is an example of para-virtualization (using a modified Linux kernel) [18].

2.2.5.3 Hardware assisted virtualization

Hardware-assisted Virtualization improves the fundamental flexibility and robustness of traditional software-based virtualization solutions; it enables efficient full virtualization by using help from hardware capabilities, primarily from the host processors. Privileged and sensitive calls are set to automatically trap to the hypervisor, removing the need for either binary translation or para-virtualization. The guest state is stored in Virtual Machine Control Structures or Virtual Machine Control Blocks. Processors with these hardware assist features (Intel VT and AMD-V) became available in 2006, so only newer systems contain these features [18].

2.2.6 Benefits of Virtualization

Virtualization has several benefits and advantages; in the following lines we'll summarize some of its benefits. One advantage is, since the virtualization layer abstracts the resources of the underlying hardware and presents them in a standardized way to the virtual machine's operating system and applications, any virtual machine can be run on any server in the data center. Also, instead of having the complexity of supporting multiple applications sharing the same operating system, the applications are isolated on dedicated virtual machines where the number of applications running on single virtual machine is limited by the machine's available resources

[20]. Another advantage is that if a single virtual machine fails, the other virtual machines running on the same physical machine are not affected and can continue running.

2.3 Robots and Robotic Applications

In this subsection first we define robots. Then we present robotic applications in different domains, such as Search and Rescue, Emergency, Logistic, and Military domains.

2.3.1 Robots

According to International Organization for Standardization (ISO) 8373 [3], a robot is an actuated programmable mechanism that can perform intended tasks by moving in its environment. This definition combines both industrial and service robots. ISO first defined the robot as industrial robot: automatically controlled, reprogrammable multipurpose manipulator, programmable in three or more axes which may be either fixed in place or mobile for use in industrial automation applications. Then by the time that service robots emerged at the markets, they defined service robot as a robot that performs useful tasks for humans or equipment.

2.3.2 Robotic Applications

Robots nowadays are present in different domains, in industry they are being used in environments that are dangerous for humans, such as search and rescue operations in disaster management or wildfire suppression operations in forests. When an earthquake occurs in a very populated area, rescue teams must be employed to save victims in dangerous locations. People lying buried under the rubbles wait for an immediate rescue, which usually takes hours or sometimes even days. The rescue teams may need to operate in dangerous environments risking their lives; therefore robot supported systems are used in hazardous areas to perform the search operations in more efficient and quick manner. They help in detecting victims in large disaster areas [21]. The rescuers usually enter areas with unstable structures, they expose themselves to hazards which threat their lives, therefore robots are used to help the human rescuers, they can enter these dangerous and structurally unstable environments instead of rescuers. Robots can search survivors using special sensors; they can also carry food and medication to the victims. These robots can be ground robots called Unmanned Ground Vehicles (UGVs) or aerial, Unmanned Aerial Vehicles (UAVs). They can bring information to an operator for analysis to help in the situation assessment [22].

Robots are also used in other emergency events, e.g. fire; they can also be ground or areal. They can be used to gather data using sensors that detects the presence of chemicals, cameras, and laser [23]. Figure 2-5 shows some examples of search and rescue robots.

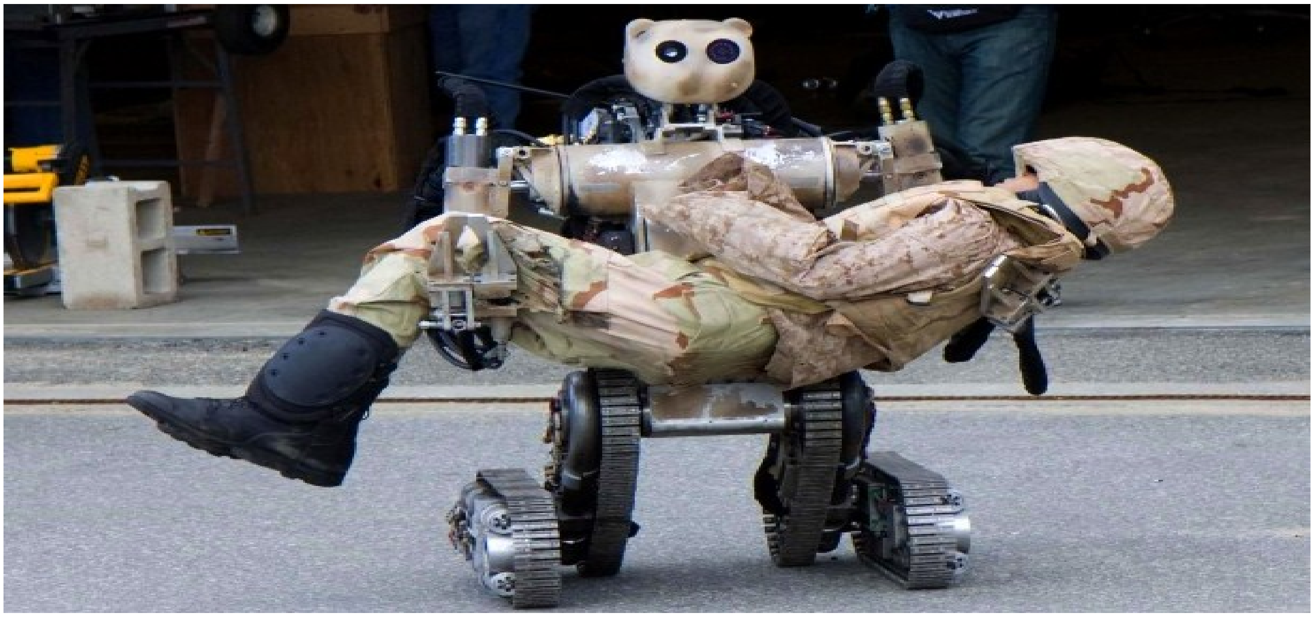


Figure 2-5 Search and Rescue Robots [24]

Another area where robots exist is logistic systems in hospitals. Robots in logistic systems are used to transport items such as medicines, medical devices, food, documents, etc. Using robots in these systems reduces the annual cost of the hospital and it has several advantages, e.g., robots never get sick, they don’t need holidays, they can work in weekends, they are predictable and don’t make human mistakes, they can work 24 hours 7 days per week, etc. [25].

Figure 2-6 shows an example of commercial system which is installed successfully in the hospitals, it is called Helpmate. A person can place the goods on robot’s platform, selects a destination using a map displayed on robot, and the robot starts moving toward the destination on preplanned path, it can avoid obstacles or wait for an obstacle to be removed, finally the person that receives the goods confirm that the mission is finished [25][26].

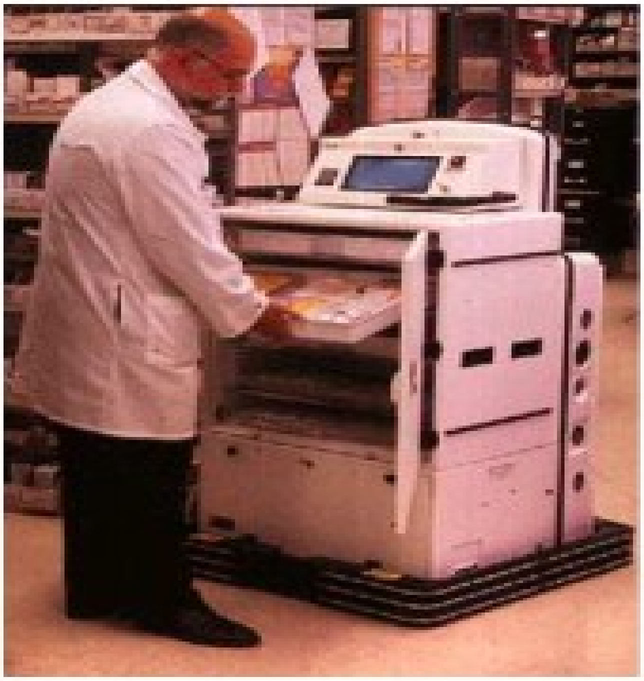


Figure 2-6 Helpmate Robot [25]

Robots can also be used in human surgeries; the first time a robot was used in such a case was in 1985 for brain biopsy using a computed tomography (CT) image and a stereotactic frame. The robot defined the trajectory for a biopsy by keeping the probe oriented toward the biopsy target. Then in 1992 another robot called Minerva robot was designed to direct tools into the brain under real-time CT guidance which allows tracking the target even if the brain tissue swells. The Da Vinci robot which is a teleoperated system is another surgical robot with over a thousand systems installed worldwide. It was offered with three arms to hold two tools and an endoscope [27]. Figure 2-7 shows the Da Vinci robot.

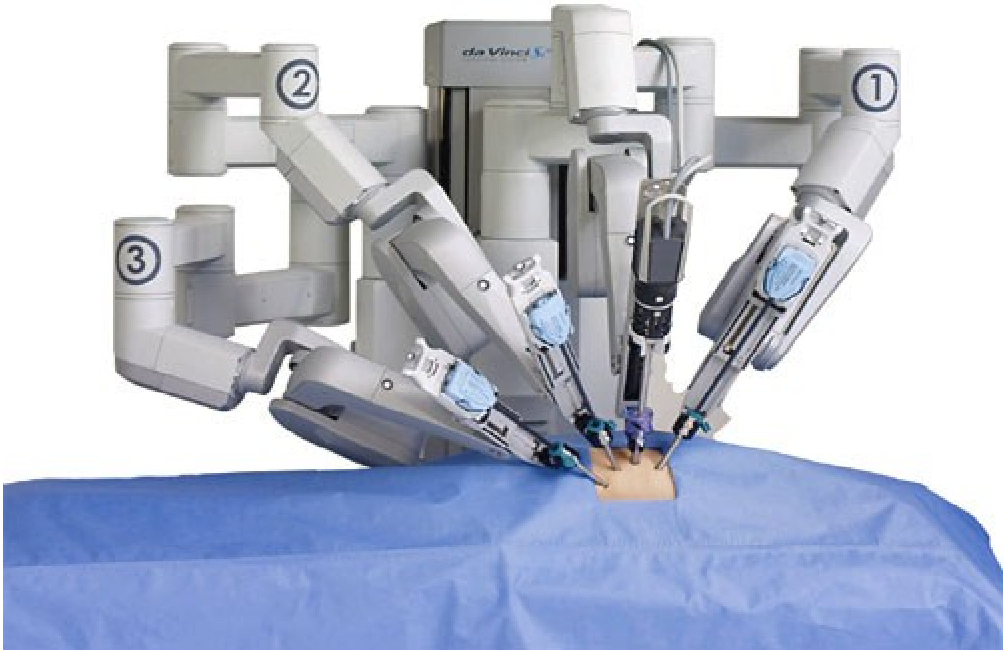


Figure 2-7 Da Vinci [27]

Robots also exist in military. The military robot will be able to substitute the real human soldier in the battle field [28]. India has got its first military robot called "Daksh" which is developed by Defense Research and Development Organization. This robot can climb stairs to reach hazardous materials; it can lift a suspicious object and scan it using its arm and X-Ray device. Also if the object is a bomb, Daksh can defuse it with its water jet disrupter [29]. Another military robot is MQ-1 Predator drone which is a UAV; it was used by United States Air Force (USAF) and CIA. It carries cameras and other sensors, and it can fire missiles [30]. Figure 2-8 and 2-9 shows the MQ-1 and Daksh robot.

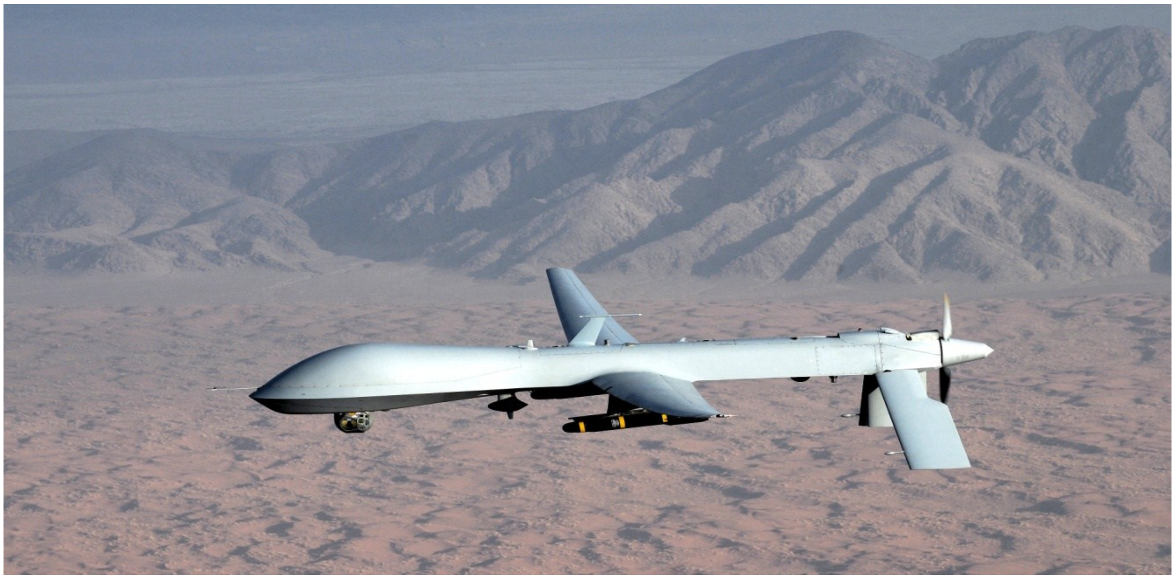


Figure 2-8 MQ-1 Drone [30]

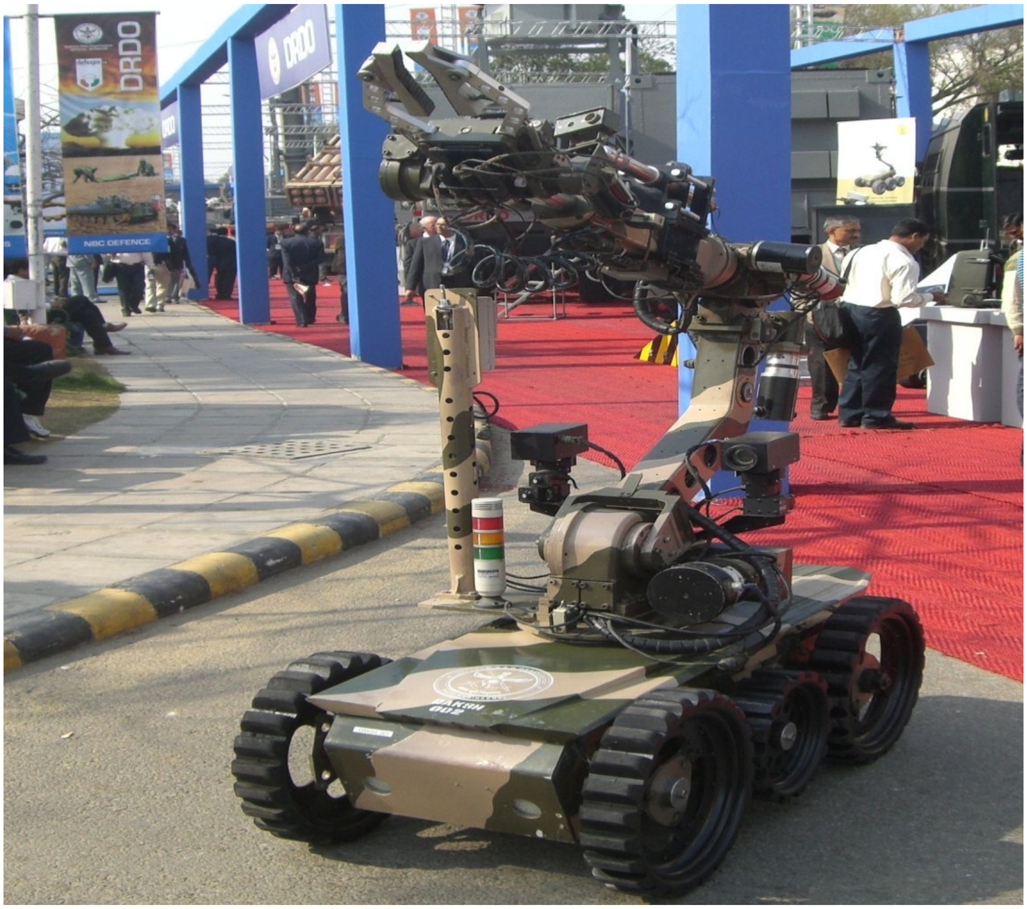


Figure 2-9 Daksh Robot [29]

2.4 Chapter Summary

In this chapter we discussed the background concepts that are related to this thesis, first we introduced the concept of cloud computing by giving several definition and discussing cloud types, architecture, and advantages. Then we followed by discussing the virtualization concept, explaining the hypervisor with its types, the different types and levels of virtualization, the robot virtualization, and some of virtualization benefits. Finally we discussed robots and robotic applications domains including disaster management, healthcare, and military domains.

Chapter 3

3 Scenarios, Requirements and State of the Art Evaluation

This chapter includes three sections. In the first section we discuss the scenarios which include three motivating scenarios: Wildfire Suppression scenario, Subway Degradation Detection scenario, and Search and Rescue Robots scenario. In the second section we derive the requirements from these scenarios and divide them into three groups: general requirements on the IaaS for cost-efficient robotic application provisioning, specific requirements on the robots hosted by the IaaS, and specific requirements for Network Level Virtualization algorithm performed by the IaaS. Next, we review and evaluate the state of the art based on our sets of requirements. Finally we summarize the chapter.

3.1 Scenarios

In this subsection, we will provide three motivating scenarios: Wildfire Suppression Scenario, Subway Degradation Detection Scenario, and Search and Rescue Robots Scenario.

3.1.1 Wildfire Suppression Scenario

The first scenario is Wildfire Suppression. We consider a Wildfire Suppression Robotic Application that detects and suppresses wildfires using a fleet of heterogeneous robots deployed in the forest. If a fire is detected, the application evaluates its intensity and the rate of spread, and deploys the most appropriate robots to extinguish it.

The robots have different capabilities. Some are equipped with cameras which allow them to supervise the fire area and send notification if the fire is spreading. Other robots are equipped with arms that help them grab the extinguishers and suppress the fire using water and foam. Other robots can detect obstacles and remove them. We assume that these robots do not necessarily belong to the same business entity. In a cloud environment, this means there are potentially several IaaSs owned by different business entities that host robots with varying capabilities. Using these robots in a cost-efficient manner is of paramount importance.

3.1.2 Subway Degradation Detection Scenario

The second scenario is a subway infrastructure that shows signs of degradation. Mobile robots are deployed in the subway to detect cracks and corrosion. The robots have different capabilities using different sensors and actuators. Some are equipped with ultrasonic sensors which allow the robots to avoid obstacles, some with cameras which allow them to take real-time videos and others are equipped with arms to collect material samples.

Also some robots are equipped with more than one sensor, which allows them to perform more than one task. For example, a robot can provide video stream for the real-time video application and it can cover an area requested by the patrolling unit at the same time using different sensors for each application.

A group of robots can also be used by the patrolling application if the requested area is too large for one robot to cover. If real-time videos are requested at the same time the robots are covering an area by patrolling, one set of the same robots may be used to provide the requested information. We again assume that all robots do not necessarily belong to the same business entity, which means there are several IaaSs owned by different business entities that host robots with different capabilities. Here as well, it is critical to make the most efficient use of the robots.

3.1.3 Search and Rescue Robots Scenario

The third scenario is the Search and Rescue Robots scenario. Natural Disaster such as earthquakes, tsunamis, floods, etc., can lead to loss of human lives and destruction of their properties. Rescuing as many survivors as quickly as possible is a top priority. We will consider a search and rescue robotic application that searches and detects survivors and bodies under the rubble whenever there is a natural disaster. When a disaster occurs, the application evaluates the disaster area including the size and the location of narrow areas, risky areas and areas where there is more likely to be humans, etc.

Robots have different capabilities; some are equipped with sound sensors to detect voices of other sounds of possible human presence within the ruins. Others robots may be very small and can access locations where human beings or other machines cannot, e.g., snake robots. Some robots may carry thermal cameras that can detect body heat. Others can have cameras that search for colors distinctive from the gray dust that has blanketed the debris. And other robots may be able to climb ramps and overcome obstacles.

Some robots may be equipped with more than one sensor, for example a robot can have both a thermal camera and a sound sensor which allows the robot to search for body heat and at the same time detect human voices if present.

In addition, since natural disaster areas are usually large, a group of robots might be needed to search for human bodies, with each group using different sensors. Some may search for blood, others for human voices, yet others may search in very narrow areas, and another group may search in dangerous areas where there is a high risk of collapse.

Again we assume that all robots do not necessarily belong to the same business entity, meaning there are several IaaSs owned by different business entities that host robots with different capabilities. Here as well, it is critical to make a cost-efficient use of the robots.

3.2 The Requirements

We divide the requirements into three groups. The first group is a set of general requirements on the IaaS for cost-efficient robotic applications as cloud computing services, the second group is a set of specific requirements on the robots hosted by the IaaS, and the third set is specific requirements on the algorithm to perform the effective coalition for each task as part of the robot Network Level Virtualization performed by the IaaS.

Based on the scenarios we provided in the previous section (section 3.1), the robots do not belong to the same business entity, which means that there are several IaaSs owned by different business entities who host these robots. Therefore, we consider a system made up of an IaaS and the robots that are part of the IaaS. The IaaS receives a request for a task from a PaaS, and interacts with the robots (which are part of the IaaS) to send them the request.

3.2.1 General Requirements of the IaaS

The first requirement is scalability. The IaaS should scale, accommodate, and function well with any number of robots. The second requirement is that the IaaS should have an overall standardized technology for the interaction interfaces, which is the interface between the IaaS and the PaaS, and also the interface that allows the interaction between the IaaS and the robots that are hosted by the IaaS. For example, in the three scenarios the robots may support different interfaces depending on each of the robots' providers. However, the architecture should be able to communicate with the robots using a unified standard interface technology. Also the Wildfire Suppression application that sends a request to the IaaS to suppress the fire through the PaaS may use different technologies at the interface level. The IaaS should be able to communicate with the PaaS using standard interface technology. It is also the case for the Subway Degradation Application and the Search and Rescue Robots Application.

The third requirement is that the IaaS should provide isolation, which allows more than one robotic application to be run on the same IaaS. For example, two applications may run on the same IaaS using different sets of robots such as the Wildfire Suppression Application and Degradation Detection scenario. One set of robots may be assigned to go and suppress the fire somewhere in the forest and another set of robots can be used to cover an area by patrolling for the Subway Degradation Application. The execution of each application should be completely isolated from the execution of the other; they should not interfere with each other at all.

The fourth requirement is that the IaaS should be able to support heterogeneous robots. Because of the diversity of robots' vendors, the IaaS may contain robots belonging to different providers with each having its own interface and programming language. The IaaS should be able to support robots belonging to different vendors, which means the overall solution should be applicable to a wide variety of heterogeneous robots.

The fifth requirement is extensibility. The solution should take future growth into consideration; extensibility can be through adding new functional entities such as fault management which may enhance the IaaS's performance, or through modification of existing functional entities. Hence, the overall solution should be extensible in terms of adding new functionalities to the IaaS.

Another requirement is that the IaaS should be able to delegate tasks to robots that belong to other clouds. This is very important in some cases, such as when the capabilities or the number of the robots belonging to one cloud may not be sufficient for a given task which may result in incompletion of a task or completing it in non-efficient manner. For example, in the Search and Rescue Robots scenario, the local cloud may not have a sufficient number of tiny robots (e.g. Snake Robots) that can access narrow locations where human beings or other machines cannot access, therefore it can delegate some tasks to tiny robots belonging to other clouds; hence executing the task in more efficient manner.

The last requirement is that the IaaS should support Network Level Virtualization since we are dealing with dynamic environments. Also, some tasks that cannot be solved individually or can be solved more efficiently as a group may need the collaboration between several robots. In the Wildfire Suppression scenario for example, one robot cannot perform the task of suppressing the fire, there is a need for collaboration between the robots and therefore a group of robots should be formed and dedicated for the task “suppressing the fire” dynamically with the correct number and appropriate capabilities of robots.

3.2.2 Requirements of the Robots Hosted by the IaaS

Robots belonging to the IaaS should support Node Level Virtualization. This enables multiple applications to reside in and run concurrently on a single robot, enabling the robot to execute more than one application at the same time without any interference between these applications. In the Subway Degradation Detection scenario a group of same robots can be used to provide real-time videos and cover an area by patrolling using different sensors for each task, instead of having different robots for each of these tasks.

3.2.3 Requirements on the Network Level Virtualization Algorithm

The first requirement on the algorithm is that it should be multi-criteria, which means it should take into consideration static and dynamic criteria: Static criteria such as robots’ sensors and actuators, and dynamic criteria or criteria that may change over time, such as robots’ position, battery, quality of service characteristics, etc. The second requirement is that the algorithm should be able to optimize multiple objectives, which is minimizing or maximizing more than one objective function simultaneously in the presence of trade-offs between two or more conflicting objectives. For example, minimizing the time needed to suppress the fire and also at the same time minimizing the cost of robot deployment are two conflicting objectives.

The third requirement is that the algorithm should minimize as much as possible or eliminate the communication among the robots since the robots have limited communication capabilities. The fourth requirement is that the algorithm should be able to select a group of robots or a coalition of robots for a given task when one robot cannot do a given task individually. In the Wildfire Suppression scenario one robot cannot suppress a fire taking place in a forest; hence a group of robots is needed. Additionally, the algorithm should select the subset of robots for a given task using specific filtering and ranking methods or functions. These methods play a role in making other robots available for other tasks and minimize the number of robots on which the algorithm runs, which in turn reduces the response time of the algorithm.

Another requirement is that the algorithm should have a good response time, meaning the processing time of the algorithm should be minimized as much as possible. Even on powerful physical machines the convergence time of the algorithm should be minimized as much as possible. Some events are time critical and the time needed by the algorithm to give a result should be fast.

Also the algorithm should optimize the resources - in our case, the robots. In other words, the algorithm should minimize the number of robots in one group performing the same task; this will help in making robots available for other tasks. The algorithm should also minimize the cost of robot deployment to perform a given task to meet the cost agreed with the customer. The last requirement is that the algorithm should minimize the time needed to perform a given task by a group of robots. We can see there are trade-offs between the last three requirements. Minimizing the time needed to perform a task that needs more powerful robots also needs more robots in one group, which in turn means increasing the cost of robot deployment and increasing the number of robots in one group.

In this thesis, we will focus on the General Requirements except for the isolation of several applications running on the same IaaS, and we will also focus on the Requirements on the Network Level Virtualization Algorithm.

3.3 The State of the Art Review and Evaluation

In this section we review the state of the art for an infrastructure for robotic applications as cloud computing services. We categorize and review the state of the art in two sections: The first section is existing frameworks for robotic applications, which we evaluated using our requirements of the IaaS and requirements of the Robots hosted by the IaaS. The second section is existing algorithms to perform the effective coalition which we evaluated using our requirements on the Network Level Virtualization algorithm.

3.3.1 Frameworks for Robotic Applications

There are several studies that present the IaaS aspects of robotic applications as cloud computing services. The architecture proposed in [31] relies on SOA. To offer robotic applications as cloud computing services SOA principles are used. It decouples robots' sensing and actuating capabilities and the applications that use them by offering these capabilities as SOAP-based services. The main concept of this work is designing and implementing a robot or device to be an all-in-one SOA unit, meaning the robot as a service (RaaS) should have the complete functions of SOA which are a service provider, service broker, and a service client. This means each robot holds a repository of preloaded services. A client can deploy new services or remove some services from the robot. These services can also be shared with other robots. A client can also compose new applications or functionalities based on the services available at the robot. Finally a client can look up the services and applications in robot's directory. The authors developed a prototype of Robot as a Service to prove the concept using an Intel processor; and developed and deployed a maze application using Microsoft Robotic Developer Studio (MRDS) [32]. Since MRDS does not support Intel architecture, a mapping layer was implemented from the device drivers to Microsoft DSS (Decentralized Software Services) in MRDS as shown in Figure 3-1 to allow the maze application to monitor and control the robot sensors and actuators. For that purpose, an interface between the MRDS framework and the Intel platform was developed.

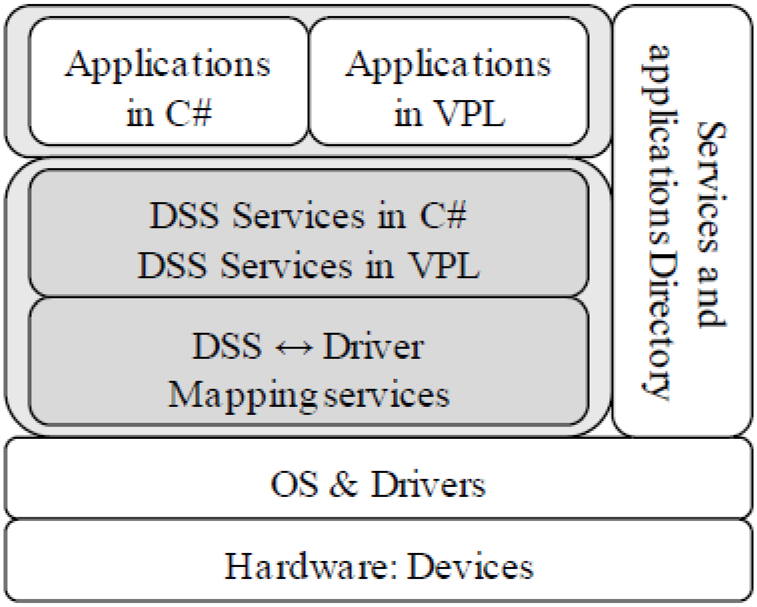


Figure 3-1 Device Driver Mapping Services [31]

In the proposed architecture in Figure 3-1, the communication between the RaaS and other services in the cloud is through standard interfaces, where Web Services Description Language (WSDL), which is an XML-based interface description language, is used. It does not discuss virtualization at Network Level nor at Node Level. However, an algorithm that guides the robot is provided. In this work delegating tasks to other clouds and the isolation of one robotic application from the other are not discussed. Also the solution that is provided in the paper is for one robot that they have designed, hence heterogeneity is not addressed. Extensibility is provided in terms of deploying new services to the robot or composing new applications to the robot and not adding new functional entities to the architecture. Finally, the scalability in terms of adding new robots is not discussed.

Chen et al. in [33] improved the architecture presented in [31] by proposing the architecture shown in Figure 3-2. They moved the directory of robotic application from the unit level to the network level. Also a mapping layer on top of the robots' infrastructure was introduced to map virtual robot objects to physical robots, thus the end users can request their desired robot service without knowing or considering what physical robots are actually assigned.

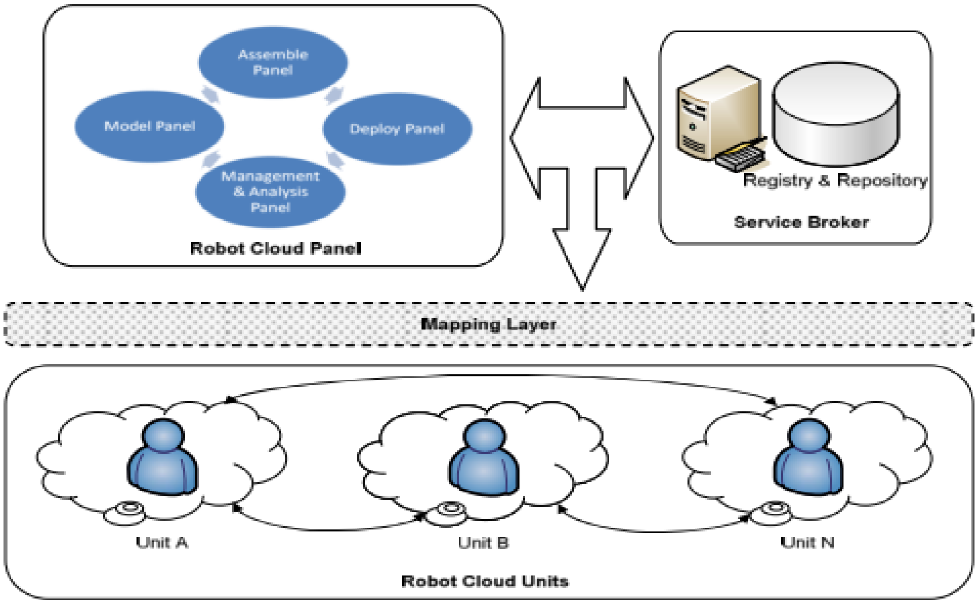


Figure 3-2 Robotic Cloud Architecture [33]

In this work Network Level Virtualization is provided by using Max-Heap algorithm for task execution, but Node Level Virtualization is not discussed. For the communication interfaces for the interaction between the robots and between robots and other parts of the system WSDL is used, which is a standard interface. Also, heterogeneity is catered; in their proposed architecture there are heterogeneous robots that provide different services and have different hardware devices and device drivers. The scalability in terms of adding new physical robots was addressed, but the extensibility in terms of adding new functional entities is not discussed. Finally, delegating some tasks to other clouds and the isolation of one robotic application from another is not discussed.

The work presented in [34] discusses the need of running both Real-time Operating System (RTOS) and General Purpose Operating System (GPOS) simultaneously on the same platform, since some applications require both real-time and general purposes services. The authors proposed an architecture where they use Linux as GPOS and microC/OS-II as RTOS. GPOS and RTOS run on separated CPUs independently. Devices are divided into real-time devices and non-real time devices. Since the key problem of multi-OS is hardware sharing, only memory is used as shared hardware. No software layer is inserted between the hardware and the OS. Figure 3-3 shows their proposed architecture.

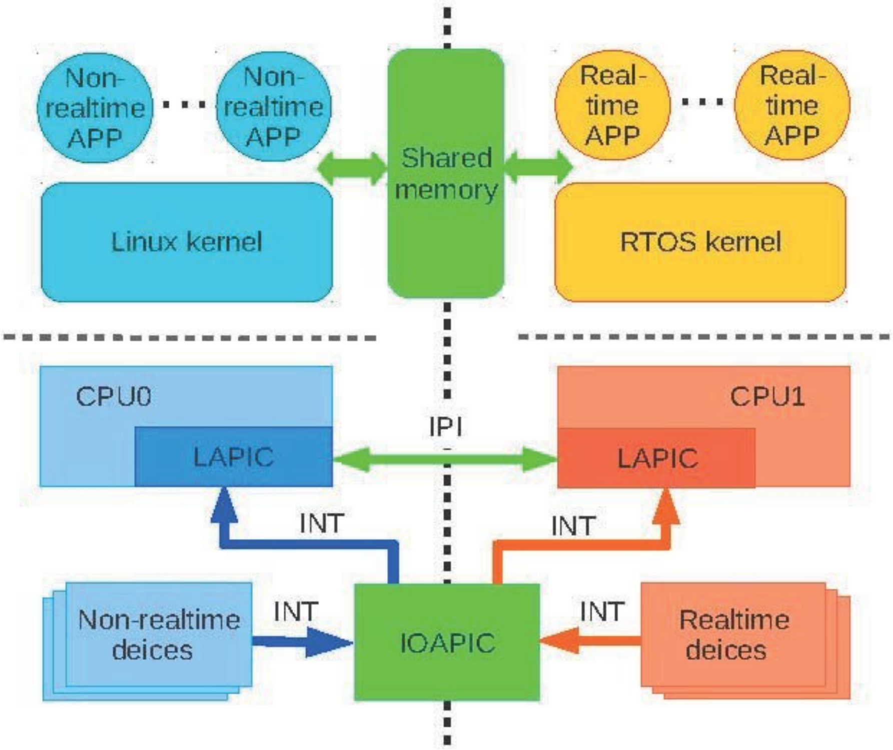


Figure 3-3 Proposed Architecture for Multi-OS [34]

Since the two OSes run on different CPUs, the synchronization of two tasks using different OSes is a main problem. The authors adopted two approaches to solve this problem. The first approach is called Mutual Exclusion which is suitable for short term and multi-processor synchronization, and the second approach is called Synchronization.

The architecture presented in Figure 3-3 mainly discusses and provides Node Level Virtualization. It doesn't offer Network Level Virtualization and does not discuss extensibility. Delegating tasks to other clouds is not applicable in this work since it is not related to cloud computing. A standard interface technology for the communication for the proposed architecture is not discussed. Isolation in terms of running more than one application on the same robot is provided, where each application runs on a different OS and a different CPU. Also, their provided solution does not discuss if it is applicable to heterogeneous robots. Finally, the scalability in terms of adding new robots is not applicable.

Reference [35] provides a framework that relies on SOA and offloads computationally-intensive algorithms from the robots to the framework. In the proposed architecture as shown in Figure 3-4 there are three main components. The first of which is the Cloud Manager which handles the robotic service requests coming from the client and checks if the specific client is allowed to use the requested service. The second component is the Robotic Service Handler where the requested service is checked and depending on its availability it is either allotted to the client if available or otherwise put in the buffer to serve it later. The last component is the Map-reduce Cluster which is used to process large amounts of data. To request a service XML is used, and the request is sent over HTTP.

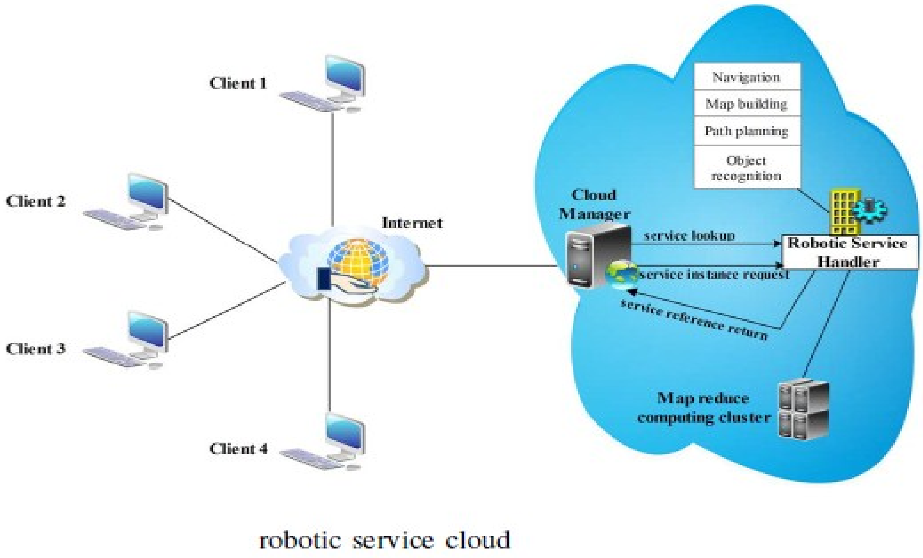


Figure 3-4 Proposed Architecture: Robotic Service Cloud [35]

The architecture proposed in this work as shown in Figure 3-4 does not provide Network or Node Level Virtualization. It provides virtualization in terms of separating services from physical devices; providing virtualization and adding robotic algorithms is the authors' future work. Delegating some tasks to other clouds and extensibility in terms of adding new functional entities are also not discussed. The architecture does not provide heterogeneity; in future support for other robots will be included. In addition, scalability in terms of adding new robots is not discussed. The communication interfaces are based on standard technologies, XML/SOAP/HTTP. Finally, the isolation between several applications running on the same architecture is not discussed.

In [36], the same authors provided a framework to support heterogeneous and low-cost robots, with the same main goal of offloading computationally intensive algorithms from the robots to the framework. In their proposed architecture, Figure 3-5, some new components have been added such as a Master Node, which is responsible for the messages published and subscribed by robots. The services come to the architecture or to the proposed robotic cloud through a WSDL interface. Registration of the robot is checked at the Service Administration Point and tested to check if the robot is allowed to use the desired service, then at the Service Registration/Removal Point the service availability is checked by Service Administration. If the service is available the Cloud Controller is notified and the request is forwarded to the requested robot.

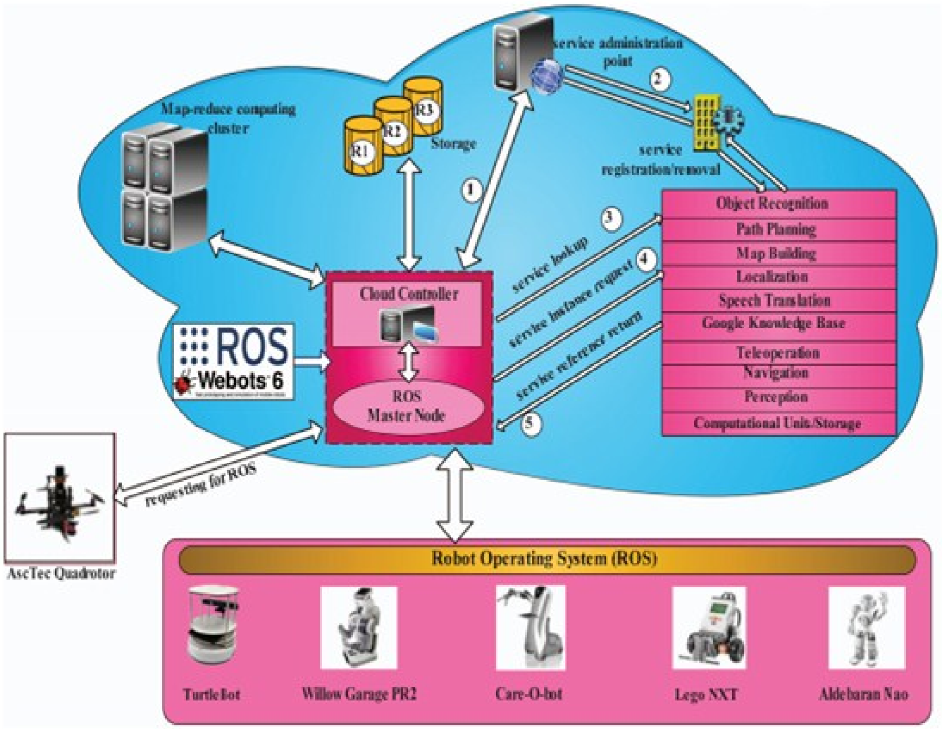
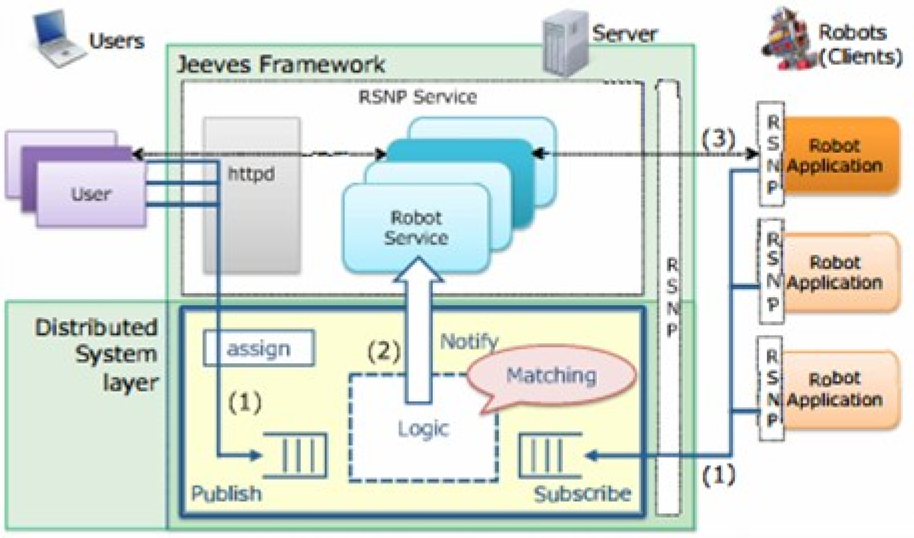


Figure 3-5 Proposed Architecture: Robot Cloud [36]

This work improves the architecture presented in [35] by providing the possibility to have heterogeneous robots using Robotic Operating System (ROS) [37], and again the communication interfaces are based on standard technologies WSDL. The other requirements are still not met such as Node and Network Level Virtualization, task delegation, scalability, isolation, and extensibility.

In Reference [38], the authors are proposing a distributed service framework using Robot Service Network Protocol (RSNP) to integrate robot services with internet services as shown in Figure 3-6. Their proposed framework can search and assign distributed robot resources dynamically and enables accessing and controlling the robots remotely through the internet. Also in their proposed framework, even developers who have no experience in the robotic field can develop robot services for different types of robots. End-users can send requests via web servers and the framework communicates with the robots to send a task via RSNP. The framework has a relational list of service requests and robot services and it uses this list to assign a task to a robot.



System architecture of Jeeves framework.

Figure 3-6 Proposed Jeeves Framework [38]

In this work the authors do not discuss anything about Network or Node Level Virtualization. However, it is mentioned that in future the performance will be improved and task assignment mechanisms will be added. For the heterogeneity, RSNP protocol which enables interoperability among robots belonging to different vendors is used for the communication with the robots, but nevertheless robots that do not support or understand RSNP will not be able to communicate with the framework. The communication between the users and the framework is done via web server using the internet (HTTP). Extensibility is not addressed in terms of adding new functional entities. Also delegating tasks to other clouds and scalability in terms of adding new robots to the architecture are not discussed.

Finally the provided framework does not discuss anything about isolating one robotic application from the other. Table 3-1 summarizes the evaluation of the related works for the architecture.

	Related Work Requirements	Robot as a Service in Cloud Computing	Design of a Robot Cloud Center	A novel multi-OS Architecture for robot application	Robotic Services in Cloud Computing Paradigm	A Framework to Assist Heterogeneous Low Cost Robots	An Implementation of a Distributed Service Framework for Cloud-Based Robot Services
IaaS	Scalability	Not Discussed	Satisfied	Not Applicable	Not Discussed	Not Discussed	Not Discussed
	Standard Interfaces	Satisfied	Satisfied	Not-Discussed	Satisfied	Satisfied	Partly Satisfied
	Isolation	Not Discussed	Not Discussed	Satisfied	Not Discussed	Not-Discussed	Not Discussed
	Heterogeneity	Not Discussed	Satisfied	Not Discussed	Not Discussed	Satisfied	Satisfied
	Extensibility	Not Discussed	Not Discussed	Not-Discussed	Not Discussed	Not-Discussed	Not Discussed
	Task Delegation	Not Discussed	Not Discussed	Not Applicable	Not Discussed	Not-Discussed	Not Discussed
	Network Virtualization	Not Discussed	Satisfied	Not Discussed	Not Discussed	Not-Discussed	Not Discussed
Robot	Node Virtualization	Not Discussed	Not Discussed	Satisfied	Not-Discussed	Not-Discussed	Not Discussed

Table 3-1 Summary of the Evaluation of the Related Work for the Architecture

3.3.2 Algorithm to Perform Network Level Virtualization

In some environments one robot may not be sufficient to perform a given task and may need collaboration with several robots to accomplish the required task in an efficient manner. For example, a given task may need several resources and since each robot has different resources (sensors and actuators), the required resources for the task may not reside in the same robot; hence the need of collaboration between several robots. Also, sometimes one robot may not be sufficient to perform a task if the requested area to perform the task is too big for it. This research area is called Multi-agent Coalition Formation which is the partitioning of the set of agents into groups called coalitions with each group being responsible for completing a task. The problem for Coalition Formation is to select the optimal set of coalitions of robots with respect to the task requirements and coalition value. The optimal solution to the coalition problem is NP-hard [39]. The problem that we are studying in this thesis is assigning a group of robots for a given task which is called Single-Task Multi-Robots Instantaneous-Assignment (ST-MR-IA) following the taxonomy presented in [40], where ST means each robot is capable of executing one task at a time, MR means that each task requires several robots to collaborate, and IA means that the available information only allows instantaneous allocations.

We have reviewed several works in this area including one work which is ST-SR-IA; Single-Task Single-Robot Instantaneous-Assignment [41], in order to review their proposed algorithm. The authors in this work have presented a dynamic task assignment algorithm integrated in a discrete event environment. The provided scenario deals with a dynamic environment, where each robot is capable of executing one task at a time, and each task requires only one robot to accomplish it. The provided algorithm called Min Conflict with Happiness algorithm (MCH) includes three steps. The first step is a greedy search for an initial assignment which may be feasible or unfeasible. The second and third steps are repair procedures to improve the assignment as much as possible.

In this algorithm many of our requirements are not applicable such as minimizing the communication between robots and minimizing the number of robots in one group, since the robots do not collaborate with each other to perform a task. Also it does not meet the rest of our requirements, such as optimizing several objectives, minimizing the cost of robot deployment, etc.

In [42], the authors mainly try to tackle the discrepancy between Multi-Agent and Multi-Robot domains. The problem is MR-MT, which is a Multi-Robot Multi-Task problem, where each robot is capable of performing more than one task, and each task requires more than one robot to start. Their proposed algorithm is comprised of two primary stages. In the first step coalition values are calculated and distributed to the agents and in the second stage each agent determines the required capabilities for each task and compares them with the capabilities of each coalition that it is a member of. It then calculates the best-expected task outcome of each coalition and selects the coalition with the best outcome.

This algorithm does not take into consideration dynamic criteria such as battery level and robots' positions. It also tries to optimize only one objective, the overall utility which is the expected net outcome when a coalition performs a task. It minimizes the communication between the robots but there is still communication among robots when they receive the coalitions' values and exchange it in inter-agent communication. The algorithm is able to select a group of robots for a given task but does not consider any filtering or ranking methods. The proposed algorithm is a modified algorithm of Shehory & Kraus's algorithm, which has excellent real-time response. However the response time is not discussed. For minimizing the number of robots in one group performing a task, there is a size limit on the maximum allowed coalition size, but minimizing it more than this value is not discussed. Minimizing the cost of robot deployment is not discussed and the time needed to perform a task is not discussed as well.

Liu and Chen in [43], proposed an algorithm based on Genetic Algorithms to find the best solution for multi-robot coalition. The proposed algorithm has the following stages: Genetic Encoding, Fitness Function Design which is the objective function that needs to be optimized, Initialization, Selection where roulette wheel selection and Monte Carlo method were used, Crossover where combining the individual with the highest fitness value with the individual with the second highest value to create better individuals were performed, Mutation, Replacement, and finally Termination.

The Genetic Algorithm (GA) presented in this work does not take into consideration the dynamic criteria when selecting the best coalition. Also it tries to optimize only one objective which is the coalition value. Minimizing the communication among robots is not discussed. However looking into the algorithm there is no communication between the robots. The proposed algorithm is able to select a group of robots or coalition for a given task but does not take into consideration any filtering or ranking methods. For the response time of the algorithm, GA was used which has quick convergence capability, but it is not discussed in the paper. For minimizing the number of robots in one group there is again a size limit on the allowed coalition size but not on minimizing the number of robots in one group. Finally minimizing the cost of robot deployment and the time needed to perform a task are not discussed. It is mentioned that the found coalition executes the task with lower cost, but it is not explained what this “cost” is.

The reference [44] proposes an algorithm to form coalitions of heterogeneous robots for a set of tasks. In the proposed algorithm the authors are trying to maximize the number of tasks completed and maximize the efficiency of the system. There is a trade-off between these two objectives. The authors are also dealing with a non-additive environment where adding new resources to a robot forming a coalition is not enough to meet the task requirement and the distribution of these resources on each robot is important as well. The problem is ST-MR (Single Task Multi Robot). Two multi-objective optimization algorithms were introduced to solve this problem: a Non-Dominated Sorting Genetic Algorithm (NSGA-II) and a Strength Pareto Evolutionary Algorithm (SPEA-II).

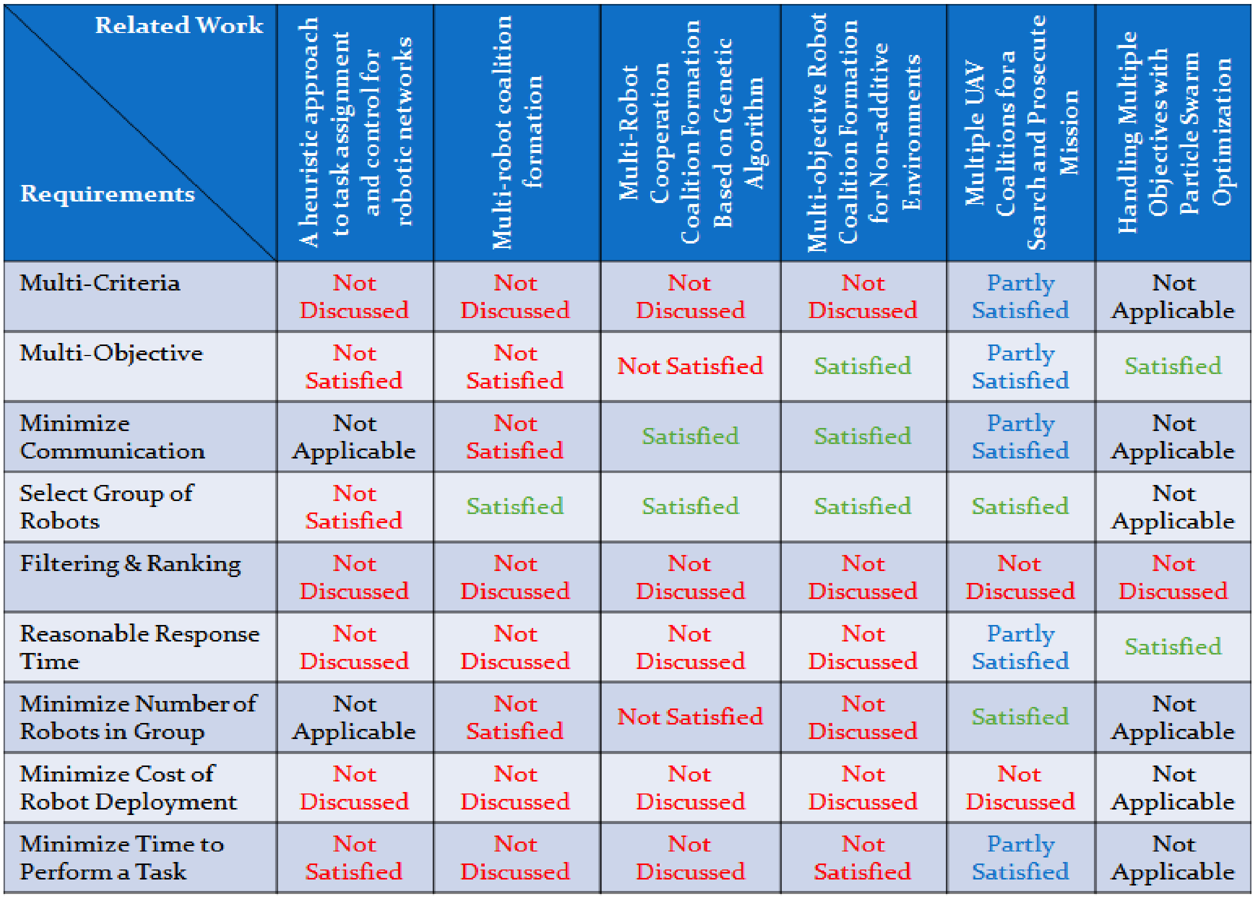
This work does not discuss the dynamic criteria such as the battery level and the location of a robot. The algorithm attempts to optimize more than one objective including maximizing the number of tasks to be executed as well as maximizing the overall system efficiency. The response time of the algorithm is not discussed. Minimizing the communication between the robots was achieved since there is a central entity which has complete knowledge of the system and generates the desired coalition schemes. Minimizing number of robots in one group and the cost of robot deployment are not discussed. For minimizing the time needed to perform a task, the authors have reworked a previous algorithm and claimed that their algorithm can minimize the time to complete all tasks compared to that approach, but they did not discuss minimizing the time to perform a task in general.

In [45] the authors try to assign Unmanned Aerial Vehicles (UAVs) to search and prosecute missions. The assigned coalition should satisfy the task requirements, minimize the target prosecution delay, minimize the size of the coalition, and simultaneously prosecute the target to induce as much damage as possible. Compared to multi-robot task allocation systems, UAVs travel with higher speed than ground robots, and the robots' resources do not deplete over time. In this work Polynomial Time Coalition Formation Algorithm (PTCFA), Optimal Coalition Formation Algorithm (OCFA), and Particle Swarm Optimization Algorithm (PSO) are provided. The PSO takes the resource depletion over time into consideration and is faster compared to Genetic Algorithms.

The presented PSO takes some dynamic criteria into consideration such as the positions of the UAVs. It tries to optimize several objectives including minimizing the cost in the time taken by a coalition to prosecute the target and reduce the resources of the agents. The agents do not communicate with each other, they only communicate with a leader, which reduces the communication costs but there is still communication among the robots. The proposed algorithm is able to select a group of robots for a given task but it does not use any filtering or ranking methods. The response time of the PSO is less compared to PTCFA and OCFA algorithms presented in the same work. Minimizing the coalition size and the number of agents in one group is one of the objectives of the algorithm. The cost of robot deployment was not discussed in terms of expense, and the time needed to perform a task is less compared to PTCFA and OCFA algorithms.

The work presented in [46] deals with multiple objectives using pareto dominance incorporated into PSO. The authors have added several steps compared to currently proposed approaches such as a constraint-handling mechanism and a mutation operator which improves their algorithm. The proposed Multi-Objective Particle Swarm Optimization (MOPSO) algorithm is not applied in the robots' domain, therefore most of our requirements are not applicable in this case such as having static criteria (sensors/actuators), minimizing the communication among robots, selecting a group of robots, minimizing the number of robots in one group, minimizing the cost of robot deployment, and minimizing the time needed by the robots to perform a task. The algorithm deals with multiple objectives, but does not take into consideration any ranking or filtering methods. The MOPSO algorithm has been compared with several other algorithms and showed that the average performance of MOPSO is the best compared to Non-dominated Sorting Genetic Algorithm II (NSGA-II), Pareto Archived Evolution Strategy (PAES), and Microgenetic Algorithm for Multi-Objective Optimization (microGA).

Based on our studies on the state of the art for both the architecture and the algorithm, and to the best of our knowledge, there is no cloud-based infrastructure that fulfills our requirements completely. Likewise, there is no algorithm that satisfies our objectives and whole requirements. Some of the works cover part of our requirements but none of them cover our requirements completely. Table 3-2 summarizes the evaluation of the related works for the algorithm.



Related Work Requirements	A heuristic approach to task assignment and control for robotic networks	Multi-robot coalition formation	Multi-Robot Cooperation Coalition Formation Based on Genetic Algorithm	Multi-objective Robot Coalition Formation for Non-additive Environments	Multiple UAV Coalitions for a Search and Prosecute Mission	Handling Multiple Objectives with Particle Swarm Optimization
Multi-Criteria	Not Discussed	Not Discussed	Not Discussed	Not Discussed	Partly Satisfied	Not Applicable
Multi-Objective	Not Satisfied	Not Satisfied	Not Satisfied	Satisfied	Partly Satisfied	Satisfied
Minimize Communication	Not Applicable	Not Satisfied	Satisfied	Satisfied	Partly Satisfied	Not Applicable
Select Group of Robots	Not Satisfied	Satisfied	Satisfied	Satisfied	Satisfied	Not Applicable
Filtering & Ranking	Not Discussed	Not Discussed	Not Discussed	Not Discussed	Not Discussed	Not Discussed
Reasonable Response Time	Not Discussed	Not Discussed	Not Discussed	Not Discussed	Partly Satisfied	Satisfied
Minimize Number of Robots in Group	Not Applicable	Not Satisfied	Not Satisfied	Not Discussed	Satisfied	Not Applicable
Minimize Cost of Robot Deployment	Not Discussed	Not Discussed	Not Discussed	Not Discussed	Not Discussed	Not Applicable
Minimize Time to Perform a Task	Not Satisfied	Not Discussed	Not Discussed	Not Satisfied	Partly Satisfied	Not Applicable

Table 3-2 Summary of the Evaluation of the Related Work for the Algorithm

3.4 Chapter Summary

In this chapter we presented three motivating scenarios that illustrate the need for cloud-based architecture and virtualization in robotic applications: the Wildfire Suppression, the Subway Degradation Detection, and the Search and Rescue robots scenarios. Then we extracted the set of requirements based on the scenarios presented. We categorized the requirements into three groups, general requirements on the IaaS, requirements on the robots hosted by the IaaS, and requirements on the algorithm to perform Network Level Virtualization in the IaaS. Then we reviewed and evaluated the state of the art based on the requirements that we presented. Finally we concluded that none of the works presented in the state of the art met all of our requirements completely.

Chapter 4

4 Proposed Architecture

In the previous chapter we derived the appropriate requirements for the IaaS for cost-efficient robotic applications as cloud computing services. Accordingly, this chapter aims to propose a suitable architecture based on these derived requirements.

This chapter begins with an explanation of the overall architecture along with the architectural principles. Then, in the second section it presents the detailed architecture, where the overlay, the functional entities, the interfaces and the procedures are discussed. In the third section, it discusses how the requirements are met by the architecture. Finally, the chapter is summarized.

4.1 Overall Architecture

In this section, we start by presenting the architectural principles we adopted in designing the architecture. Then, we give a description of the overall architecture.

4.1.1 Architectural Principles

To design our architecture, we followed a set of principles. The first principle is the use of peer-to-peer (P2P) overlays for the communication between different IaaSs that are part of our proposed system. We used P2P overlays because they provide distributed architectures, do not require centralized control, are self-reorganized, and allow scalability in terms of the number of the nodes in the overlay. In addition, in P2P overlays there is no single point of failure [47].

The second principle is that the interaction interfaces of the IaaS with the PaaS, which is part of our proposed system, and the interaction interfaces of the different layers of the IaaS are REpresentational State Transfer (REST)-based. We selected REST because it is lightweight, standards-based, and can support multiple data representations (e.g., plain text, JSON, and XML). It does not depend on any specific communication protocol, but is most commonly used with HTTP. REST has three main design principles: **addressability**, which means that the REST models the information as resources, where a resource is any information that is important to be named and referenced. Each resource should be addressable via a unique identifier; **uniform interfaces**, which means that REST resources can be accessed and manipulated in a standard way, and **statelessness**, which means that each request is isolated from previous requests and leads to better scalability and performance [48].

4.1.2 Architecture Overview

The overall architecture is depicted in Figure 4-1. It is mainly comprised of a P2P Overlay and a Robotic Cloud. The Robotic Cloud includes PaaS and the IaaS, as shown in Figure 4-2. We will focus on the IaaS Layer

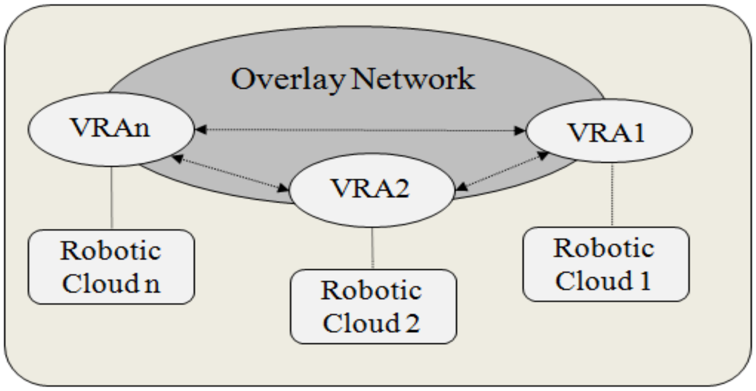


Figure 4-1 Overall Architecture

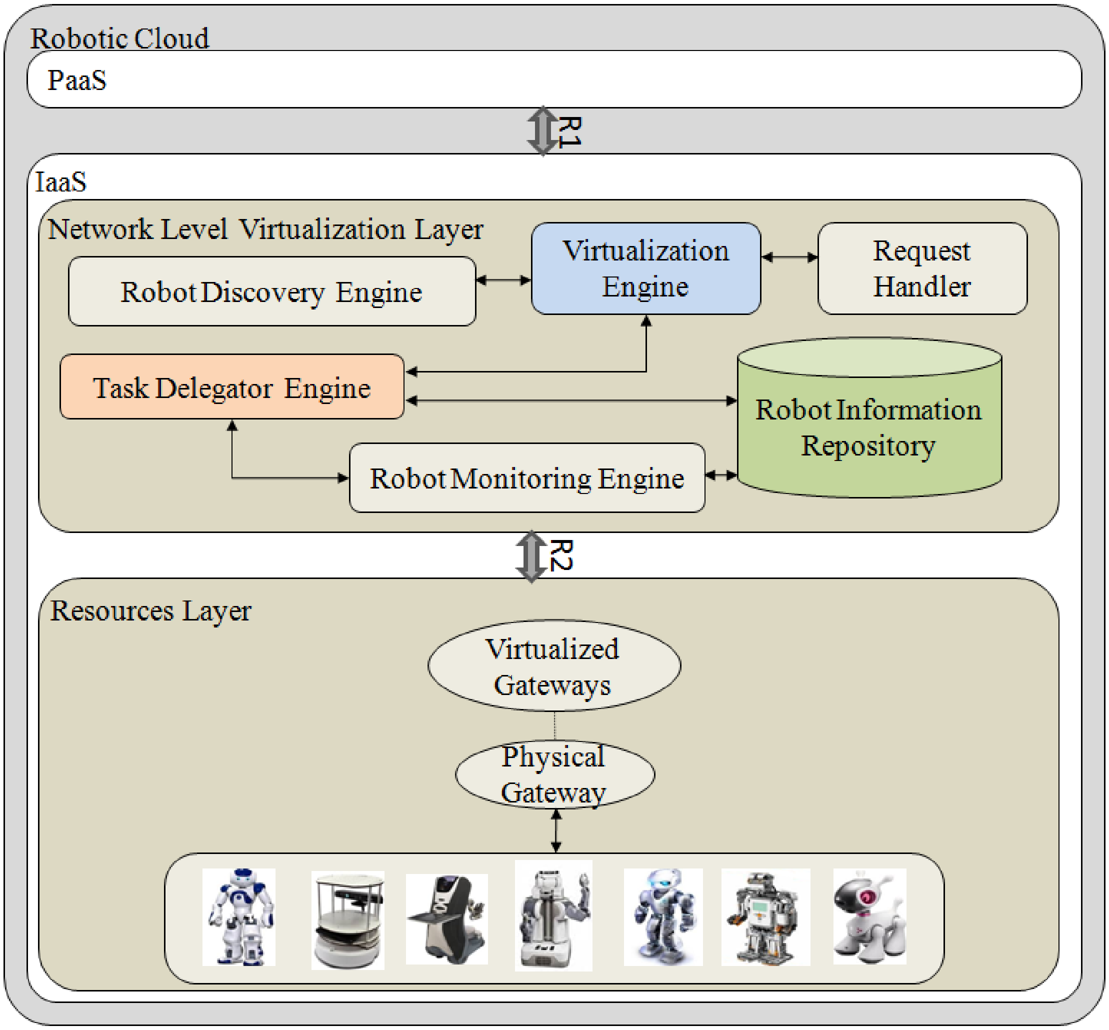


Figure 4-2 The Robotic Cloud

The IaaS consists of two layers: The Network Level Virtualization Layer and the Resources Layer. The P2P Overlay is used as the interaction network between different IaaSs. We have one type of node in the overlay called *VirtualRoboticAgent* (VRA), which represents the Robotic Cloud in the Overlay. We describe this node's architecture in Section 4.2.1.1.

The IaaS interacts with the PaaS to receive a request, and selects robots for that request in a cost-efficient manner at the Network Level Virtualization Layer. It first divides the task into subtasks, and then assigns the subtasks to the robots via the Gateway at the Resources Layer. It should be noted that some of the selected robots might belong to other IaaSs. The IaaS discovers them through the Overlay and delegates tasks to them via their IaaS through the same Overlay.

The Gateway caters to heterogeneity by mediating between the standard interface supported by the Network Level Virtualization Layer and the proprietary interfaces supported by the robots. It receives the task assignment request from Network Level Virtualization Layer and creates a Virtualized instance of the Gateway based on the interface supported by the requested robot.

4.2 Detailed Architecture

In this section we start by describing the Overlay Network where we discuss the Overlay Node's architecture, the Overlay protocols that we designed, and the messages exchanged in the Overlay. Subsequently, we describe the Robotic Cloud where we list and explain the functional entities involved, and we present the communication interfaces along with the REST resources between the IaaS and PaaS, and between the different layers of the IaaS. Finally, we describe four main functional procedures.

4.2.1 P2P Overlay Network

In the first subsection we describe the Overlay Node architecture. The Overlay protocols are presented in the second subsection in which we defined two protocols: Request/Response protocol and Subscribe/Notify protocol.

4.2.1.1 Functional Entities of P2P Overlay Network

The Overlay Network includes only one functional entity which is the Overlay Node, called *VirtualRoboticAgent* (VRA). Each VRA represents one robotic cloud in the Overlay. A VRA communicates with the pertinent Robotic Cloud as well as the other nodes in the Overlay Network. Figure 4-3 shows the architecture of the VRA node.

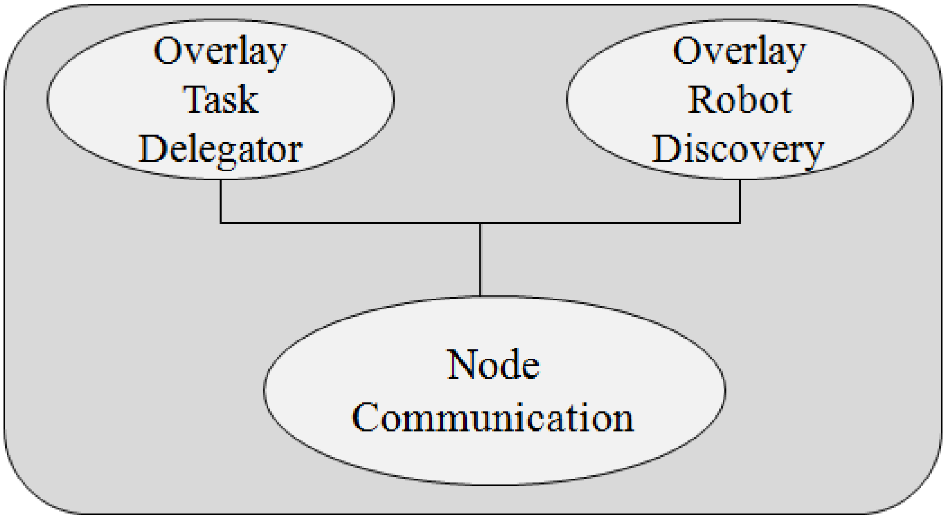


Figure 4-3 The VRA Node Architecture

A VRA node composes three components: the Node Communication, the Overlay Robot Discovery, and the Overlay Task Delegator. .

- <u>The Node Communication</u>

  The Node Communication allows communication with other nodes in the Overlay. It receives the requests coming from other nodes in the Overlay. It also sends the request coming from the Overlay Robot Discovery and Overlay Task Delegator components to the other nodes in the Overlay. The Node Communication component is also responsible for the interactions with the Robotic Cloud.

- The Overlay Robot Discovery

  The Overlay Robot Discovery component receives the idle robot discovery request from its corresponding IaaS. It sends the discovery request to the other nodes in the Overlay through the Node Communication component.

- The Overlay Task Delegator

  The Overlay Task Delegator component is responsible for sending the task subscription request to other nodes in the Overlay via the Node Communication component. It may also receive the task subscription request from other nodes, in which case it is responsible for sending back notification to the node that sent the task subscription request.

4.2.1.2 Overlay Protocols and Messages

We defined two protocols for the Overlay, the first being the Request/Response protocol, and the second being the Subscribe/Notify protocol. The Request/Response protocol uses the messages summarized in Table 4-1 between the Overlay nodes. The "Discover Request" message is a broadcast message since it is sent to all the nodes in the Overlay in order to get the information of all idle robots belonging to all the IaaSs, and the "Discover Response" message is a unicast, since it is sent directly to the node that sent the "Discover Request" message. When a VRA receives a "Discover Request" message, it asks its corresponding IaaS for the list of idle robots belonging to it, and sends a response message back to the VRA that issued the message. If it has idle robots it includes the list of these robots in the response. Otherwise, it sends an empty response.

The Subscribe/Notify protocol uses the messages summarized in Table 4-2. The "Task Subscription" and the "Task Notification" messages are unicast. The "Task Subscription" is sent directly to the VRA informing that robots belonging to it were selected. When a VRA receives a "Task Subscription" message, it subscribes the VRA that sent the message for event notification, and sends the message to its corresponding IaaS with the robot Id that was selected.

The "Task Notification" message is sent directly to the VRA that sent the "Task Subscription" message and subscribed for event notification. When a VRA receives a "Task Notification" message from its IaaS, it sends it to the subscribed VRA, which in turn notifies its corresponding IaaS. These messages should be transported using a reliable transport protocol such as TCP.

Message Name	***Message Description***	***Message Address***
Discover Request	Discover Idle robots belonging to other IaaSs Sent by a VRAx to the overlay after receiving a Discover request from a RoboticCloud	Broadcast
Discover Response	Sends a response to another node. VRAx sends a response to VRAy when it receives Discover Message from VRAy, with the list of Idle robot belonging to its IaaS	Unicast

Table 4-1 Messages Exchanged in the Overlay for Protocol Request/Response

Message Name	***Message Description***	***Message Address***
Task Subscription	Send the task to robots belonging to other IaaS and implicitly subscribe for event notification. Sent by a VRAx to the specified VRAs	Unicast
Task Notification	Sends notification to the subscribed node when robots finish their task. Sent by a VRAx to the VRA that subscribed for event notification	Unicast

Table 4-2 Messages Exchanged in the Overlay for Protocol Subscribe/Notify

4.2.2 Robotic Cloud

The Robotic Cloud, as we discussed in the previous sections (section 4.1.2), consists of the PaaS and the IaaS. And since our focus is on the IaaS layer, we will explain the functional entities of the IaaS. The IaaS includes two layers: the Network Level Virtualization Layer and the Resources Layer. In the Network Level Virtualization Layer there are six functional entities: the Request Handler, the Virtualization Engine, the Robot Discovery Engine, the Task Delegator Engine, the Robot Information Repository, and the Robot Monitoring Engine. In the Resources Layer there are the Virtualized Gateways and the Physical Gateway.

In the following sections, first we will explain the functional entities of the Robotic Cloud, and then in the next section we will explain the interfaces used for the communication between the IaaS and the PaaS, and also the communication interfaces between the IaaS's layers.

4.2.2.1 The Functional Entities

- Network Level Virtualization Layer:
  - ❖ Request Handler:

    The Request Handler handles the request coming from the PaaS; it analyzes the request and derives the task requirements and the constraints.

  - ❖ Virtualization Engine:

    The Virtualization Engine performs Network Level Virtualization by using an appropriate algorithm for coalition formation in Multi-Robot systems [40], in order to choose and assign the most suitable group of robots for a given task. It runs the algorithm on local robots and robots belonging to other IaaSs.

- Robot Discovery Engine:

  The Robot Discovery Engine is responsible for discovering local and external idle robots. The local robots are discovered through the Robot Information Repository, and the external robots are discovered via the Overlay.

- Task Delegator:

  The Task Delegator Engine sends task assignment requests to robots belonging to other IaaSs through the Overlay, and to the local robots through Robot Monitoring Engine. It may also receive task assignment request from other IaaSs.

- Robot Information Repository:

  The Robot Information Repository holds a list of robots belonging to the IaaS with their states (busy/idle), capabilities (sensors/actuators) and constraints. This component is modified every time we add or remove a robot from the IaaS, also when a robot changes its state from busy to idle or when a robot fails.

- Robot Monitoring Engine:

  The Robot Monitoring Engine monitors the robots' status. A robot sends a notification when it finishes its subtask to move back from busy to idle state, and this Engine updates the Robot Information Repository. Also it updates the Repository when a robot fails. This entity also sends the task assignment request to the robots which are in the Resources Layer.

- Resources Layer
  - ❖ Virtualized Gateways

    The Virtualized Gateways is the entity that enables the architecture to support heterogeneous robots. It receives a task assignment request from the Robot Monitoring Engine in the Network Level Virtualization Layer by the standard interface supported by this layer, and creates Virtualized instance of the Gateway to send the request to the robot based in the interface supported by the desired robot.

  - ❖ Physical Gateway

    The Physical Gateway is the actual Gateway that communicates with the robots to send them the request received from the Network Level Virtualization Layer.

4.2.2.2 Interfaces

REpresentational State Transfer (REST) is the main communication interface in the proposed architecture. The resource is a key concept in REST. It represents any information important enough to be modeled and uniquely identified via Uniform Resource Identifier (URI). The REST interface is used for the communication between the IaaS and the PaaS (R1), and also for the communication between the Network Level Virtualization Layer and the Resources Layer in the IaaS (R2).

4.2.2.2.1 Gateway Resources

Table 4-3 summarizes the proposed REST interface for the communication between the Network Level Virtualization Layer and the Resources Layer. It defines the resources on the Resources Layer side.

The Resources Layer side resources are used to reserve robot resources when adding new robots to the IaaS, to reserve resources when creating a group of robots, and to reserve task resources when sending a task to a specific robot or group of robots. They are also used to modify the task resources for an ongoing task. Furthermore, they are used to get the list of robots to update the repository, to get information about a specific group such as the members of the group and their capabilities, and to get the state of an ongoing task. Finally, they are used to delete the resource for a group of robots when they finish their assigned task.

Resource	***Operation***	***Http Action***	***Client->Server***	***Server->Client***
List of Robots	Create: Add a Robot	POST:/Robots	Robot Description: Robot, Id, State, Capabilities	/Robots/{RobotId}
	Read: Get list of all robots to update the repository	GET:/Robots	None	All robots with their description
List of Groups	Create: Add a group of Robot	POST:/Robots/Groups	Group Description: Group Id, robots in group	/Robots/Groups/{GroupId}
Specific Group	Read: Get information about specific group	GET:/Robots/Groups/{GroupId}	None	Group Description

	Delete: Delete the specific group after the task is finished	DELETE:/Robots/Groups/{GroupId}	None	None
List of Tasks for specific Robot	Create: Send the task to specific robot	POST:/Robots/{RobotId}	Task Description	/Robots/{RobotId}/{TaskId}
Specific Task Specific Robot	Read: Get the state of the task from a specific robot	GET:/Robots/{RobotId}/{TaskId}	None	Task Description
	Update: Change the status of an ongoing task on specific robot	PUT:/Robots/{RobotId}/{TaskId}	The new state of the task	None
List of Tasks for Specific Group	Create: Send the task to specific Group	POST:/Robots/Groups/{GroupId}	Task Description	/Robots/Groups/{GroupId}/{TaskId}
Specific Task Specific Group	Read: Get the state of the task from a specific Group	GET:/Robots/Groups/{GroupId}/{TaskId}	None	Task Description
	Update: Change the status of an ongoing task on specific group	PUT:/Robots/Groups/{GroupId}/{TaskId}	The new state of the task	None

Table 4-3 Resources on the Gateway

4.2.2.2.2 IaaS Resources

Table 4-4 summarizes the proposed REST interface on the Network Level Virtualization Layer side. The Network Level Virtualization Layer side resources allow the Resources Layer to send notification to the Network Level Virtualization Layer when the robots finish their assigned task or when there is a failure in one of the robots.

Also, it allows the PaaS to send a request to the IaaS (Network Level Virtualization Layer), to get the status of a request such as whether the request has been accomplished or not, and to cancel a request in the middle of the execution.

Resource	***Operation***	***Http Action***	***Client->Server***	***Server->Client***
List of Requests	Create: Send a request to the IaaS (implicitly subscribe for notification)	POST: /Requests	Request Description	/Requests/{RequestId}
Specific Request	Read: Get the status of the request	GET:/Requests/{RequestId}	None	Request Description
	Delete: Cancel the request in the middle of execution	DELETE:/Requests/{RequestId}	None	None
Notification of failure	Create: Send notification when there is failure in any robot	POST:/FailureNotification	Robot Id	/FailureNotification/{NotificationId}
Notification of State change	Create: Send notification when robot finish task	POST:/TaskNotification	Robot Id	/TaskNotification/{NotificationId}

Table 4-4 Resources on the IaaS

4.2.2.2.3 PaaS Resources

Table 4-5 summarizes the proposed REST interface for the communication between the PaaS and the IaaS. It defines the resources on the PaaS side. The PaaS side resources allow the IaaS to send notification to the PaaS when the requested task finishes.

Resource	***Operation***	***Http Action***	***Client->Server***	***Server->Client***
Notification	Create: Send notification from IaaS to PaaS	POST:/Notification	Request Id	None

Table 4-5 Resources on the PaaS

4.2.2.3 Procedures

This section discusses the four main procedures: Idle Robot Discovery, Selecting Robots for a Given Task, Subtask Assignment for the Selected Robots and Notification of a Finished Subtask.

- Idle Robot Discovery:

  The functional entities involved in this procedure are the Robot Discovery Engine and the Robot Information Repository in the Network Level Virtualization Layer.

  The Idle Robot Discovery procedure is used by the Robot Discovery Engine to discover idle robots belonging to local and other IaaS. It allows the Virtualization Engine to perform Network Level Virtualization algorithm on all robots, including robots that belong to other IaaS. This procedure is triggered when the IaaS receives a request from the PaaS; where the Request Handler sends the request to the Virtualization Engine, and the latter asks the Robot Discovery Engine to discover the idle robots. The discovery of local idle robots is done through a look up in the Robot Information Repository, and the discovery of idle robots belonging to other IaaS is done through the Overlay.

- Selecting Robots for a Given Task:

  The only functional entity involved in this procedure is the Virtualization Engine which is in the Network Level Virtualization Layer.

  Selecting Robots for a Given Task is performed by the Virtualization Engine after it receives the list of idle robots belonging to all IaaS from the Robot Discovery Engine. The Virtualization Engine runs a coalition formation algorithm for Multi-Robot systems which is discussed in the next Chapter (Chapter 5). It selects the robots in cost-efficient manner by considering the robots sensing skills/actuating capabilities, the task requirements, the robots constraints such as the battery level and the speed, and the task constraints such as location, execution delay, cost agreed with the customer, and the number of robots involved in one group.

- Subtask Assignment for the Selected Robots:

  The functional entities involved in this procedure are the Virtualization Engine, the Robot Monitoring Engine in the Network Level Virtualization Layer, and also the Resources Layer.

  After the Virtualization Engine selects the most appropriate group of robots for a given task, the Subtask Assignment for the Selected Robots procedure occurs. This procedure involves dividing a task into subtasks and assigning each subtask to the appropriate robot from the group of selected robots. This assignment is also done in a cost-efficient manner considering robots sensing skills/actuating capabilities and the subtask requirements. The subtask is assigned to the local robots through the Virtualized Gateway in the Resources Layer, which creates an instance of the Gateway to communicate with the robots, and to robots belonging to other IaaS via the Overlay.

- Notification of a Finished Subtask:

  The functional entities involved in this procedure are the Robot Monitoring Engine and the Robot Information Repository in the Network Level Virtualization Layer, and the Resources Layer.

The Notification of a Finished Subtask is sent by the robots, when they finish their subtask, it is sent to the Gateway where the latter sends it to the Robot Monitoring Engine in the Network Level Virtualization Layer. The Robot Monitoring Engine updates the Robot Information Repository by changing the robots' states from busy to idle state.

4.3 How the Requirements are Met by the Architecture

The refined architecture satisfies all the requirements that we mentioned and focused on in Chapter 3. First, the IaaS can accommodate any number of robots; it can scale in terms of adding new robots to the IaaS. This is achieved because the gateways are instantiated on-demand when the number of robots increases.

The communication interfaces of the IaaS are based on standardized technology. The IaaS interacts with the PaaS through a REST interface. The IaaS also interacts with the robots that are hosted by itself via a standard REST interface through the Gateway, independently of the interfaces supported by the robots' providers. Also, the IaaS is able to support heterogeneous robots. The Gateway is responsible for satisfying this requirement; it caters to heterogeneity. The Gateway receives a task from the Robot Monitoring Engine by the standard interface and creates an instance of Virtualized Gateway to send the request to the desired robot through the proprietary interface supported by that robot. Furthermore, it is possible to add new functional entities to the IaaS since we are following a modular design, which means that the IaaS is extensible in terms of adding new functionalities to it.

The IaaS is able to delegate tasks to robots belonging to other IaaS. This requirement is achieved by the Task Delegator Engine, which delegates tasks to robots belonging to other IaaSs through the Overlay.

The Network Level Virtualization is achieved by the Virtualization Engine, which can run a coalition formation algorithm for Multi-Robot systems to select the most appropriate group of robots for each task in a cost-efficient manner. The next chapter (Chapter 5) is dedicated to Network Level Virtualization Algorithm.

Figure 4-4 shows the event sequencing diagram of the procedures.

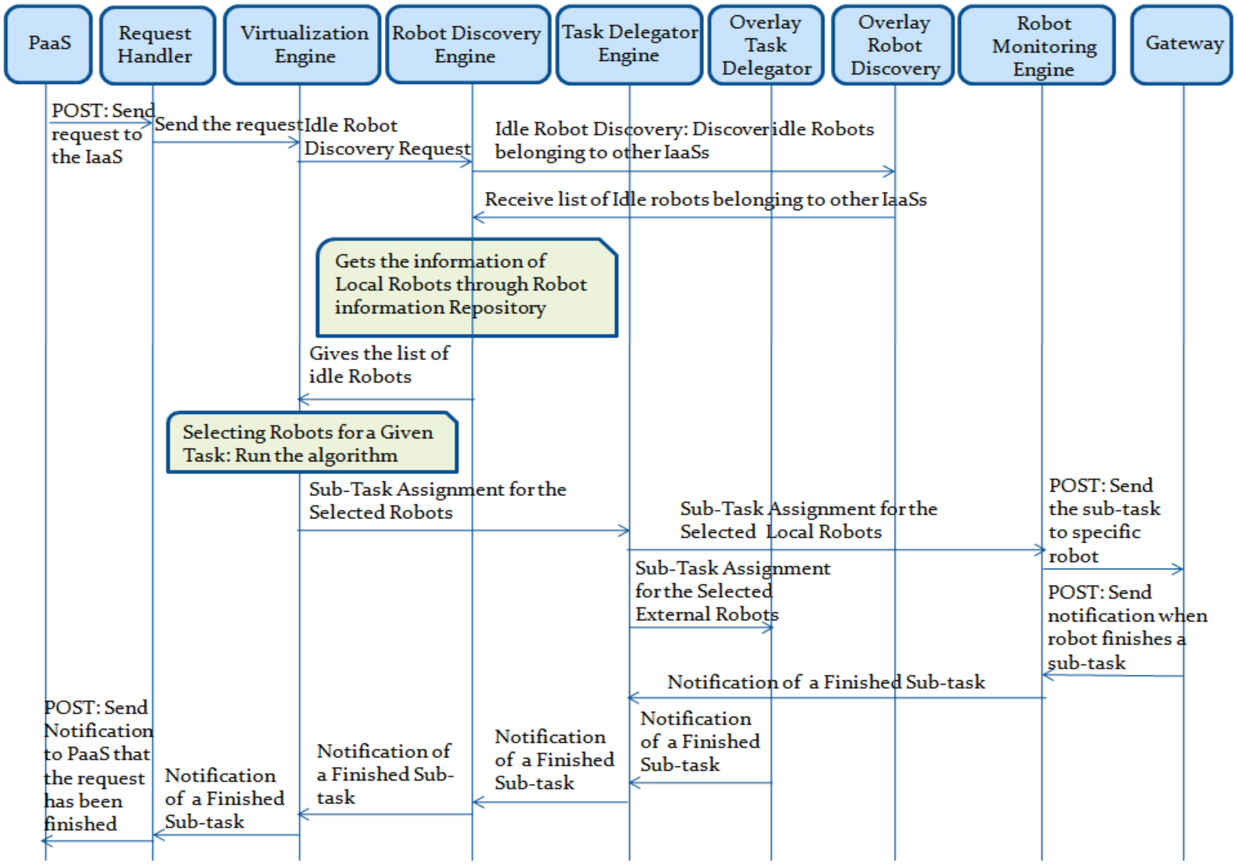

PaaS
Request Handler
Virtualization Engine
Robot Discovery Engine
Task Delegator Engine
Overlay Task Delegator
Overlay Robot Discovery
Robot Monitoring Engine
Gateway
POST: Send request to the IaaS
Send the request
Idle Robot Discovery Request
Idle Robot Discovery: Discover idle Robots belonging to other IaaSs
Receive list of Idle robots belonging to other IaaSs
Gets the information of Local Robots through Robot information Repository
Gives the list of idle Robots
Selecting Robots for a Given Task: Run the algorithm
Sub-Task Assignment for the Selected Robots
Sub-Task Assignment for the Selected Local Robots
POST: Send the sub-task to specific robot
Sub-Task Assignment for the Selected External Robots
POST: Send notification when robot finishes a sub-task
Notification of a Finished Sub-task
Notification of a Finished Sub-task
Notification of a Finished Sub-task
Notification of a Finished Sub-task
Notification of a Finished Sub-task
POST: Send Notification to PaaS that the request has been finished

Figure 4-4 The Sequence Diagram of the Procedures

4.4 Chapter Summary

In this chapter we explained our proposed architecture which comprises a P2P Overlay and a Robotic Cloud where the latter includes PaaS and IaaS. We went through the architectural principles, and then we explained the architecture in detail. We presented the Overlay protocols where we had two protocols; Request/Response and Subscribe/Notify protocols, the Overlay node's architecture, the functional entities, the interfaces, and the procedures. Finally in the last section, we discussed the requirements met by the architecture, where we showed that we satisfied all the requirements that we presented and focused on in Chapter 3.

Chapter 5

5 Network Level Virtualization Algorithm

In this chapter we discuss and present the heuristic algorithm proposed for Network Level Virtualization. First, we start by giving background information on the concepts used in the proposed algorithm. In the second section, we present our proposed algorithm. In the next section we discuss how the requirements are met by the algorithm. Finally, in the last section we give an illustrative scenario to show how our algorithm works in real world scenarios.

5.1 Background information

In this section we give background information on the concepts used in our proposed algorithm. First, we start by giving background information on Heuristic Algorithms, since our proposed algorithm is a heuristic algorithm. In the second subsection, we give background information on Multi-Objective Optimization problems, since the problem that we are trying to optimize is multi-objective in that it aims to optimize several objectives at the same time.

In the last subsection we give background information on the Promethee Ranking method which is a Multi-Criteria Decision Making (MCDM) method and is the ranking method used by the proposed algorithm. Multi-Objective Optimization problems have a set of solutions which we will discuss in the following subsections. A ranking method can be used in order to rank these solutions and select the most effective solutions.

5.1.1 Background Information on Heuristic Algorithms

Heuristic Algorithms are designed for solving a problem that cannot be easily solved quickly, or for finding an approximate solution when classic methods fail to find any exact solution [49]. The solution that it produces may not be the best of all actual solutions, it may simply approximate the exact solution, but it is still valuable because of the short time it requires to find this solution. Evolutionary Algorithms and Swarm Intelligence are two heuristic techniques. They are explained in the following subsections.

5.1.1.1 Evolutionary Algorithms

Evolutionary Algorithms are one heuristic technique. They are population-based optimization algorithms which are inspired by biological evolution, such as reproduction, mutation, recombination and selection. The fitness function which is the objective function, determines the value of a solution. These algorithms apply the principle of survival on a set of solutions to produce better solutions. A new set of solutions are created by selecting individuals according to their fitness value, then breeding them together using the operators mentioned above [50].

Genetic Algorithms (GAs) are the most successful algorithms among the Evolutionary Algorithms. They have been investigated by John Holland in 1975 and they demonstrated essential effectiveness. Genetic Algorithms search for the optimal solution until a specified termination condition is met. The solution to a problem is called a chromosome, which consists of a collection of genes that are simply parameters to be optimized. It starts by creating an initial population (chromosomes), and then tries to improve the population through the operators mentioned above.

5.1.1.2 *Swarm Intelligence*

Swarm intelligence [50] is another heuristic technique. It has a collective behaviour in decentralized, self-organized systems. Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) are two successful types of this technique. We will describe PSO in more detail because the proposed algorithm is a PSO Algorithm. In ACO, artificial ants build solutions by moving on the problem graph and changing it in such a way that future ants can build better solutions.

The PSO is inspired by the behaviour of bird flocks. Each particle in a swarm is a potential solution in the search space, and a particle adjusts its velocity according to its own flying experiences and its flock's experiences. Also, each particle keeps its best positions in a memory [45]. It is a population based optimization technique developed by Eberhart and Kennedy [51]. It requires only primitive mathematical operators. PSO is effective with several kinds of problems. It is initialized with a population of random solutions, each solution called a particle is assigned a random velocity, and then the particles are moved within the search space. The collection of particles is called swarm.

Each particle keeps in its memory its best solution achieved so far which is the fitness value. The main concept of PSO is that at each iteration or time step the particle has to move to a new position, it does this by adjusting its velocity according to the following two equations:

$$v^{k+1} = x(wv^k + c_1r_1(pBest^k - x^k) + c_2r_2(gBest^k - x^k))$$

$$x^{k+1} = x^k + v^{k+1}$$

$pBest$, is the best solution this particle has ever reached.

$gBest$, is the best solution obtained so far by any particle in the population.

c_1,c_2 are constants called cognitive and social scaling parameters respectively which help in convergence of the solution.

r_1, r_2 are random numbers used to maintain the diversity of the swarm population.

k, is the iteration number.

w, is the inertia weight which is used to control the velocity. It determines the contribution rate of a particle's previous velocity to its velocity at the current time step [52].

x, is the constriction factor which controls the effect of velocity on the particles.

5.1.2 Background Information on Multi-Objective Optimization

Multi-objective problems have several objectives that need to be optimized simultaneously. Conflicts always exist between two or more objective functions that need to be optimized; hence there is not usually a best solution with respect to all objectives. In contrast with single-objective optimization, multi-objective problems are characterized by trade-offs between the objective functions, hence they have set of solutions that are called non-dominated solutions or Pareto optimal solutions. In the following subsections we will present first the Multi-Objective Optimization Concept, and then we will present the Multi-Objective Particle Swarm Optimization algorithm.

5.1.2.1 Multi-Objective Optimization Concept

Let $S \subset R^n$ be an n-dimensional search space, and $f_i(x), i = 1, \ldots, k$ is a k objective functions defined over$S$. Themulti-objective optimization problem can be viewed as:

$$\text{Minimize} f(x) \ = \ [(f_1(x), f_2(x), \ldots f_k(x)]$$

$$\text{Subject to } g_i(x) \leq 0 \qquad i = 1,2, \dots m$$

$$h_j(x) \leq 0 \qquad j = 1,2, \dots m$$

Where$x = (x_1, x_2, \dots, x_n) \epsilon R^n$, is the decision-making vector on the search space. $f(x)$ is the goal vector, $g_i(x) and\ h_j(x)$ are the constraint functions of the problem.

The objective functions $f_i(x)$ can be conflicting with each other, so that finding the single global minimum is impossible. That's why optimality of a solution in multi-objective problems needs to be redefined properly.

Let $u = (u_1, u_2, \dots u_k)$ and $v = (v_1, v_2, \dots, v_k)$ be two vectors of search space S. We can say u dominates v $(u < v)$ if and only if u is better than v $(u_i \leq v_i)$ for all $= 1,2, \dots, k$, and $u_i < v_i$ for at least one component. This is called *Pareto dominance.* A solution x, of the multi-objective problem $f(x)$ is said to be *Pareto optimal*, if and only if there is no other solution y in S such that $f(y)$ dominates $f(x)$. We can say that solution x is non-dominated solution. If there is more than one Pareto optimal solution, we call them *Pareto optimal set*.

5.1.2.2 Multi-Objective Particle Swarm Optimization

The MOPSO algorithm was developed was developed by Coello-Coello et al [53]. There are two fundamental approaches in the MOPSO algorithm. In the first approach each particle is evaluated by one objective function at a time and the determination of the best position is performed similarly in the case of single-objective optimization. In the second approach, each particle is evaluated by all the objective functions, and based on the concept of Pareto optimality they produce non-dominated best positions that are used to guide the particles. There may exist several non-dominated solutions in the neighborhood of a particle, only one is used as the best position of a particle to update the velocity equation [53].

5.1.3 Background Information on Promethee Ranking

The Promethee I which is a partial ranking and Promethee II which is a complete ranking, were developed by J.P Brans, and presented for the first time in 1982 at University of Laval in Quebec, Canada at the conference "L'ingénierie de la décision". Later on, Promethee III and Promethee IV were also developed by J.P. Brans and B. Mareschal [54][55]. It is multi-criteria decision problem. The same authors suggested visual, interactive modulation GAIA, which represents a graphic interpretation of Promethee methods. In 1992 and 1995, more modifications were suggested, Promethee V and Promethee VI. The success of the Promethee method is basically due to its mathematical properties and to its friendliness of use. In the proposed algorithm we used the Promethee II ranking method.

5.2 Proposed Algorithm

We used in our algorithm the PSO approach which is a heuristic algorithm. PSO has gained significant attention over other heuristic approaches such as GA. References [56][57][58] compare these two approaches and show the advantages of PSO over GA. PSO computationally is less expensive and more efficient since it uses fewer functions and mathematical operators compared to other approaches. Both PSO and GA are population-based search approaches; GA accepts only binary encoding while PSO can take any value to start.

In the following subsections we present our proposed algorithm for coalition formation in Multi Robot systems. Coalition Formation in Multi-Robot systems has received significant attention, where coalition members coordinate in order to achieve the coalition's goal(s) or objective(s). Horling and Lesser defined a coalition as the following [59]:

> *"Coalitions in general are goal-directed and short-lived; they are formed with a purpose in mind and dissolve when that purpose no longer exists, or when they cease to suit their designed purpose, or when the profitability is lost as agents depart."*

First, we start by presenting the assumptions and the problem statement. In the second subsection we give an overall view of the algorithm that we propose. Firstly, we discuss the Filtering and Ranking methods that we used in the algorithm along with the Pseudocode for each of them. These methods help in reducing the number of robots that the MOPSO will run on, and make other robots available for other requests coming to other IaaSs. For the Filtering, we used a simple method. For the Ranking method, we used Promethee II ranking which is a Multi-Criteria ranking method as explained in section 5.1.3. Secondly, we present and discuss the MOPSO algorithm that we used in order to select the most efficient coalition for each task.

5.2.1 Assumptions and Problem Statement

Let us define an infrastructure composed of n robots:

$$N = \{R_1, \ldots, R_i, \ldots, R_n\}$$

Each of these robots has set of capabilities; sensing skills and actuating capabilities, which define the tasks it can perform. We assume that the sensing and actuating skills of the robots are tied.

For robot $\mathrm{R_n}$, the set of capabilities it owns:

$$\text{Actuating capabilities: } A_n = \{a_1^n,\ \dots,\ a_r^n\}$$

$$\text{Sensing skills: } S_n = \left\{s_1^n,\ \dots,\ s_r^n\right\}$$

Also, each robot has set of constraints such as location, execution delay (speed of the robot), power (battery level), etc.

For robot R_n, the set of constraints it has:

$$CS_n = \{CS_1^n, \ . \ . \ . \ , CS_r^n \ \}$$

Also, each robot has three states: Idle, Allocated, and Busy. The Idle state is when the robot is not performing any task, the Allocated state is the state where the robot is locked and the algorithm is running on it, and the Busy state is the state when the robot is performing a task.

The infrastructure can perform m tasks assigned to it:

$$T \ = \ \{T_1, \ . \ . \ ., \ T_j, \ . \ . \ , T_m\}$$

And each task requires a specific set of sensing skills and/or set of actuating capabilities to start. We represent the requirements of each task by two vectors; sensing requirements and actuating requirements.

For task m, the set of requirements it has:

$$S_m = \{s_1^m,\ \dots,\ s_r^m\}$$

$$A_m = \{a_1^m,\ \dots,\ a_r^m\}$$

Each task has also set of constraints such as location, execution delay, etc.

For task T_m the set of constraints it has:

$$CS_m = \{CS_1^m, \ldots, CS_r^m\}$$

We will consider the situation where each task should be attached to a group of robots which will perform the task. This is important when tasks cannot be performed by a single robot because of the limited capabilities of a robot, and requires the cooperation between several robots. Thus only a set or subset of robots that have the required skills and capabilities can be assigned to that task. We call these sets of robots coalitions, where a single robot may be unable to complete a task; the collective abilities of a coalition may meet the task requirements.

Now our problem can be viewed as a coalition formation problem which is to select the optimal set of coalition of robots with respect to the task requirements and coalition value V.

A coalition CLT has two vectors of capabilities; each one is the sum of the capabilities owned by the group of robots in that coalition:

$$\text{Sensing skills capabilities: } S_c = \sum_{R_i \in C} S_i$$

$$\text{Actuating capabilities: } A_c = \sum_{R_i \in C} A_i$$

Coalition CLT^m can perform task T_m only if the vector of its capabilities fulfills the requirements S_m and A_m of task T_m:

$$\forall 0 \le i \le r$$

$$S_i^c \ge s_i^m \qquad \text{And/or} \qquad A_i^c \ge a_i^m$$

We assume that a coalition can work on a single task at a time and each robot is a member of one coalition at a time:

$$C1_i \cap C2_i = \emptyset$$

5.2.2 Algorithm Overview

We propose an algorithm for Network Level Virtualization, which will select the most appropriate group of robots for a given task. Our algorithm consists of Filtering and Ranking methods which will be discussed in the next subsection, and MOPSO approach which will also be discussed in the next subsection. Filtering and Ranking methods are used to choose the best robots that meet our goals in order to make the robots that have not been chosen available for other requests, and also in order to run the MOPSO algorithm on the chosen robots only. MOPSO is used to find the best coalition.

When a request comes to the IaaS from the PaaS, the Request Handler at the Network Level Virtualization Layer gives it to the Virtualization Engine with a set of inputs, the Virtualization Engine starts running the algorithm. The set of the inputs is:

- n: Maximum number of robots needed and allowed in one group
- t: Time needed to perform a given task
- c: Cost agreed with the customer
- $Criteria\_Importance$: To rank the robots
- $Filtering\_Rule$: The rule on which the robots will be filtered
- $Sensors$ and $actuators$ needed for a task

The Virtualization Engine executes one task at a time. If multiple requests come at the same time, they are executed in the same order as they arrived; First In First Out (FIFO) method. Which means the second request will be blocked until the first request has fully completed.

The algorithm first starts with filtering the robots based on the filtering rule, and then ranking them based on the Promethee II Ranking method. After that it selects from the ranked robots the first n robots and calculates all possible coalitions of size n or less: Less, because the infrastructure may include some robots more powerful than the others and which are capable of performing the function of several robots, thus the algorithm tries to reduce the number of robots in a coalition keeping in mind to meet the cost threshold. The number of coalitions that will be calculated is $\sum_{r=1}^{n} \binom{n}{r}$.

After calculating all possible coalitions, the algorithm selects the coalitions that satisfy the task requirements, and tries to find the optimal coalition using the Multi-Objective Particle-Swarm-Optimization (MOPSO) algorithm, where the latter gives set of solutions, we use Promethee II ranking in order to choose the most effective coalition.

Algorithm 1, describes the main steps in our proposed algorithm. Since the algorithm runs on all the robots belonging to all IaaSs, we added a locking system to the algorithm to lock the robots that are being used in order to prevent other IaaSs from using them in the case they receive a request. After filtering the robots, the selected robots are put in Allocated state in order to lock them (Steps 5 to 7), and then the algorithm performs three release procedures at different steps to release the robots that were not selected, and make them available for other requests.

The first release procedure is done after ranking and selecting the first n robots, the robots that were not selected by the filtering function are released and put back to Idle state (Steps 12 to 15). The second release is done after finding the coalitions that do not satisfy the task requirements, where the robots belonging to these coalitions are put back to Idle state (Steps 23 to 26). The last release procedure is done after finding the best coalition or the optimal solution, where the robots that do not belong to the best coalition are put back to Idle (Steps 33 to 36). Finally the robots belonging to the best coalition are put to Busy state at Steps 30 to 32.

Algorithm 1: Main Algorithm

```
1.  Initialize n, t_Task, t_Location, cost, Filtering_Rule, Criteria_Importance
2.  Set Selected_Robots = [ ], Idle_Robots = [ ], bestValue = ∞, Selected_Coalitions = [ ],
    Selected_Partitions_List = [ ], bestCOST = ∞
3.  Function: Filtered_Robots = Filter_Robots (Idle_Robots, Filtering_Rule)
4.          Selected_Robots  ⟵  Filtered_Robots
5.  foreach (Robot in Selected_Robots)
6.          Set Robot.State ⟵  allocated
7.  end for
8.  Function: Ranked_Robots = Rank (Selected_Robots, Criteria_Importance, t_Location, t_Task,
    cost)
9.  for i = 1 to number of the  Ranked_Robots
10.         do    Selected_Robots  ⟵   Robot[i]
11.         while i ≤ n
12.         do
13.               set Robot[i].state ⟵  idle
14.               Idle_Robots ⟵ Robots[i]
15.         while i> n
16. end for
17. Calculate all possible CLTs_List in Selected_Robots
18. foreach (Coalition in CLTs_List)
19.     if sensors(Coalition) ≥ sensors(task)  & actuators(Coalition) ≥ actuators(task) then
20.                 Selected_Coalitions  ⟵  Coalition
21.     end if
22.     else if sensors(Coalition) ≤ sensors(task)  or actuators(Coalition) ≤ actuators(task)  then
23.                 foreach(Robot in Coalition)
24.                     Set Robot.State ⟵  Idle
25.                     Idle_Robots   ⟵Robot
26.                 end for
27.     end if
28. end for
29. Function: Best_Coalition = MOPSO(Selected_Coalitions, t, c)
30. foreach (Robot in Best_Coalition)
31.         set Robot.State  ⟵ busy
32. end for
33. foreach (Robot doesn't belong to Best_Coalition)
34.       set Robot.State  ⟵  Idle
35.       Idle_Robots ⟵  Robot
36. end for
```

5.2.2.1 Filtering and Ranking Methods

Algorithm 2, explains the filtering function. We proposed a simple filtering method. In this method, if the battery level of the robots is lower than the $Filtering\_Rule$, they are excluded from next steps. The $Filtering\_Rule$ is an input of the algorithm which comes from the PaaS. Algorithm 3, describes the ranking function that we adopted. To rank each robot, we used Promethee II method [60].

The Promethee II is well suited for multi-criteria ranking problems. It is based on pairwise comparison of the alternatives, which are the robots in our case. It considers the difference between the evaluations of two alternatives over each criterion. The alternatives are the robots in our case, and the criteria are the time needed for each robot to perform a task, and the cost of the robot deployment. Algorithm 3 describes our ranking method, it is inspired by the reference [61]. Before starting to rank the robots, the Criteria-Importance is evaluated at Steps 3 to 9.

First, the deviations between the evaluations of the alternatives (R_i and R_j) within each criterion should be calculated. This is presented at Step 13.

$$d_k(R_i, R_j) = criteria_k(R_i) - criteria_k(R_j)$$

Then, the preference function should be calculated (Steps 14 to 18); we used Usual Criterion which is a common generalized criterion for all of the criteria. It is based on the following rule.

$$P(d) = \begin{cases} 0 & d < 0 \\ 1 & d > 0 \end{cases}$$

Using the generalized criteria, aggregated preference indices are calculated to express with which degree R_i is preferred to R_j over all the criteria.

$$\pi(R_j, R_i) = \sum\nolimits_{j=1}^{k} P_j(R_j, R_i) w_j$$

The w_j, is the weight for each criteria which defines the importance of the criteria, we defined the same importance for both criteria.

$$\sum\nolimits_{i=1}^{k} w_i = 1 \qquad k = numberofcriteria$$

The next steps are the calculation of the outranking flows (Steps 21 and 22).

The positive outranking flow; shows how alternative R_i outranks other alternatives.

$$\emptyset^{+}(R_i) = \frac{1}{n-1} \sum_{R_j \in R_m} \pi(R_j, R_i)$$

The negative outranking flow; shows how alternative R_i is outranked by the others.

$$\emptyset^{-}(R_i) = \frac{1}{n-1} \sum_{R_j \in R_m} \pi(R_i, R_j)$$

And finally, the net outranking flow is calculated to create a complete ranking of the alternatives (Step 23).

$$\emptyset(R_i) = \emptyset^{+}(R_i) - \emptyset^{-}(R_i)$$

At Step 24 the alternatives or the robots are ranked from the highest to the lowest and the first n robots are chosen to be included in the next steps.

Algorithm 2: Filtering Function

```
Function Filtered_Robots = Filter_Robots (Idle_Robots, Filtering_Rule)
if   Battery_Level (Idle_Robots ) ≥ Filtering_Rule
    Filtered_Robots  ⟵ Idle_Robots
end if
```

Algorithm 3: Ranking Function PROMETHEE II

```
Function Ranked_Robots = Rank (Selected_Robots, Criteria_Importance, t_Location, t_Task, cost)
Initialize Criteria_List
if Criteria_Importance = 0
        both criteria have the same importance
else if Criteria_Importance = 1
        it is time critical, time is more important than cost
else if Criteria_Importance = 2
        it is cost critical, cost is more important than time
end if
foreach (Robot in Selected_Robots)
    foreach (criteria in Criteria_List)
        Compare with other robots in Selected_Robots for each criteria
        $d_k\ (R_i, R_j) = criteria_k(R_i) - criteria_k\ (R_j)$
        if d $(R_i, R_j) \geq 0$
            Preference $(R_i, R_j) = 1$
        else if d $(R_i, R_j) < 0$
            Preference $(R_i, R_j) = 0$
        end if
    end for
end for
Calculate flow_plus (Robot)
Calculate flow_minus(Robot)
Ranking = flow_plus (Robot) - flow_minus(Robot)
Sort the robots from highest ranking to lowest
```

5.2.2.2 MOPSO Algorithm

Algorithm 4 describes the MOPSO algorithm. First it initializes the particles' positions, which are the positions of the coalitions that met the task requirements (Steps 19 to 21 in Algorithm 1). It also initializes the velocity of each particle and the target randomly. We assume that the target is within the search space of each particle.

First, a primary evaluation of the particles is done at step 5; Algorithm 5 explains the Evaluate Population function. At this step, the time needed by each particle to reach the target is calculated, the cost of the robots involved in the particle is calculated, and the number of robots involved in each particle is also calculated. At Step 6, the values of the particles that represent non-dominated vector are stored in repository called REP. At Step 9, iterations start. At each iteration, the velocity and the position equations are updated, and the population is evaluated again. The $pBEST[i]$ is the best position a particle has had, and $REP[j]$ is a value taken from the repository randomly.

If we have more than one Pareto optimal solution at the end of the iterations, we select among them based on the Promethee II ranking method giving the same importance (weight) for all criteria (time, cost, and number of robots). The selection of the best particle is done after removing the particles that exceed the time and cost thresholds, in other words the time and the cost the IaaS received from the PaaS.

Algorithm 4: Function MOPSO

```
1.  Function Best_Coalition = MOPSO(Selected_Coalitions, t, c)
2.  Initialize the position and the velocity of each particle: par[i], vel[i]
3.  Initialize the Target randomly
4.  Initialize number_of _iteration
5.  Function: value = Evaluate Population(par)
6.  Store the position of particles that represents non-dominated vector in repository REP
7.  Initialize memory for each particle
8.  pBESTS[i] = par[i]
9.  while ( number_of _iteration is not reached) do
10.     foreach particle
11.         Update velocity and positions equations
12.         vel[i] = C*(w*vel[i] + c1*r1*(pBEST[i] – par[i]) + c2*r2*(REP[j]-par[i])))
13.         par = par + vel
14.         Function: value = Evaluate population(par)
15.         Update the contents of REP
16.         if we have more than one pareto solutions then
17.         Rank the pareto optimal solutions based on Promethee II Ranking
```

```
            Check the time and cost thresholds $(t, c)$ for each particle
            if  t_particle> t   or  c_particle> c
                    Remove the particle from pareto optimal solutions
            end if
            Select the particles with highest ranking
                        Best_Coalition $\longleftarrow$ Particle with highest ranking
            else
                        Best_Coalition $\longleftarrow$ Best_Particle
            end if
        end for
    end while
```

Algorithm 5: Function Evaluate Population

```
Function value = Evaluate population(par)
for each particle
    time_obj = |particle_position – Target| / vel
    cost_obj = sum of costs of robots in that coalition
    n_obj = total number of robots in one coalition
end for
```

5.3 Requirements Met by the Algorithm

The proposed algorithm satisfies all the requirements that we mentioned in Chapter 3. First, the proposed algorithm is multi-criteria since it is taking into consideration static criteria such as a robot’s sensors and actuators, and dynamic criteria such as a robot’s position and battery level. Also, it optimizes multiple objectives in the presence of trade-offs between them: minimizing the number of robots involved in coalition, minimizing the cost of robot deployment and minimizing the time needed to perform a task.

The proposed algorithm eliminates the communication among the robots. There is no communication between the robots. The Virtualization Engine runs the algorithm, selects the most suitable group of robots and sends the task to them. The next requirement, the ability of the algorithm to select subset of robots for a given task, is also satisfied. For each task, the algorithm selects a group of robots. The proposed algorithm uses Filtering and Ranking methods. For the Filtering method a simple method was used, for the Ranking method the Promethee II ranking was used.

The algorithm has a good response time. The speed of the solution produced by Heuristic algorithms such as PSO is sufficient. The solution may not be the best among all solutions, but it requires a short time to produce it. Also the response time of the algorithm is calculated and shown in Chapter 6.

The resources, which are the robots, are minimized in the defined algorithm. There is maximum number of allowed robots for each coalition; the algorithm tries to minimize this number as long as the group meets the task requirements. There is also a maximum allowed cost for the robots involved in a coalition; the algorithm tries to reduce this cost by finding the coalition with the minimum cost. Finally, the time needed to perform a task is minimized; there is a maximum allowed time for each coalition to perform each task. The algorithm tries to find the coalition that needs the least time to perform a task.

5.4 Illustrative Scenario

Let us consider a Wildfire Suppression Robotic application that detects and suppresses wildfires using a fleet of heterogeneous robots deployed in the forest. If a fire is detected, the application at the PaaS level evaluates its intensity and rate of spread, and sends the request to the IaaS to deploy the most appropriate robots to suppress the fire with set of inputs:

- n = 50 robots

- $cost$ = 500$
- t = 10 minutes
- $Filtering\_Rule$ = select robots with battery level >70%
- $Criteria\_Importance$ = 1; which means it is time critical request
- Task requirements:

  Sensing skills requirements = $\{S_1, S_2, S_3\}$

  Actuating capabilities requirements = $\{A_1, A_2, A_3\}$

Also, the total number of robots available in the IaaS is 1000 robots. At the Network Level Virtualization Layer, the Virtualization Engine receives the request; it asks the Task Delegator Engine for all idle robots belonging to all the IaaSs. When the Virtualization Engine receives this information, it should select and dedicate the most appropriate group of robots or the coalition for this task.

Starting with the filtering procedure, we assume that 100 robots meet the $Filtering\_Rule$, the other robots are released.

For each robot we form the following matrix:

$$R_1 = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ S_1 \\ S_2 \\ S_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_L & t_C & cost_L & cost_C \end{array} \\ \left[\begin{array}{cccc} \quad & \quad & \quad & \quad \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \end{array}\right] \end{array}$$

- A_i: i^{th} actuating capability for Robot 1
- S_i: i^{th} sensing capability for Robot 1

- t_L: time needed to go to the required location (this column is the same in all rows)
- t_C: time needed to perform the task using corresponding capability (sensor and actuator)
    - If two capabilities are used at the same time we do: $t_C = t_{C1} \times t_{C2}$
    - If two capabilities are used one after the other: $t_C = t_{C1} + t_{C2}$
        - Note: C_i can be either A_i or S_i
- $cost_L$: the cost of the robot to arrive to the location of the event (this column is the same in all rows)
- $cost_C$: the cost of using the corresponding capability

Next, this 100 robots will be ranked, giving higher priority or weight to the time.

$$Ranking = \begin{array}{c} R_1 \\ R_2 \\ . \\ . \\ . \\ R_{100} \end{array} \begin{bmatrix} Ranking_1 \\ Ranking_2 \\ . \\ . \\ . \\ Ranking_{100} \end{bmatrix}$$

After sorting this ranking matrix from highest to lowest, the algorithm selects the first 50 robots, and calculates all possible coalitions. The number of coalitions calculated is $\sum_{r=1}^{50} \binom{50}{r}$. The algorithm then selects the coalitions that meet the task requirements, and starts MOPSO algorithm, which finds the most optimal coalition for the task by satisfying three objectives; minimizes V: minimize number of robots in coalition, minimize the cost of robots deployment, and minimize the time needed to perform a task by a coalition.

After selecting the optimal coalition, the Virtualization Engine sends a task assignment request to the external robots through the Overlay, and to the local robots through the Resources Layer.

5.5 Chapter Summary

In this chapter, we presented our proposed algorithm for Network Level Virtualization to select the most appropriate or the optimal group of robots for a given task. We started first with background information on heuristic algorithms, where we gave examples such as ACO, GA, and PSO. We explained in more detail the PSO, since it is the algorithm used in the proposed algorithm. Also we gave background information on Multi-Objective Optimization Problems, since we tried in our proposed algorithm to optimize multiple objectives at the same time in the presence of trade-offs between them. Finally, we gave background information on the Promethee ranking method, which is multi-criteria ranking method and we used it in the algorithm in order to reduce the number of robots on which the MOPSO runs, and also we used it to select among the Pareto optimal solutions that MOPSO produces. Then, in the second section we started presenting our algorithm, we discussed first the assumptions and the problem statement, then we gave an overview of the proposed algorithm, and finally we presented our algorithm in details with the Pseudocode, where we used Ranking and Filtering methods. Furthermore, we presented the requirements met by the proposed algorithm. Finally, we gave an illustrative scenario in the last section using a real world scenario.

Chapter 6

6 Validation: Prototype, Simulation and Evaluation

We start this chapter by presenting the overall prototype architecture. The second section describes the prototype architecture in detail. The third section includes the prototype setup and the performance measurements of the prototype. The following section includes the simulation environment setup and the performance measurements of the algorithm. Finally, we summarize the chapter.

6.1 Overall Prototype Architecture

In this subsection, we will start by presenting the implemented scenario. Then, we give a high-level description of the prototype. Finally, we describe the software tools we used in the implementation.

6.1.1 Implemented Scenario

As a prototype, we implemented one of the scenarios presented in Chapter 3, the Wildfire Suppression scenario. We consider a forest where wireless sensor networks have been deployed to monitor wildfires. A Wildfire Suppression Application manages the wireless sensor network. When there is a fire breakout, the Wildfire Suppression Application sends a notification to the IaaS to deploy the most appropriate robots to suppress the fire. The robots have different capabilities, some are equipped with arms which allow them to grab the extinguishers and suppress the fires, others are equipped with sensors which allow them to detect obstacles and eventually remove them. These robots belong to different clouds, where each robot is managed by the IaaS that it belongs to it. The Wildfire Suppression Application is located on a dedicated cloud.

6.1.2 Prototype High Level Description

Figure 6-1 shows the GUI of the Wildfire Suppression Application which enables a request to be sent to the IaaS. We used three buttons. The first button is “Send a Request”, when the user clicks this button, an HTTP request is sent from the PaaS to IaaS. The IaaS selects a group of robots in a cost efficient manner in the Network Level Virtualization Layer to suppress the fire. The PaaS receives as a response from the IaaS the request Id which appears on the “REQUESTS” window. To know the state of a request, the user selects the request Id on the “REQUESTS” window and clicks the second button “Get State of a Request”. In this case another HTTP request is sent to IaaS and the state of the request appears on the “REQUEST STATE” window. To cancel a request in the middle of the execution, the user clicks the “Cancel a Request” button after selecting the request that needs to be cancelled on the “REQUESTS” window. By clicking this button, another HTTP request is sent to the IaaS.

We used three robots with different capabilities. The first robot is equipped with arms that allow the robot to grab the extinguisher, and thereby suppress the fire. We used plastic balls instead of real extinguishers. The second and third robots can detect obstacles. We used red color objects as obstacles that can be detected by the robots’ light sensors. These two robots can also remove the objects using the hands attached to their motors. We assume each of these three robots belong to a different business entity. This leads to an implementation with three clouds, one robot per cloud. One of the clouds hosts the Wildfire Suppression Application in addition to the IaaS to which its robot belongs, while the other two clouds only host the IaaS to which their robots belong. The communication with the robots is done through the Gateway in the Resources Layer via Bluetooth.

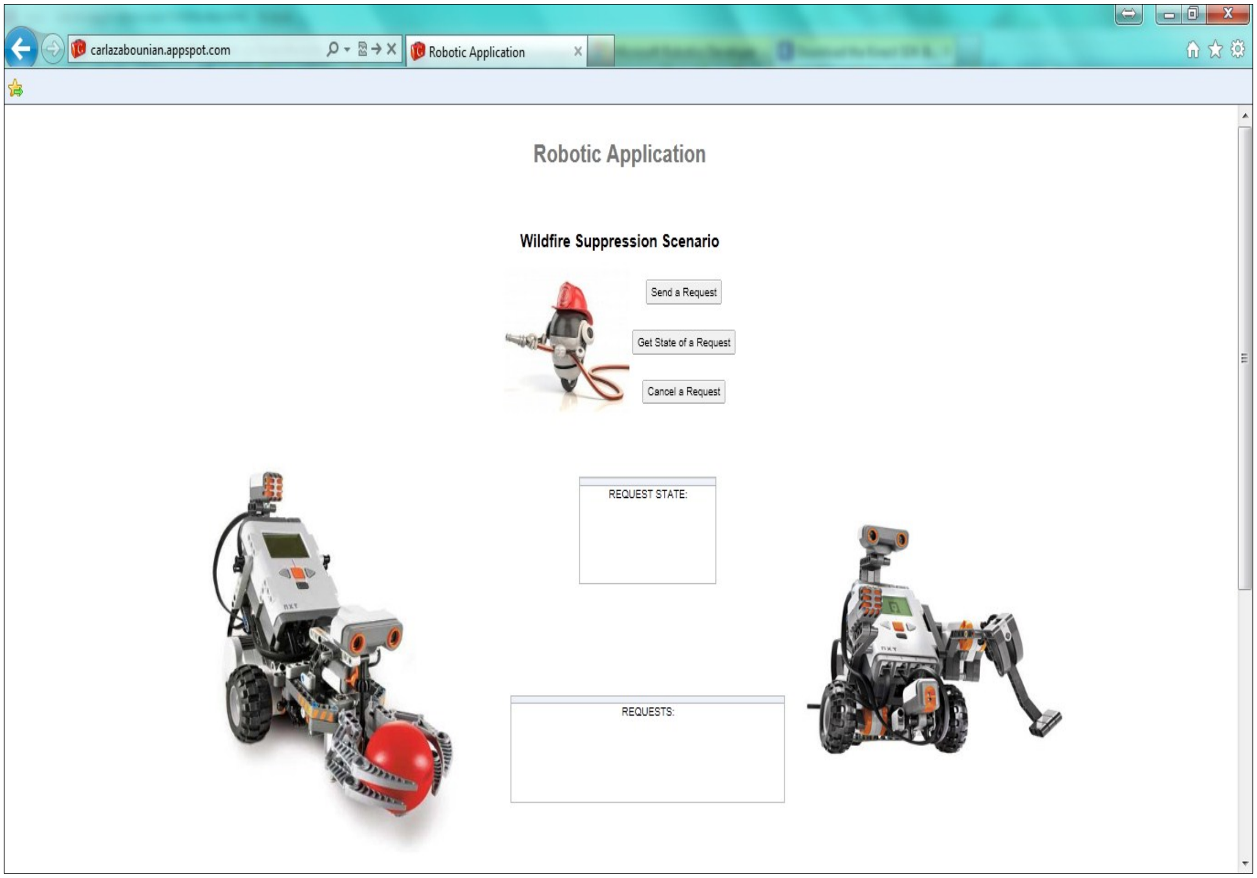


Figure 6-1 The Wildfire Suppression Application End User Interface

6.1.3 Software Tools

In this subsection we will describe the software tools that we used to implement the prototype shown in Figure 6-4.

6.1.3.1 Google Apps Engine

We used Google App Engine (GAE) to implement the Wildfire Suppression Application. GAE is a Platform as a Service (PaaS) cloud computing platform used for developing and hosting web applications on Google's Infrastructure. GAE is offered as pay-as-you-go. The user only needs to upload an application, and it will be ready to be used by others. A user can serve her application on free domain (appspot.com) or on specified domain. GAE supports several programming languages such as Java, Python, PHP, and Go. It is free up to a specified limit: 1 GB of storage and bandwidth to serve five million page views per month. The free limits are raised once the user enables billing for her application and the user pays only when she exceeds the free levels. The user does not need to maintain any server, because by simply uploading the application, Google will host and scale it, and the application will be ready to serve the users. The user can also determine if the access to the applications is public or restricted to a limited number of members. We chose the GAE platform because it is free and easy to deploy and maintain applications [10].

6.1.3.2 LEGO Mindstorms NXT Robots

We used LEGO Mindstorms NXT robots as the robot fleets. LEGO Mindstorms NXT is a programmable robotics kit released by LEGO in July 2006. They give the power to create and command your own LEGO robots. LEGO Mindstorms NXT robots can take input from up to four sensors, and control up to three motors using RJ12 cables which are similar to RJ11 phone cords. They are widely used in educational institutes and they come with a graphical programming environment. LEGO Mindstorms NXT robots come in modular blocks that have to be assembled and they can only communicate via Bluetooth or USB. They have an NXT Brick which is an intelligent, computer-controlled LEGO brick which is the brain of the LEGO Mindstorms [62][63].

We used three LEGO Mindstorms NXT robots. One of them is type Tribot Figure 6-2, and the other two are the base type Figure 6-3. The Tribot type is a flexible and fast three-wheeled robot. It has four sensors, a sound, a touch, a light, and an ultrasonic sensor. It has also three servo motors, two motors that move the robot around and one motor that move the robot's arms to grab objects. This robot can be programmed to follow a line with Light Sensors, can feel objects with Touch Sensors, and/or see with Ultrasonic sensors. The Base type comes again with four sensors; sound, touch, light, and ultrasonic sensors. It is also equipped with three servo motors, two to move the robot around, and the third motor is used as an arm to remove or kick obstacles. We chose LEGO Mindstorms because of its easy programmability and its reusability where the same robot kit can be assembled into different robots.

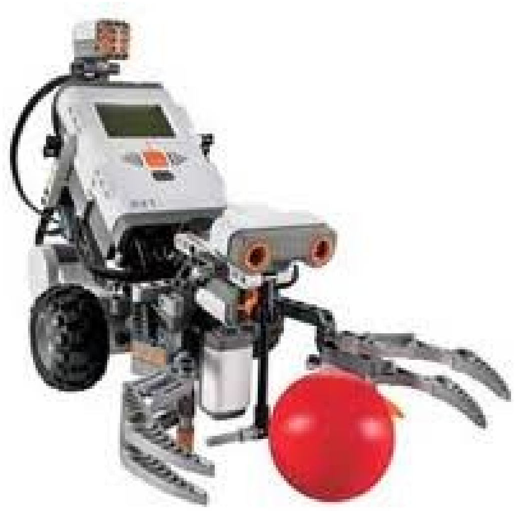


Figure 6-2 LEGO Mindstorms NXT Robot of Type Tribot

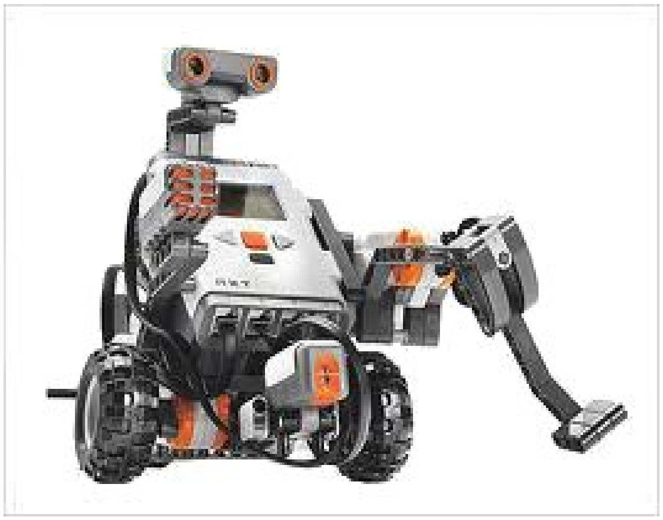


Figure 6-3 LEGO Mindstorms NXT Robot of Base Type

6.1.3.3 JXTA

JXTA is an open source project to create structured P2P overlay networks. JXTA is based on a set of open XML protocols; it can be implemented in different languages, such as JAVA SE, C/C++, and Java ME. We used JXSE 2.5, a Java-based implementation of JXTA to implement our overlay network. JXTA peers create a Virtual Overlay Network which allows a peer to interact with other peers even when some of the peers are behind firewalls and NATs or use different network transports. JXTA peers can be edge peers or super peers. Super peers can be either rendezvous or relay peers. An edge peer has low bandwidth network connectivity and resides on the border of the Internet, hidden behind firewalls. A super peer can enable edge peers to communicate with firewalls. Peers can communicate through pipes which can be either unidirectional or bidirectional. The peers and the pipes are described using XML files called advertisement [64].

6.1.3.4 Webots

Webots is a development environment used to program and simulate mobile robots. It is used by over 1097 universities and research centers worldwide. It is a professional mobile robot simulation software package.

Webots allows the user to create 3D virtual worlds. The robots can be wheeled robots, legged robots or flying robots, and they can be equipped with a number of sensors and actuator devices such as distance sensors, touch sensors, emitters, receivers, drive wheels, motors, etc. The robot controllers can be programmed with the built-in IDE or with third party development environments.

Webots is well suited for research and educational projects. It can be used for mobile robot prototyping, robot locomotion research, multi-agent research (Swarm intelligence, collaborative mobile robots groups, etc.), adaptive behavior research (e.g., genetic algorithm), teaching robotics, and robot contests. By using Webots a user can program a robot using six different languages: C, C++, Java, Python, Matlab, and URBI. Webots includes supervisor programming which can take screen-shots, make movies, record trajectories, move objects, change objects' properties while the simulation is running. It also has a rich Graphical User Interface and includes a large library of indoor and outdoor objects [65].

6.1.3.5 Additional Software Tools

The JAVA programming language is the programming language that is used to implement our prototype. We have used Eclipse IDE [66] for implementing the prototype.

The REST interfaces of the IaaS, the PaaS, and the Gateway were implemented with the Restlet framework [67][68]. We chose Restlet because it has the Java SE/EE edition that lets the user run Restlet applications on regular JVMs. It also has the edition for GAE, which lets the user develop Restlet applications on a GAE cloud computing platform. It provides a complete web server, where it can serve static files just like Apache HTTP, and it offers good scalability. An application that is developed with Restlet can play the role of HTTP server and HTTP client interchangeably without any need of change in the code of the application. Also it is simple and easy to use.

6.2 Detailed Prototype Architecture

Our prototype architecture is depicted in Figure 6-4. We implemented three IaaSs. We used Google App Engine as the PaaS Layer for Robotic Cloud 1. For the Network Level Virtualization Layer, we implemented all the functional entities that it involves. For the Resources Layer, the Virtualized Gateways have been excluded, and only a real Physical Gateway is implemented. We implemented a VRA node corresponding to each IaaS. Finally, we implemented the four procedures described in Chapter 4: Idle Robot Discovery, Selecting Robots for a Given Task, Subtask Assignment for the Selected Robots, and Notification of a Finished Subtask.

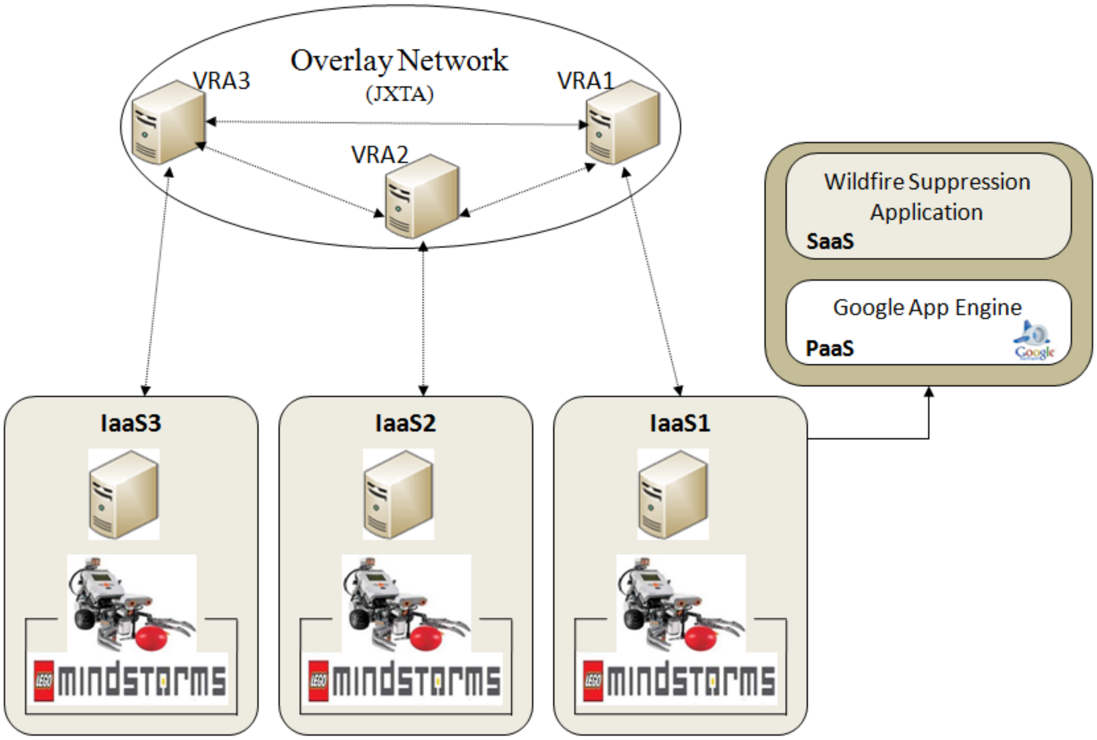


Figure 6-4 The Prototype Architecture

In the following subsections, we start by presenting the software architecture of the IaaS. Then, we describe the overlay nodes software architecture. Next, we describe the software architecture of the Wildfire Suppression Application.

6.2.1 The IaaS Layer

Figure 6-5 shows the software architecture of the IaaS Layer. The Virtualization Engine, the Robot Discovery Engine and the Task Delegator Engine were developed using JAVA classes and methods. The web service provider of the Request Handler and the Robot Monitoring Engine was developed as a RESTful web service using a Restlet framework. And the web service requester is a REST client.

We excluded the Virtualized Gateways in the implementation and we implemented only a Physical Gateway. The web service provider of the Physical Gateway was developed as a RESTful web service using a Restlet framework and the web service requester is a REST client. We implemented a Mapping Module which maps the REST commands into proprietary commands supported by the robot, in our case the robots support leJOS NXJ API, a Java programming environment for the LEGO MINDSTORMS NXT robot. If the robot is changed, the "Proprietary API for Robot" component should be changed.

We used a simplified version of the algorithm in the prototype at the Virtualization Engine component to perform Network Level Virtualization. The algorithm runs on all the idle robots belonging to the three IaaSs. The task which is "suppress the fire" requires a set of sensing skills and actuating capabilities: {two light sensors to detect the obstacles, one set of arms to grab the extinguisher, two kicking arms to remove the obstacles, and three movement motors}. This task cannot be performed by a single robot because of the limited capabilities of individual robots; it requires the cooperation of several robots. Thus, the algorithm selects a group of three robots for the given task based on their capabilities: Robot 1 with {one set of arms and one movement motor}, and Robots 2 and 3 each with {one light sensor, one kicking arm and one movement motor}. The sum of the capabilities owned by this group of robots fulfills the task requirements. Finally, the task is assigned to the selected robots.

When the Physical Gateway receives the subtask assignment for the selected robot from the Robot Monitoring Engine, it maps the REST commands to leJOS NXJ's commands through the Mapping Module, and sends it to the robot via Bluetooth. For the Physical Gateway monitoring Robot 1, the command on the Proprietary API for Robot is to go to location A and grab a ball (the ball simulates the extinguisher), then go to location B and release the ball. For the Physical Gateway that is monitoring Robot 2 and the Physical Gateway that is monitoring Robot 3, the command on Proprietary API for Robot is to detect the obstacle using the light sensor and to remove it using the kicking arm. For example, for grabbing and releasing a ball we implemented two methods called grab( ) and release( ), which instructs the robot to grab and to release the ball:

```
public void grab( )
{
    Motor.B.stop( );
    Motor.C.stop( );
    Motor.A.setSpeed(200);
    Motor.A.forward( );
    Delay.msDelay(500);
    Motor.A.backward( );
    Delay.msDelay(1000);
    Thread.sleep(500);
}
public void release()
{
    Motor.A.forward();
    Delay.msDelay(500);
    Motor.A.stop(); }
```

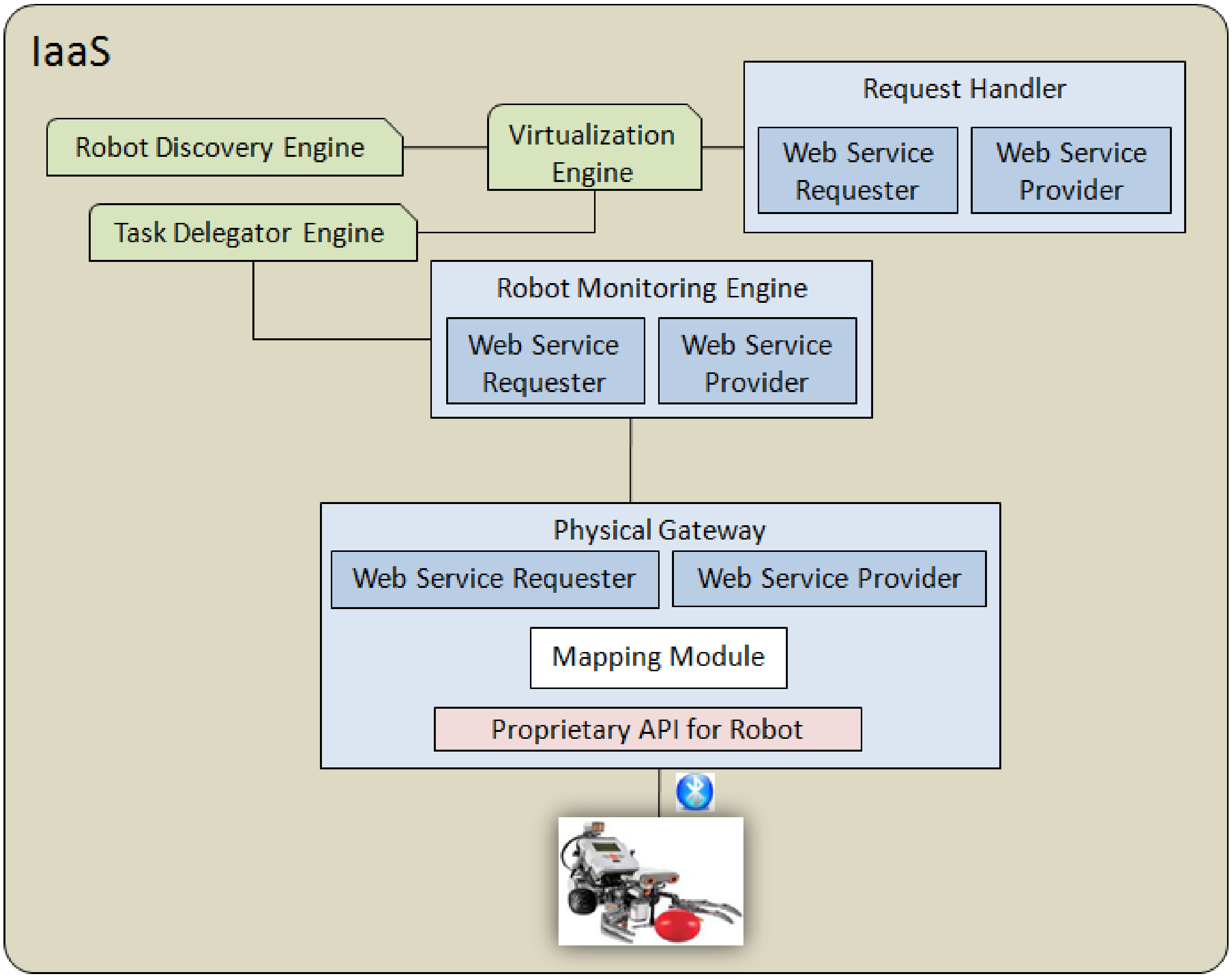


Figure 6-5 The Software Architecture of the IaaS

6.2.2 Overlay Network

Figure 6-6 shows the software modules of the VRA node. The Overlay Task Delegator and Overlay Robot Discovery components were developed using JAVA classes and methods. The web service provider of the Node Communication component was developed as a RESTful web service using Restlet framework. The web service requester is a REST client. The "Discover Request" message of the IaaS in the Overlay was carried out by JXTA advertisement. The "Task

Subscription" and "Task Notification" messages of the IaaS were mapped to JXTA messages that are exchanged through JXTA bidirectional pipes.

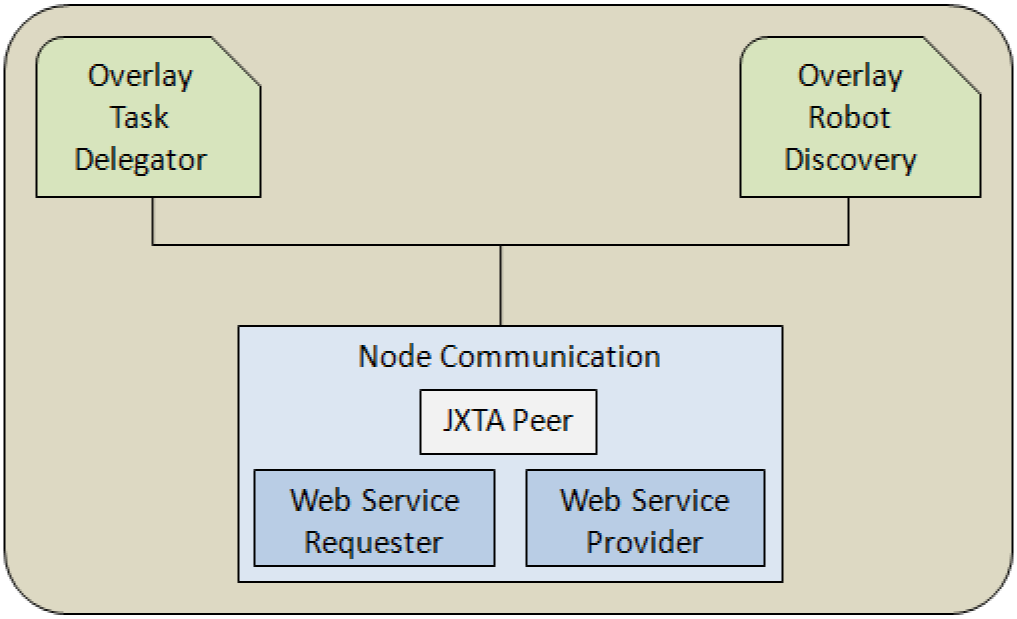


Figure 6-6 The Software Architecture of Overlay Nodes

6.2.3 The Wildfire Suppression Application

The Wildfire Suppression Application was developed using Google App Engine as the PaaS Layer. It contains a web service requester and a web service provider. The web service provider is used to receive requests/responses from the IaaS when the robots finish their assigned task. The web service requester is used to send requests to the IaaS when one of the buttons on the GUI (Figure 6-1) is pressed. The web service provider is developed as a RESTful web service using Restlet framework and the web service requester is a REST client.

6.3 Prototype Setup and Performance Measurements

In this subsection, we describe the networking choices we made and the details of the setup. Then, we present the performance measurements where we explain first the metrics that we used, and then give the results that we obtained.

6.3.1 Prototype Setup

Figure 6-7 shows the prototype setup. The Wildfire Suppression Application that sends the request is hosted on Google's Infrastructure. The VRA nodes are executed on laptops, each one on a different laptop. The three laptops are connected in the same Local Area Network (LAN). One of the laptops has two network interfaces, on one interface it has a public IP which can receive requests coming from the Wildfire Suppression Application, and the other interface is the LAN interface with a private IP.

We have developed a Network Address Translation (NAT) server as a Restlet application which redirects the requests coming from GAE to IaaS on our LAN.

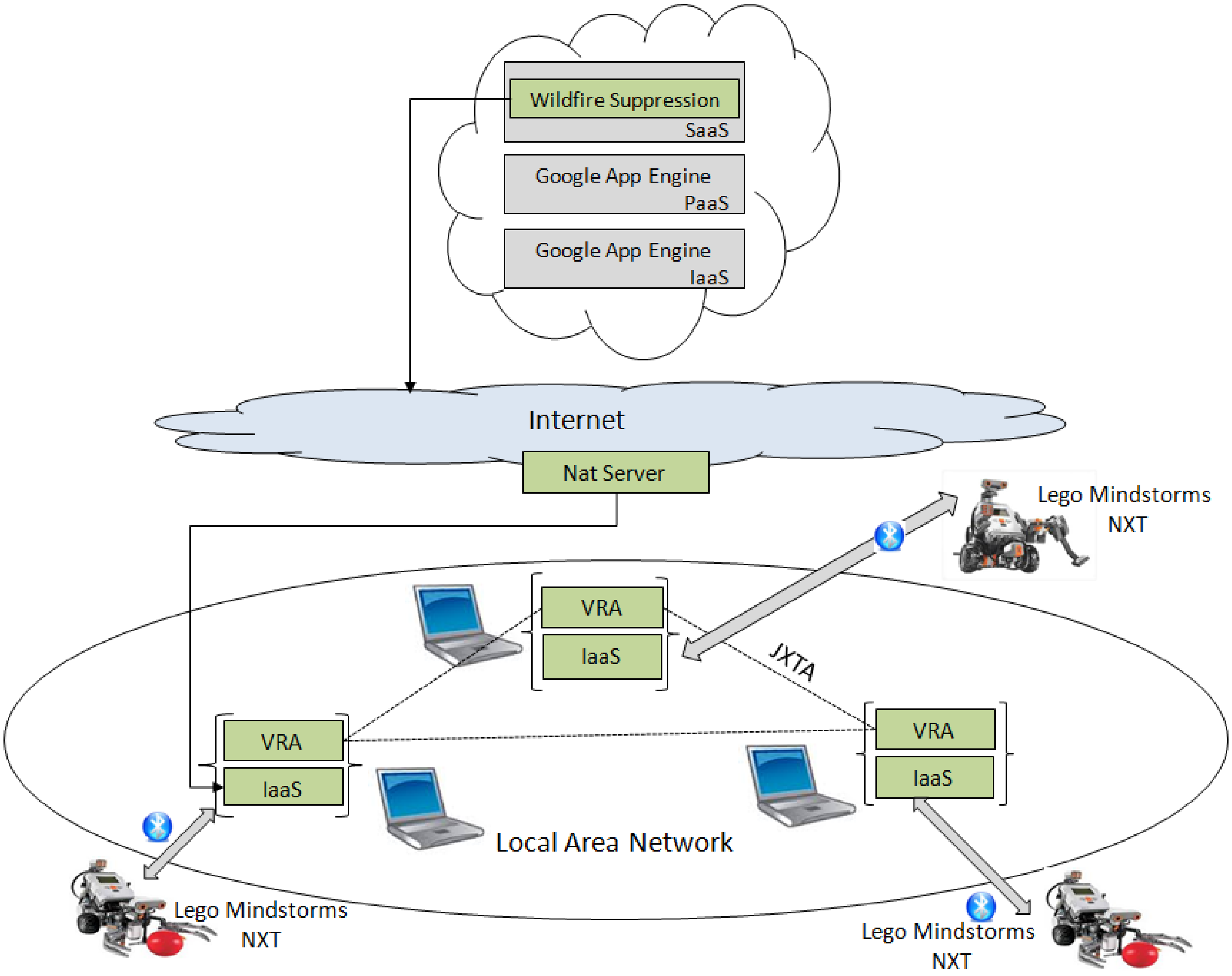


Figure 6-7 The Prototype Setup

6.3.2 Performance Measurements

The main purpose of the measurement is to proof the scalability of our proposed architecture in terms of Robots, and also in terms of several IaaSs. It is also to evaluate the time spent in the Overlay for the "Subtask Assignment for the Selected Robots" procedure.

6.3.2.1 Experimental Setup

The execution setup is the same as the prototype setup. We used three laptops; the first laptop executes the NAT server, the IaaS1, and the VRA1 node. The second laptop executes the IaaS2, and the VRA2 node. The third laptop executes the IaaS3 and the VRA3 node. In order to precisely evaluate our prototype and take measurements, we used a fourth laptop which executes the IaaS4 and VRA4 node. All laptops run with Windows 7 Professional. The first three laptops have an Intel® Core ™i5-2540 CPU with 2.60Hz, and 4 GB of RAM, the fourth laptop has Intel® Core ™i7-2620 CPU with 2.70Hz, and 8 GB of RAM. All laptops are connected through Ethernet cable to a Linksys router and they belong to the same LAN. The first laptop is also connected to the Internet through its wireless interface.

6.3.2.2 Performance metrics

We evaluated the prototype using two metrics: Idle Robot Discovery Delay (IRDD), and Neighborhood Assignment Delay (NAD). The IRDD is the time difference between the moment that the IaaS1 receives a request from GAE, and when it discovers the idle robots. Two types of IRDD are measured: The first type is with a different number of robots in each IaaS (100 robots, 500 robots, and 1000 robots), where we simulated the number of robots by modifying the Robot Information Repository. The second type is with a different number of IaaSs (two IaaSs, three IaaSs, and four IaaSs). The IRDD delay is measured to proof the scalability of the proposed architecture in terms of robots and also IaaSs. The NAD is the time difference between the moment that the IaaS1 sends a task assignment, and when the other IaaSs (IaaS2, IaaS3, and

IaaS4) receive the task assignment. The NAD delay is measured to evaluate the time spent in the Overlay for sending the task assignment request between the nodes in the Overlay.

6.3.2.3 Performance Results

The delays are measured in milliseconds and each result is calculated as an average of 20 experiments.

Figure 6-8 and Table 6-1 show the average time for IRDD using a different number of robots (100 robots, 500 robots, and 1000 robots). The average time for IaaS1 to discover 100 robots in IaaS2, IaaS3, and IaaS4 is 3260msec. The average time for IaaS1 to discover 500 robots in IaaS2, IaaS3, and IaaS4 is 3262msec, and the average time for IaaS1 to discover 1000 robots in IaaS2, IaaS3, and IaaS4 is 3265msec. We can observe that the values are increasing linearly as we add more robots in each IaaS. This shows the scalability of our system which is its ability to perform the discovery of the robots correctly as the number of robots increases.

Figure 6-9 and Table 6-2 show the average time for IRDD using a different number of IaaSs (2 IaaSs, 3 IaaSs, and 4 IaaSs). In the case of having two IaaSs, the average time for IaaS1 to discover the robots in IaaS2 is 101.85msec. In the case of having three IaaSs, the average time for IaaS1 to discover the robots belonging to IaaS2 and IaaS3 is 165.55msec. And in the case of having four IaaSs, the average time for IaaS1 to discover the robots belonging to IaaS2, IaaS3, and IaaS4 is 228.5msec. We notice that the values are again increasing linearly as we increase the number of IaaSs in the system. This shows the scalability of our system which is its ability to perform the discovery of the robots correctly as the number of IaaSs increases.

Figure 6-10 and Table 6-3 show the average time for NAD. The average time for IaaS1 to send the task assignment to IaaS2 is 163.66ms, the average time for IaaS1 to send the task assignment to IaaS3 is 199.58ms, and the average time for IaaS1 to send the task assignment to IaaS4 is 181.375ms. We observe that the average times for the three IaaSs to receive the task assignment message are very close. This is expected because they all use the same Overlay protocol to send a message from one Overlay node to another. These results show the viability of using an Overlay protocol for task assignment.

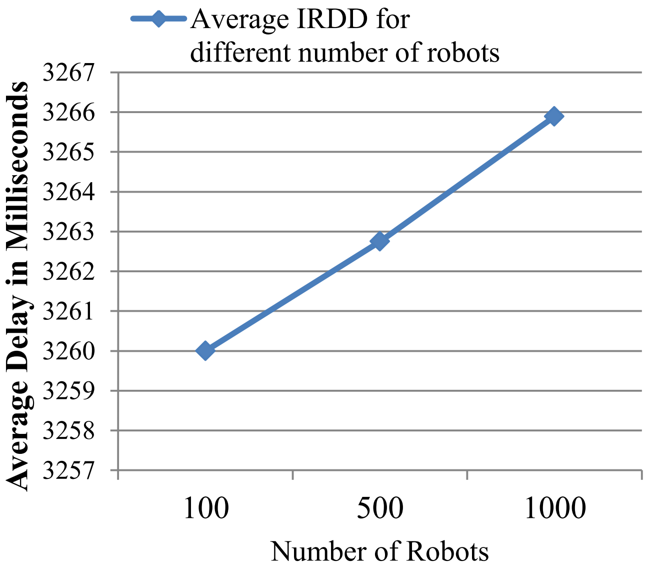
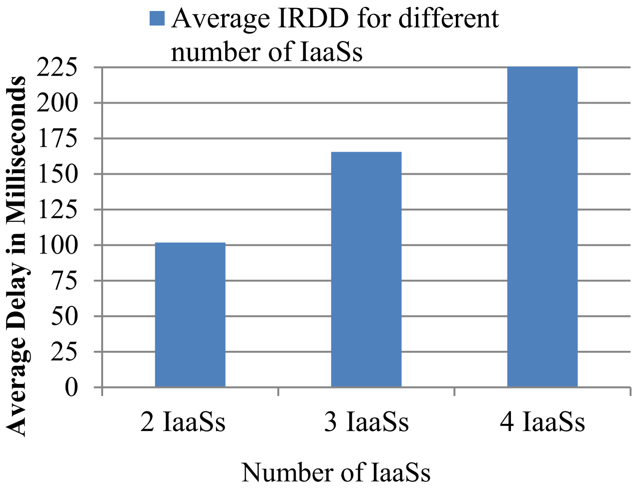


Figure 6-8 Idle Robot Discovery Delay (IRDD) with different Number of Robots

	100 Rs	500 Rs	1000 Rs
IaaS2 Rs	3260	3262	3265
IaaS3 Rs	3064	3045	3040
IaaS4 Rs	3118	3172	3175
All Rs	3260	3262	3265

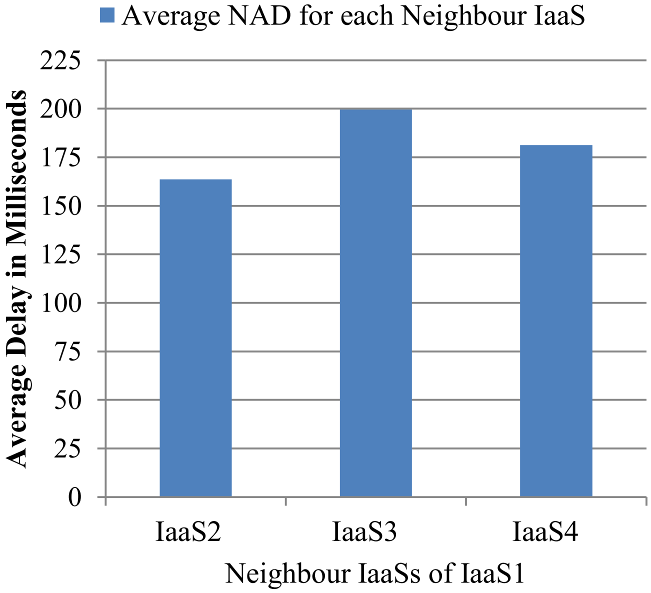
Table 6-1 Average Time (msec) for IaaS1 to Discover Different Number of Robots



	2IaaSs	3IaaSs	4IaaSs
IaaS2 Rs	101.85	165.55	228.5
IaaS3 Rs	-	42.33	27.5
IaaS4 Rs	-	-	122.5
All Rs	101.85	165.55	228.5

Table 6-2 Average Time (msec) for IaaS1 to Discover Robots with Different Number of IaaSs

Figure 6-9 Idle Robot Discovery Delay (IRDD) with Different Number of IaaSs



	IaaS1
IaaS2	163.66
IaaS3	199.58
IaaS4	181.26

Table 6-3 Average Time (msec) for IaaS1 to Send Task Assignment to 3 IaaSs

Figure 6-10 Neighbourhood Assignment Delay (NAD)

6.4 Simulation Environment and Performance Results

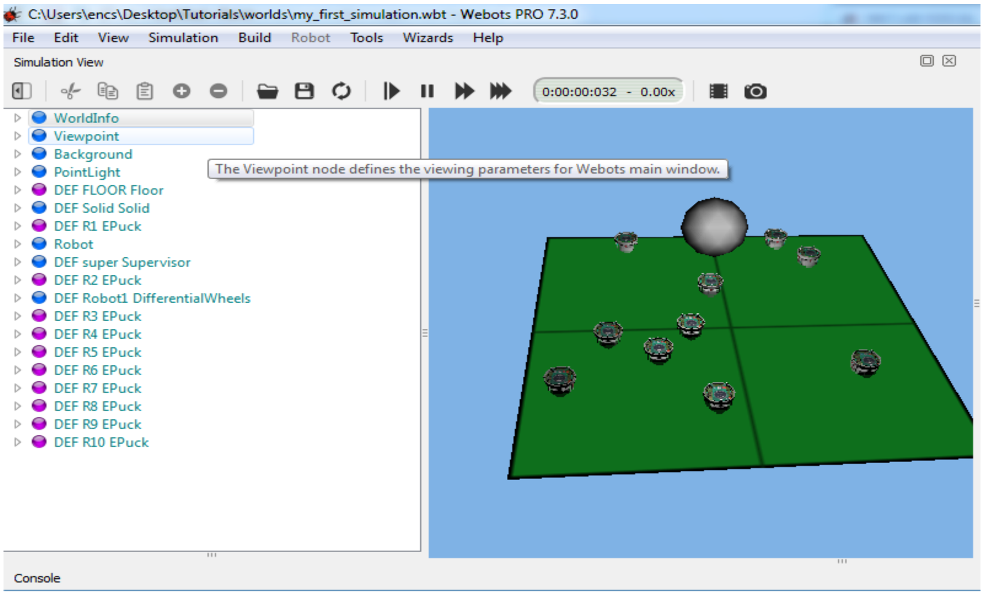
In this subsection we will present first the simulation environment, then we will explain the performance metrics, and finally we will show our results. The main purpose of the performance result is to show the Processing Time of our algorithm, and to prove that our algorithm finds the best coalition among all coalitions.

6.4.1 Simulation Environment

We used Webots to simulate our algorithm. We did three experiments each with two setups. In the first experiment we put 10 robots, in the second experiment we put 15 robots, and in the third experiment we put 20 robots. In the first setup of each experiment the maximum number of robots allowed in each group is n=3, and in the second setup the maximum number of robots allowed in each group is n=6.

In each experiment the speed, the cost, the position, the battery level of each robot, and the position of the target - which is the fire location - are generated randomly. All the robots are in Idle state at the beginning of each experiment, and each robot has two sensors and one motor to move the wheels of the robot. The algorithm takes these values as input, and starts running to select the most efficient group of robots. After finding the best coalition, the algorithm sends the task to the robots and the robots move toward the target. The algorithm was implemented in Matlab.

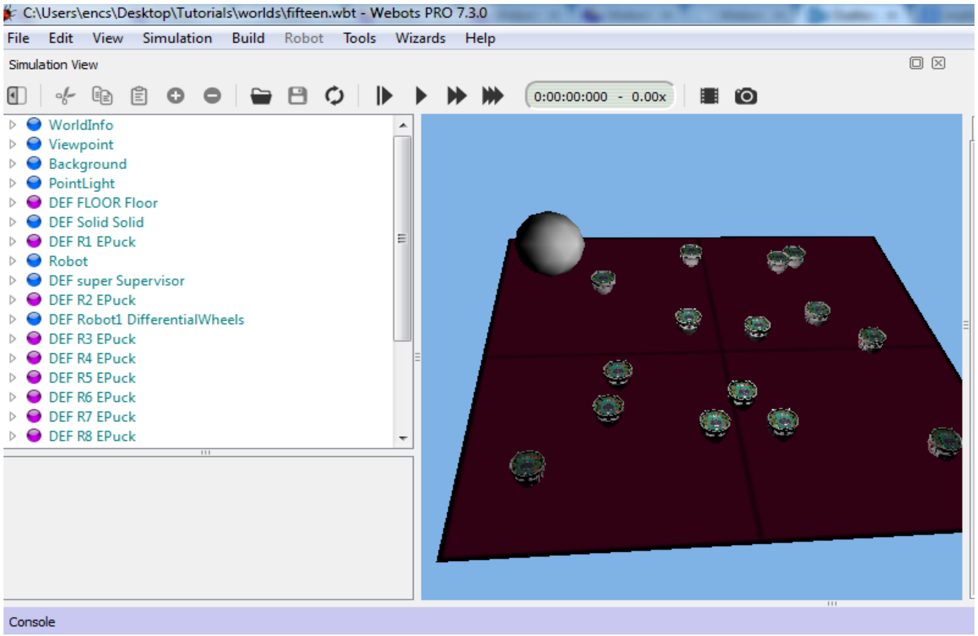
Figure 6-11, 6-12, and 6-13 shows the simulation environment with 10, 15, and 20 robots respectively.



C:\Users\encs\Desktop\Tutorials\worlds\my_first_simulation.wbt - Webots PRO 7.3.0

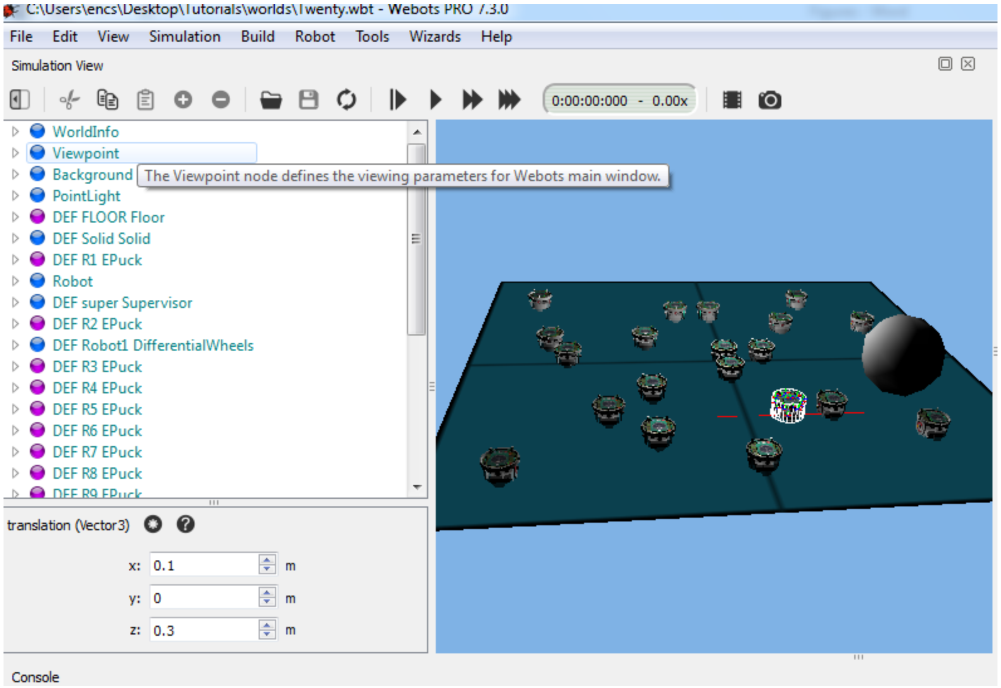
The Viewpoint node defines the viewing parameters for Webots main window.

Figure 6-11 The Simulation Environment with 10 Robots



C:\Users\encs\Desktop\Tutorials\worlds\fifteen.wbt - Webots PRO 7.3.0

Figure 6-12 The Simulation Environment with 15 Robots



C:\Users\encs\Desktop\Tutorials\worlds\Twenty.wbt - Webots PRO 7.3.0

The Viewpoint node defines the viewing parameters for Webots main window.

translation (Vector3)

x: 0.1 m

y: 0 m

z: 0.3 m

Figure 6-13 The Simulation Environment with 20 Robots

6.4.2 Performance Metrics

We evaluated the algorithm using two metrics: The Processing Time of the Algorithm (PT), and the Best Coalition (B-CLT). Two types of PT were calculated: the first type includes the communication with the robots, which obtains the information from the robots (speed, position, battery, etc.), performs the algorithm, and finally sends the task to the chosen robots. The second type does not include any communication with the robots; it is only the Processing Time of the algorithm. In both types we calculated PT with two setups, one with the maximum number of robots allowed in a group n=3, and the second with maximum number of robots allowed in a group n=6. The B-CLT shows the best coalition that the algorithm selects at the end of its execution where the objective is to minimize the combination of time needed to perform a task, cost of robot deployment, and number of robots per group at the same time.

6.4.3 Performance Results

The delays for the PT are measured in seconds and for the B-CLT in milliseconds. Each result is calculated as an average of 10 experiments.

Figure 6-14 and Table 6-4 show the Processing Time of the algorithm including the communication with the robots. When n=3 for 10 robots the PT is 0.484sec, for 15 robots the PT is 1.468sec, and for 20 robots the PT is 3.230sec. When n=6 for 10 robots the PT is 0.613sec, for 15 robots the PT is 1.760sec, and for 20 robots the PT is 3.518sec. We can see the Processing Time increases as the number of robots in the environment increases. This is expected, since the algorithm will communicate with a larger number of robots.

Figure 6-15 and Table 6-5 show only the Processing Time of the algorithm without any communication with the robots. The average Processing Time for the algorithm in case of 10 or 15 or 20 robots are very close. For 10 robots when n=3, the PT is 0.072sec, while when n=6 the PT is 0.200sec. For 15 robots when n=3, the PT is 0.079sec, while when n=6 the PT is 0.202sec. And for 20 robots when n=3, the PT is 0.090sec, while when n=6, the PT is 0.208sec. We notice that when the number of robots allowed in a coalition (n) doubles, the Processing Time of the algorithm increases by 2.5 times, since the algorithm will calculate a larger number of coalitions. The number of coalitions that will be calculated is $\sum_{r=1}^{n}\binom{n}{r}$. The processing time of the algorithm does not change greatly with a larger number of robots, since there is no communication with the robots.

We can conclude that the Processing Time is affected when there is communication with the robots. This means that the communication overhead has an impact on the Processing Time. Also, we find that the Processing Time of the algorithm is reasonable since the emergency response time is around 10 to 15 minutes in general; the Processing Time that we got is less than 10% of that time.

Table 6-6 shows an example of the best coalition when having 10 robots with n=3. We can see that the best coalition chosen by the algorithm has a time of 100.221msec, costs $130, and has two robots per group. Table 6-7 shows an example of the best coalition when having 20 robots with n=6. We can see that the best coalition chosen by the algorithm has a time of 498msec, costs $130, and has two robots per group. In the first case the B-CLT has the best cost and the best number of robots among all solutions, and in the second case the B-CLT has the best time and the best number of robots among all solutions. Since our algorithm is multi-objective, there is not a single solution that simultaneously optimizes each objective. None of the objective functions can be improved in value without degrading some of the other objective values. We did the same experiment with n=6, which we did not include because of the excessive size of the tables. In this experiment the number of coalitions calculated was 57, and we got the same results.

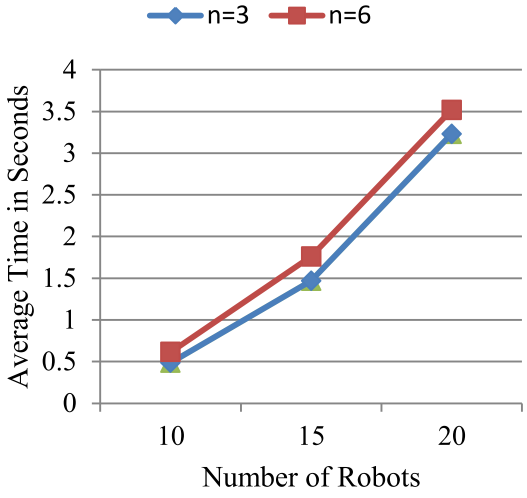


Figure 6-14 The Processing Time of The Algorithm Including The Communication with the Robots

	n=3	n=6
10Rs	0.484	0.613
15Rs	1.468	1.76
20Rs	3.23	3.518

Table 6-4 Average PT Including Communication with The Robots (sec) with n=3 and n=6

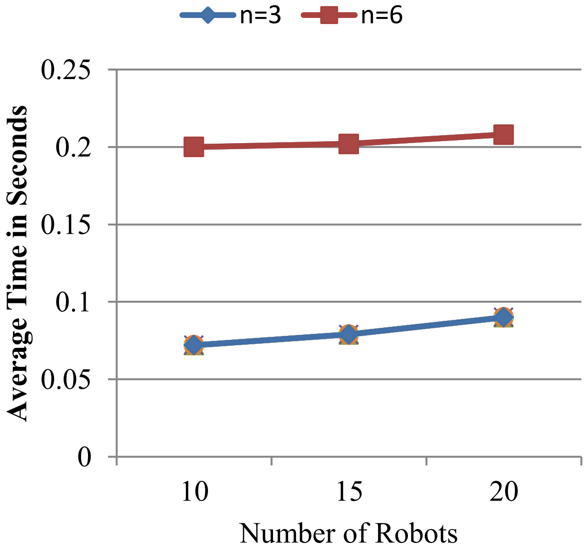


Figure 6-15 The Processing Time of The Algorithm Without Communication with the Robots

	n=3	n=6
10Rs	0.072	0.200
15Rs	0.079	0.202
20Rs	0.090	0.208

Table 6-5 Average PT without Communication with The Robots (sec) with n=3 and n=6

Time (msec)	Cost ($)	Number
100.221	**130**	**2**
98.794	161	2
100.221	137	2
100.221	214	3

Table 6-6 Best Coalition with 10 Robots and n=3

Time (msec)	Cost ($)	Number
498.91	**130**	**2**
600.35	112	2
600.35	120	2
600.35	181	3

Table 6-7 Best Coalition with 20 Robots and n=3

6.5 Chapter Summary

In this chapter we described the overall prototype architecture, including the software tools we used in the implementation. We also described the application that we implemented. We then presented the prototype setup and the performance measurements on the prototype. Finally we presented the simulation environment and the performance measurements on the algorithm. We found that our architecture is scalable in terms of the number of robots and in terms of IaaSs. We also showed that our algorithm found the best coalition among existing coalitions.

Chapter 7

7 Conclusion and Future Work

In this chapter, we will first highlight the contributions of this thesis and then give some hints about future work.

7.1 Contributions Summary

Robotic applications are ubiquities. Unfortunately, provisioning them as cloud computing services in a cost-efficient manner remains a difficult task. In dynamic environments there is no prediction of the size and the location of the event that may occur. Also some tasks that either cannot be solved individually or can be solved more efficiently as a group may need the collaboration between several robots. Forming and dedicating the effective coalition dynamically with the correct number and capabilities of the robots is highly critical. Furthermore, in some cases the capabilities and the number of the robots belonging to one cloud may not be sufficient for a given task, which results in a task not being completed or it being completed in a non-efficient manner.

The requirements we have identified fall into three categories: General requirements, requirements on the robot, and requirements on a Network Level Virtualization algorithm. Moreover, we have reviewed the most relevant related works. We have divided these related works into two categories: Frameworks for robotic applications and algorithms to perform

Network Level Virtualization. Subsequently, we have evaluated these related works based on our requirements; we have observed that none of them meets all of our requirements.

In addition, we have proposed architecture for an infrastructure that enables cost-efficient robotic application provisioning; our architecture fulfilled the requirements that we derived and focused on. Furthermore, the proposed architecture comprised a P2P Overlay and a Robotic Cloud. The Robotic Cloud included an IaaS and a PaaS. The P2P Overlay was used as the interaction network between several IaaSs. The IaaS contained a Network Level Virtualization Layer and a Resources Layer. The Network Level Virtualization Layer includes a Task Delegator Engine which enables task assignment requests to be sent to robots belonging to other IaaSs through the Overlay. Moreover, the proposed architecture contains a Virtualization Engine which selects the most suitable group of robots for a given task by running an algorithm for Network Level Virtualization; we have proposed an algorithm to do this job. Our proposed algorithm contains a filtering and ranking method, and Multi-Objective Particle Swarm Optimization to find the optimal coalition for each task. We have also discussed the REST interfaces of the proposed architecture and the functional procedures.

A proof of concept prototype has been implemented based on the Wildfire Suppression scenario. We implemented the robotic application that sends a request to the IaaS; it is hosted on Google's Infrastructure. We also implemented three IaaSs and the interaction Overlay Network. Lastly, to validate our prototype, a preliminary performance evolution of the overall architecture has been taken. Based on these results, we conclude that our architecture is a valid and promising approach for provisioning robotic applications as cloud computing services in a cost-efficient manner.

Finally, we performed a simulation in order to evaluate our algorithm, where we concluded that our proposed algorithm finds the best coalition among the robots it has, and also that the processing time of the algorithm is reasonable.

7.2 Future Work

The Virtualization Engine executes the requests in the same order that they arrive without giving priority to any request. In other words, if several requests come to the IaaS at the same time they will be blocked until the first request is served. In future work, we can look into prioritizing the incoming requests and having two queues for them. The high priority requests will be put in the first queue and the low priority requests in the second queue. The execution of the requests always starts from the high priority queue.

In our proposed architecture, currently the robots belonging to the IaaS can perform one task or can support one application at a time. In future work, we can look into adding the ability of supporting multiple applications on the same robot; Node Level Virtualization of robots. Also, in our proposed algorithm we are using a very simple filtering method. In future work we aim to add more complicated filtering methods such as dendrogram filtering which is a tree-based algorithm [69].

We proposed an algorithm and evaluated its performance measurements. In future work we can look into comparing our algorithm with other existing algorithms, such as Multi-Objective GA and NSGA-II.

Bibliography

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

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

[3] S. Moon, S.-G. Lee, and K.-H. Park, "Recent progress of robotic vocabulary standardization efforts in ISO," in *Proceedings of SICE Annual Conference 2010*, 2010, pp. 266–268.

[4] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, "Leveraging virtualization to optimize high-availability system configurations," *IBM Syst. J.*, vol. 47, no. 4, pp. 591–604, 2008.

[5] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, New York, NY, USA, 2006, pp. 139–152.

[6] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, "Virtual Sensor Networks - A Resource Efficient Approach for Concurrent Applications," in *Fourth International Conference on Information Technology, 2007. ITNG '07*, 2007, pp. 111–115.

[7] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.

[8] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, 2010.

[9] I. Sriram and A. Khajeh-Hosseini, “Research Agenda in Cloud Technologies,” arXiv e-print 1001.3259, Jan. 2010.

[10] “Google App Engine.” [Online]. Available: https://developers.google.com/appengine/.

[11] “What is Cloud Computing? - Google Developers Academy — Google Developers.” [Online]. Available: https://developers.google.com/appengine/training/intro/whatiscc.

[12] “http://www.vmware.com/pdf/virtualization.pdf.” .

[13] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, “Intel virtualization technology,” *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[14] J.-Y. Hwang, S. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim, “Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones,” in *5th IEEE Consumer Communications and Networking Conference, 2008. CCNC 2008*, 2008, pp. 257–261.

[15] “IBM Systems: Virtualization - eicay.pdf.” .

[16] N. M. M. K. Chowdhury and R. Boutaba, “A survey of network virtualization,” *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2010.

[17] J. Carapinha and J. Jiménez, “Network virtualization: a view from the bottom,” in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, New York, NY, USA, 2009, pp. 73–80.

[18] “http://vmware.com/files/pdf/VMware_paravirtualization.pdf.” .

[19] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, “Network virtualization: a hypervisor for the Internet?,” *IEEE Commun. Mag.*, vol. 50, no. 1, pp. 136–143, 2012.

[20] "'Virtualization in education'. IBM. October 2007.".

[21] H. Kuntze, C. W. Frey, I. Tchouchenkov, B. Staehle, E. Rome, K. Pfeiffer, A. Wenzel, and J. Wollenstein, "SENEKA - sensor network with mobile robots for disaster management," in *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, 2012, pp. 406–410.

[22] A. Chikwanha, S. Motepe, and R. Stopforth, "Survey and requirements for search and rescue ground and air vehicles for mining applications," in *Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference*, 2012, pp. 105–109.

[23] Y. Baudoin, D. Doroftei, G. De Cubber, S. A. Berrabah, C. Pinzon, F. Warlet, J. Gancet, E. Motard, M. Ilzkovitz, L. Nalpantidis, and A. Gasteratos, "VIEW-FINDER : Robotics assistance to fire-fighting services and Crisis Management," in *2009 IEEE International Workshop on Safety, Security Rescue Robotics (SSRR)*, 2009, pp. 1–6.

[24] "Evolution of Robotics: Rescue Robots Are Here!" [Online]. Available: http://robotsfuture.blogspot.ca/2011/04/rescue-robots-are-here.html.

[25] "Robot Based Logistics System for Hospitals - Survey (#30 - IDT," *yumpu.com*. [Online]. Available: http://www.yumpu.com/en/document/view/8697526/robot-based-logistics-system-for-hospitals-survey-30-idt.

[26] J. M. Evans, "HelpMate: an autonomous mobile robot courier for hospitals," in *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems '94. "Advanced Robotic Systems and the Real World", IROS '94*, 1994, vol. 3, pp. 1695–1700 vol.3.

[27] R. A. Beasley, “Medical Robots: Current Systems and Research Directions,” *J. Robot.*, vol. 2012, Aug. 2012.

[28] S. Naskar, S. Das, A. K. Seth, and A. Nath, “Application of Radio Frequency Controlled Intelligent Military Robot in Defense,” in *2011 International Conference on Communication Systems and Network Technologies (CSNT)*, 2011, pp. 396–401.

[29] “DRDO Daksh - Wikipedia, the free encyclopedia.” [Online]. Available: http://en.wikipedia.org/wiki/DRDO_Daksh.

[30] “General Atomics MQ-1 Predator - Wikipedia, the free encyclopedia.” [Online]. Available: http://en.wikipedia.org/wiki/General_Atomics_MQ-1_Predator.

[31] Y. Chen, Z. Du, and M. García-Acosta, “Robot as a Service in Cloud Computing,” in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE)*, 2010, pp. 151–158.

[32] “Microsoft Robotics Developer Studio.” [Online]. Available: http://www.microsoft.com/robotics/.

[33] Z. Du, W. Yang, Y. Chen, X. Sun, X. Wang, and C. Xu, “Design of a Robot Cloud Center,” in *2011 10th International Symposium on Autonomous Decentralized Systems (ISADS)*, 2011, pp. 269–275.

[34] Q. Yu, H. Wei, M. Liu, and T. Wang, “A novel multi-OS architecture for robot application,” in *2011 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2011, pp. 2301–2306.

[35] R. Doriya, P. Chakraborty, and G. C. Nandi, “Robotic Services in Cloud Computing Paradigm,” in *2012 International Symposium on Cloud and Services Computing (ISCOS)*, 2012, pp. 80–83.

[36] R. Doriya, P. Chakraborty, and G. C. Nandi, “#x2018;Robot-Cloud #x2019;: A framework to assist heterogeneous low cost robots,” in *2012 International Conference on Communication, Information Computing Technology (ICCICT)*, 2012, pp. 1–5.

[37] “M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, ‘ROS: An open-source robot operating system,’ in ICRA Workshop on Open Source Software,2009.”.

[38] S. Nakagawa, N. Igarashi, Y. Tsuchiya, M. Narita, and Y. Kato, “An implementation of a distributed service framework for cloud-based robot services,” in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 4148–4153.

[39] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, “Coalition structure generation with worst case guarantees,” *Artif. Intell.*, vol. 111, no. 1–2, pp. 209–238, Jul. 1999.

[40] B. P. Gerkey and M. J. Matarić, “A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems,” *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, Sep. 2004.

[41] D. Di Paola, D. Naso, and B. Turchiano, “A heuristic approach to task assignment and control for robotic networks,” in *2010 IEEE International Symposium on Industrial Electronics (ISIE)*, 2010, pp. 1784–1790.

[42] L. Vig and J. A. Adams, “Multi-robot coalition formation,” *IEEE Trans. Robot.*, vol. 22, no. 4, pp. 637–649, 2006.

[43] H.-Y. Liu and J.-F. Chen, “Multi-Robot Cooperation Coalition Formation Based on Genetic Algorithm,” in *2006 International Conference on Machine Learning and Cybernetics*, 2006, pp. 85–88.

[44] M. Agarwal, L. Vig, and N. Kumar, “Multi-objective Robot Coalition Formation for Non-additive Environments,” in *Intelligent Robotics and Applications*, S. Jeschke, H. Liu, and D. Schilberg, Eds. Springer Berlin Heidelberg, 2011, pp. 346–355.

[45] J. G. Manathara, P. B. Sujit, and R. W. Beard, “Multiple UAV Coalitions for a Search and Prosecute Mission,” *J. Intell. Robot. Syst.*, vol. 62, no. 1, pp. 125–158, Apr. 2011.

[46] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, “Handling multiple objectives with particle swarm optimization,” *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 256–279, 2004.

[47] “Eng Keong Lua; Crowcroft, J.; Pias, M.; Sharma, R.; Lim, S., ‘A survey and comparison of peer-to-peer overlay network schemes,’ Communications Surveys & Tutorials, IEEE , vol.7, no.2, pp.72,93, Second Quarter 2005.”.

[48] F. Belqasmi, R. Glitho, and C. Fu, “RESTful web services for service provisioning in next-generation networks: a survey,” *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 66–73, 2011.

[49] “Heuristic (computer science),” *Wikipedia, the free encyclopedia*. 02-Dec-2013.

[50] “Natalia Kokush, ‘An Introduction to Heuristic Algorithms.’”.

[51] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS ’95*, 1995, pp. 39–43.

[52] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, “Inertia Weight strategies in Particle Swarm Optimization,” in *2011 Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*, 2011, pp. 633–640.

[53] “konstantinos E. Parsopoulos and Micheal N. Vrahatis, Multi-objecitve particles swarm optimization approaches 2008.” .

[54] “Jean-Pierre Brans, ‘PROMETHEE Methods.’”.

[55] “Vojislav Tomic, Zoran Marinkovic, Dragoslav Janosevic, ‘PROMETHEE Method Implementation With Multi-Criteria Decisions.’”.

[56] R. Hassan, B. Cohanim, O. de Weck, and G. Venter, “A Comparison of Particle Swarm Optimization and the Genetic Algorithm,” in *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, American Institute of Aeronautics and Astronautics.

[57] Y. Duan, R. G. Harley, and T. G. Habetler, “Comparison of Particle Swarm Optimization and Genetic Algorithm in the design of permanent magnet motors,” in *Power Electronics and Motion Control Conference, 2009. IPEMC ’09. IEEE 6th International*, 2009, pp. 822–825.

[58] J. E. Onwunalu and L. J. Durlofsky, “Application of a particle swarm optimization algorithm for determining optimum well location and type,” *Comput. Geosci.*, vol. 14, no. 1, pp. 183–198, Jan. 2010.

[59] B. Horling and V. Lesser, “A Survey of Multi-agent Organizational Paradigms,” *Knowl Eng Rev*, vol. 19, no. 4, pp. 281–316, Dec. 2004.

[60] J. Figueira, S. Greco, and M. Ehrgott, *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer, 2005.

[61] Y. O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady, "Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, 2010, pp. 91–98.

[62] "LEGO.com Mindstorms." [Online]. Available: http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com.

[63] "Lego Mindstorms NXT - Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT.

[64] L. Gong, "JXTA: a network programming environment," *IEEE Internet Comput.*, vol. 5, no. 3, pp. 88–95, 2001.

[65] "Webots: robot simulator." [Online]. Available: http://www.cyberbotics.com/.

[66] "Eclipse - The Eclipse Foundation open source community website." [Online]. Available: http://www.eclipse.org/.

[67] "Restlet Framework - RESTful web API framework for Java." [Online]. Available: http://restlet.org/.

[68] "Restlet 2.1 - Tutorial." [Online]. Available: http://restlet.org/learn/tutorial/2.1/.

[69] F. Murtagh, "Counting dendrograms: A survey," *Discrete Appl. Math.*, vol. 7, no. 2, pp. 191–199, Feb. 1984.