AUTOMATED MANAGED CLOUD-PLATFORMS BASED ON ENERGY POLICIES

by

MARWAH ALANSARI

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
May 2016

Abstract

Delivering environmentally friendly services has become an important issue in Cloud Computing due to awareness provided by governments and environmental conservation organisations about the impact of electricity usage on carbon footprints. Cloud providers and cloud consumers (organisations/enterprises) have their own defined *green policies* to manage energy consumption at their data centres. At service management and execution level, *green policies* can be mapped as *energy management policies* or *management policies*. These *management policies* are implemented using various strategic plans. Focusing on the cloud consumers side, *management policies* are initialised by business managers as a set of 'if/then' statements. *Management policies* can change regularly based on the nature of the technical environment, changes in regulation, and business requirements. The usage of low-level programming methods for executing *management policies* into a cloud-platform automatically could be time-consuming and costly. Thus, we found that there is a gap between the level of describing and implementing such policies in the cloud platform. This thesis provides a method to bridge that gap by covering three main dimensions:

- To automatically execute *management policies* into a cloud-platform, we propose a runtime policy-based architectural framework called *MP-Framework*. *MP-Framework* is a generic architecture that would be useful for enforcing two different sets of low-

level and high-level *energy management policies* in the cloud platform.

- To simplify the expression of *management policies* at both the policy description and the implementation levels, we propose a specification for formulating various types of *management policies* that can be used by either rule languages or rule-modelling languages. The proposed specification is based on the existing UML-Rule Modelling Language (URML) meta-model which is utilised for designing a domain specific language called CloudMPL.

- To identify the suitable energy cost saving policy from a set of suggested *management policies* before a real implementation, we provide an off-line method based on the modelling and the analysis of the executable *management policies* and cloud platform using Coloured Petri-Nets (CPN). We suggest two methods for calculating the cost of energy consumption and the cost of migrating virtual machines from the produced energy management cloud architectural models.

To evaluate each method covered in this thesis, we used an Energy Management Case Study for a private cloud scenario. The case study is implemented in a real cloud testbed for demonstrating the applicability of the proposed MP-Framework and the suggested specification. In addition, the case study is stimulated in a Coloured Petri Nets tool called CPN Tool for evaluating the proposed modelling and analysis method.

Dedication

To my grandfather *Dr. Nassir Abdulellah Alansari* who inspired me through his knowledge and education during his career in academe,

“ Since I was a child, I dreamt to be like you one day ”

To my loving parents, for their love, their motivation, their patience and their support

To my brothers, for being always with me and for all positive power they gave

To my family and my friends, for being supportive

To my supervisor *Dr. Behzad Bordbar*, for all the support he gave during my research time

Acknowledgements

O Allah, I am very grateful for assisting me, directing me to the knowledge, and the strength to continue. There have been long years in studies and research abroad away from my family. They were years full of determination to achieve my goals. O Allah, I am very grateful to you for lighting my path to reach these goals. My mother, my father and my brothers were the second supportive power during my studies. Thank you for being always with me, coping with me during my difficult moments and sharing my happiness in my achievements. My supervisor *Dr.Behzad Bordbar* is one of my influences who believed in my ideas and my determination. I am very grateful to my supervisor for his advice, his passion in research, and his encouragement in learning. I am very thankful to *Dr.Behzad Bordbar* for every discussions that we made to solve various research challenges and also for dedicating his time to support us as PhD students. I am very thankful to my friends specially (*Wafa*, *Afnan*, *Abeer*, *Bayan*, *Nada*, *Randa* and *Fatema*) for their support and being with me during happiness and difficult times. Furthermore, I am very thankful to my colleagues at the school of computer science as well as research collaborators. And last but not least, again I am very thankful to Allah for assisting me to complete this thesis with happiness, courage and confidence.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Delivering environmentally friendly services has become an important issue in Cloud Computing. Awareness of the significance of developing energy-aware services has increased due to the encouragement of governments and environmental conservation organisations focussed on the impact of electricity usage on carbon footprints. Over the past ten years, there has been a movement towards measuring the amount of electricity consumed by data centres and IT services [101, 129]. According to figures published in a report for US Congress in 2006 on the energy efficiency of servers and data centres in the United States [31], data centres consumed approximately 61 billion kilowatt-hours (kwh) at this time. These figures have risen significantly since this report. Compared to figures published in [31], recent statistical figures which were published in 2014 by the NRDC and Anthesis [129], show that nearly 95% of existing data centre segments in the United States (including small, medium, corporate and multi-tenant data centres) are on average not energy efficient [129].

Here, we draw attention to the figures obtained by Hyper-Scale Cloud Computing data centres [129]. Cloud computing data centres consumed nearly 3.3 billion kwh/y from a total of 76.4 billion kwh/y of energy consumed by data centres in the United States [129]. As a percentage, cloud data centres consumed approximately 4% of the total amount of electricity used in this country [129]. Based on the figures in [129], it is estimated that the amount of energy consumption will decrease by 38% by 2020 if data centres become *energy-efficient* on a global scale and apply techniques to reduce energy consumption [129]. In European data centres, it is projected that if energy-efficiency strategies and action plans are applied, the total electricity usage in data centres will be reduced by 20% by 2020 [2]. The previously mentioned figures reveal that the high usage of electricity is one of the sources of increasing levels CO2 and greenhouse gas emissions [2]. Therefore, the need to solve the challenges involved in the design, development and deployment of energy-aware management for cloud resources and services has received considerable attention from the Cloud Computing community.

In Cloud Computing, cloud providers and cloud consumers fulfil two interacting roles. Cloud providers tend to have massive data centres, which might be geographically distributed. Employing recommendations for designing energy-efficient data centres, as suggested in [2, 129], cloud providers can generate and implement their own green policy at various levels in data centres [101]. At the services execution level, *green policies* can be mapped as *energy management policies* implemented using various dynamic algorithms. One example of the implementation of *green policies* targets the control of energy consumption via the design of energy-aware resource allocations, as in [35, 59, 101]. Another example of the implementation of *energy management policies* aims to manage the virtualisation layer by developing middleware. This controls the migration and placement operations of virtual machines, as described in [24, 26, 66, 95, 133]. *Green policies* can be implemented by focussing on the facilities provided in the data centres and the installation of energy-efficient equipment, as suggested in [79, 101, 129]. For example, the appropriate distribution of cooling and heating within a data centre building can reduce the amount of consumed energy, as recommended in [101].

It is necessary to manage the amount of energy consumption at cloud consumer data centres to increase the credibility of their response to environmental issues. Therefore, the cloud consumers (organisations/enterprises) can contribute to the reduction of energy usage for their cloud platform by assigning *energy management policies*. The importance of using such policies for cloud consumers, as the recommendations found in [2, 129] specify, is that it will decrease the amount of CO2 emissions produced by IT-infrastructure. An example of a suggested recommendation is employing physical servers with a virtualisation layer, which can help to decrease the number of servers used in a data centre [129]. Another example of an energy-efficient plan in the data centre is outsourcing computing services using public cloud providers [101]. Therefore, cloud consumers have their own *energy management policies* that are generated by business managers. However, the cloud consumers might face the problem of implementing the *energy management policies* into a cloud-platform automatically. This is because of the existing gap between the levels of description and implementation of *management policies*. In this research context, we refer to *energy management policies* as *management policies*.

To sum up, both cloud providers and cloud consumers have their own defined *management policies* that are implemented via their own strategic plans. By focussing on the cloud consumer side, this thesis addresses the following problems: (a) defining a specification for formulating *management policies* into an executable form for an infrastructure-as-a-service (IaaS) cloud model; (b) the automatic execution of the described *management policies* in the cloud-platform; and (c) identification of the potential *energy management policy* that would be executed in the cloud-platform and would save energy-cost.

This chapter is organised as follows: Section 1.1 explains the paradigm of Cloud Computing and highlights some of the existing research on automatic management. Section

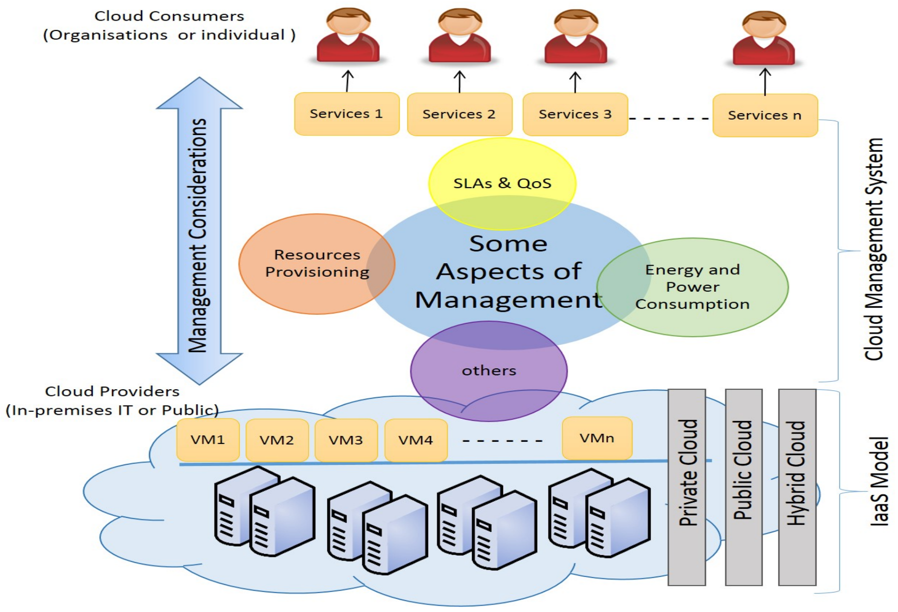


Figure 1.1 – A cloud architecture for Infrastructure-as-a-Service (IaaS) deployment model and some management aspects considered by cloud providers and cloud consumers

1.2 discusses the research problem in detail. In Section 1.3, briefly we present our contributions. Section 1.4 lists the documents that we published during the progress of this research. Finally, the structure of the thesis is presented in Section 1.5.

1.1 Cloud Computing and Its Management

Cloud Computing is defined by Foster *et al.* as *"a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualised, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet"* [54]. From this definition, it is evident that services are delivered using highly scalable and reliable mechanisms, regardless of their geographical locations [34]. In the cloud business model concept, services are provided through cloud providers. The cloud providers initiate service provision to cloud consumers after assigning service-level agreements (SLAs). Establishing such a business concept distinguishes Cloud Computing technology from other distributed systems, such as Grid Computing and Utility Computing [54].

According to Foster *et al.*, a typical cloud architecture model is organised into layers, namely *fabric, unified resources, platform and application* layers [54]. The *fabric layer* includes hardware components of the cloud data centres, such as servers, network and storage. The *unified resource* implements the virtualisation technique. Virtualisation allows multiple instances of either similar or different Operating Systems to be run on a single host, which assists in increasing server utilisation level [134] (see Figure 1.1 for the actual representation of both the fabric and unified resource layers). The *platform layer* consists of middleware that manages and handles the delivering of cloud services; in some research, this is referred to as a *cloud management system.* Finally, the *application layer* is the cloud service that is delivered to cloud consumers [54].

There are three basic deployment models for the cloud, namely the *private cloud*, *public cloud*, and *hybrid cloud* [101]. A cloud-platform is known as a *private cloud* both the physical resources and the virtualisation layer are within the scope of the organisation [101]. Meanwhile, a *public cloud* involves vendors' provision of their service publicly using either short-term or long-term contracts [101], such as providers of Amazon services [21]. A *hybrid cloud* deployment model combines both the *private cloud*, and *public cloud* [101]. Those cloud deployment models can be managed via a cloud management system.

A cloud manager or cloud management system is responsible for performing different automatic management functionalities. Practically, HP Helion Eucalyptus [67], and OpenNebula [105] are examples of existing cloud management systems used to construct various dynamic management frameworks in a cloud infrastructure. Automatic management is essential for ensuring the on-demand availability of resources and deployed services. As shown in Figure 1.1, a number of management factors have been considered in various studies of automatic management in Cloud Computing. One of these is the problem of resource provisioning, which is addressed by developing algorithms described in [97, 111, 133] to allocate, place and deploy virtual machines to physical hosts and services to cloud consumers. Another type of management research is shown in [22, 93, 103]; these studies tackle the issue of ensuring the availability of cloud services in cloud infrastructure after provisioning resources. Such research proposed mechanisms to monitor and evaluate quality-of-Service (QoS) for the provided services [22, 93, 103].

Any delivered cloud service results from a process of assigning Service Level Agreements (SLAs), which are contracts established between cloud providers and cloud consumers . Therefore, the problems of managing SLAs and monitoring their metrics must be addressed via dynamic and automatic management in the cloud community. There are some existing frameworks for automating the negotiation and establishment of SLAs, as in [39]; moreover, frameworks for dynamically managing and monitoring violations in SLAs have been suggested in [50, 91]. Furthermore, research has covered other elements, such as the consideration of security [27, 44] and management failure [62].

This thesis addresses some of the challenges of managing energy consumption. Such management is based upon applying plans that would save energy usage and energy cost. Examples of such plans are live migration of virtual machines or services, reconfiguration of resources provided for running services and switching-off of idle hosts or those that have low utilisation levels for a long period [92]. Some of the existing architectures of a cloud manager for energy management share a similar theme. Such architecture is organised in two levels of controllers, namely a *global controller* and a set of *local controllers*. The *global controller* makes decision affecting the overall amount of energy consumption for the environment. In contrast, *local controllers* apply management actions on a small scale targeting the hosting node level (a detailed investigation of the state of the art of automatic architectures, particularly energy-efficient ones, is presented in Section 2.7 in Chapter 2).

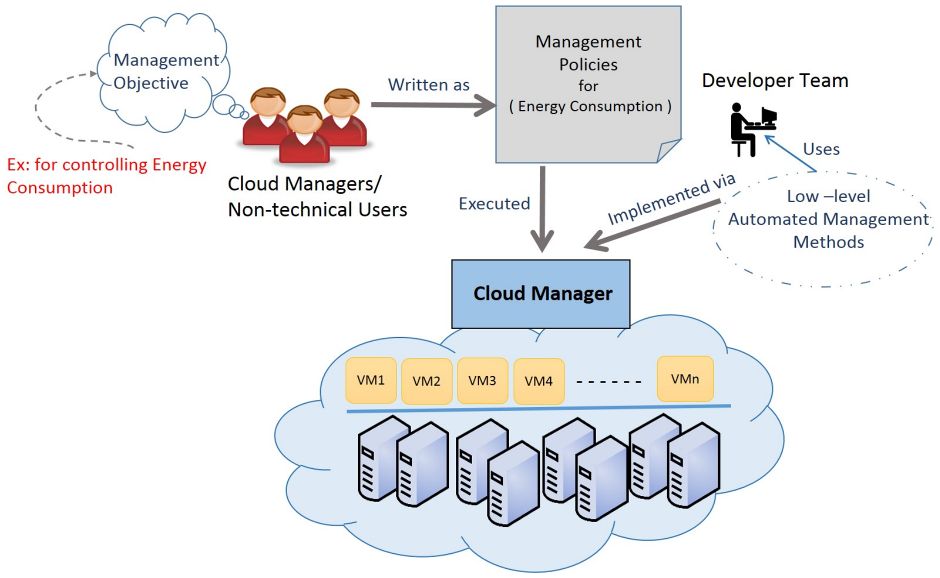

Management Objective
Ex: for controlling Energy Consumption
Cloud Managers/ Non-technical Users
Written as
Management Policies for (Energy Consumption)
Developer Team
Uses
Low –level Automated Management Methods
Implemented via
Executed
Cloud Manager
VM1
VM2
VM3
VM4
VMn

Figure 1.2 – The gap of executing management policies in a cloud platform

1.2 The Research Problem

Before elaborating on the research problem, let us consider a cloud platform, as shown in Figure 1.2; such a platform is managed by a Cloud Manager, such as OpenNebula, which is a cloud management system [105, 120]. As presented in Figure 1.2, cloud business managers or non-technical users have a management objective. In our research, we consider the amount of energy consumption governed by the cloud infrastructure and the control of SLA violations to a minimal level as a management objective. This objective is applied to the cloud platform as a long-term goal.

To practically apply the specified management objective in a cloud platform, such as that shown in Figure 1.2, the management objective is expressed in the form of management policies during the design stage. Such management policies must be executed automatically in the cloud platform. One of the possible solutions is a manual development process. Here, the development team manually converts the expressed management policies to executable ones. This is achieved by implementing the policies using *low-level programming methods*. Such methods are applied to one of the suggested automatic management architectures used in previous studies, such as those proposed in [35, 91, 92]. The development team uses the policy description and low-level application programming interfaces (APIs) provided, which deal with the cloud platform. Then, they implement the policy via designing a system that uses the monitoring-decision making-acting cycle [26]. An object-oriented strategy pattern [55] can be used to encode the management policies. Next, a controller software component can be used to select a specified management strategy and its execution time according to the description defined in the management policy. The controller can interact with a number of software components to monitor the cloud platform parameters and execute actions.

At the current stage in cloud management systems, particularly OpenNebula [105], some management operations are accomplished manually, which require human intervention. For instance, a Cloud Operator triggers live migration action to move virtual machines among physical nodes in OpenNebula [105, 120] with objective to manage the running infrastructure. The manual approaches for triggering management actions are not beneficial for the current architectural design of a cloud management system.

Manual methods do not scale to handling enormous running services in a cloud platform or the complexity of the described management policies. We would like to point out that *management policies* are expressed according to a defined set of business, financial or environmental requirements. Based on previous research on automatic management [26, 35, 92], the requirements for identification of a management policy are directly related only to low-level parameters. Such parameters can be measured in a cloud infrastructure, such as through resource consumption for the running service, or SLA metrics such as the violation level and energy consumption [26, 35, 92]. Therefore, it is possible to construct a management policy to automatically manage energy consumption in the cloud infrastructure based on low-level parameters using any proposed automatic management architecture, as explained in Section 2.7.

There is another set of unmeasured elements or logical constraints required in the cloud platform that might be found in management policies. Such parameters or constraints have received less attention in the existing research on automatic management. In our research context, we refer to these constraints as a high-level policy. High-level parameters are those related to an organisational perspective [79]. Examples of high-level parameters are the time and the location of a cloud service. Such metrics follow the regulations and environmental constraints defined by cloud business managers (or non-technical users). To elaborate, cloud business managers can define a high-level policy that governs the execution of cloud services during peak and off-peak times. In addition, the managers can define another type of high-level policy that would control the movement of their cloud services across multiple geographically distributed data centres based on an energy cost-saving location. Thus, for a management policy related to combinational parameters (low-level and high-level), the existing automatic energy management architectures must be modified to cope with the multi-level parameters that might be found in the management policy.

There is another dimension related to the characteristics of the defined management policies and the cloud platform. Management policies can change regularly. This implies that any set of defined policies can be modified or extended. The continuous changing of such policies is based on the nature of the technical environment and changes in regulation and business requirements. Furthermore, the alteration of the management policy is also based on the nature of the cloud environment, which is considered dynamic and of massive size [81]. Therefore, the cloud platform might include different types of events that require the management policy to be extended to form a new set of policies. Thus, if the automatic system for enforcing management policies is designed using low-level programming methods or patterns, as explained above, the time and cost required for software development and maintenance may increase. As a result, employing a cloud management system with a method that can automatically execute management policies in a less complex manner is essential.

To summarise, we found that there is a gap between the level of describing policies and the conversion of such policies to be implemented in the cloud platform. The emergence of this situation is a consequence of the following issues:

1. There is no generic architectural framework for automatically enforcing specified energy management policies on a cloud platform;
2. A methodology is lacking to specify how to use lower-level APIs provided by a cloud management system, such as OpenNebula APIs [105, 120] and integrate them to build an energy-saving cloud platform;
3. There is no specification to design a declarative language where energy-management policies can be written to reduce the development and modification process of such policies;
4. There is no methodology for evaluating the potential executable energy management policies before real implementation on a cloud platform.

1.3 Research Contributions

The novel contribution of this research is enabling the dynamic energy efficient management concept on a cloud platform via the analogy of *management policies* and the automatic execution of such defined policies. By automatic execution, we mean that a management action is triggered automatically as a result of analysis of a predefined set of monitored parameters suggested in the policies. Therefore, we summarise our contributions as follows:

1. To bridge the gap between the level of describing the management policies and the level of implementing them, we propose a policy-based architectural framework for automatically triggering a set of management actions into a cloud platform with

the purpose of reducing the amount of energy consumption and energy cost. The framework targets the *IaaS* cloud model, where a cloud provider advertises computing or storage as a service. The purpose of using this framework is to automate the management process in a cloud infrastructure consisting of a set of running virtual machines and physical nodes. The policy-based framework is designed to be plugged into the interfaces of any cloud management system;

2. To simplify the expression of *management policies* at both the policy description and the implementation levels, we propose a specification for formulating various types of *management policies* that can be used by either rule languages or rule-modelling languages. In addition, we design a conceptual mapping to transform management policies described in a high-level language into executable management policies in rule language; and
3. Since our research is a part of providing automatic management on a cloud platform while considering energy efficiency aspects, it is necessary to have a methodology to assess the suggested *management policies* before a real execution takes place in the cloud environment in terms of saving energy costs. Our methodology for evaluating policies involves two elements. The first is modelling both *management policies* and a *cloud-platform* using Coloured Petri-nets (CPNs) [71], whilst the second is analysing costs associated with the modelled policies using two methods to calculate costs. In this research, the proposed off-line modelling and analysis method can allow the cloud consumer to gain knowledge about estimating the energy costs of potential policies.

From the summarised contributions, this thesis aims to solve an energy management issue described in the form of policies. Cloud developers will have detailed guidelines employed with specifications for designing an architecture that can be used to build automatic management that governs energy consumption on a cloud platform. Furthermore, cloud administrators or developers can use the suggested off-line modelling and analytical methods in this research to evaluate the energy cost of running various set of management policies before implementation.

1.4 Publications

Four conference papers have been published during the development of this thesis, as listed below.

1. Alansari, M. and Bordbar, B. (2013). ‘An architectural framework for enforcing energy management policies in cloud’. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 717-724.
2. Alansari, M. and Bordbar, B. (2014). ‘Modelling and analysis of migration policies for autonomic management of energy consumption in cloud via petri-nets’. In *2014 International Conference on Cloud and Autonomic Computing*, pages 121-130.
3. Alansari M., Almeida, A., Bencomo, N. and Bordbar, B. (2015). ‘CloudMPL: a domain specific language for describing management policies for an autonomic cloud infrastructure’s’. In *Proceedings of the 5th International Conference on Cloud Computing and Services*, pages 451-462.
4. Alansari M. and Bordbar, B. (2016). ‘An off-line analytical approach to identify suitable management policies for autonomic cloud architecture’. In *Proceedings of the 6th International Conference on Cloud Computing and Services*, pages 232-239.

1.5 Thesis Outline

The remainder of the thesis is structured as follows: Chapter 2 gives an overview of the technologies and theoretical elements required to understand each contribution made in this thesis. Furthermore, it reviews related work to each contribution. Chapter 3 explains the architectural *Management Policy Framework (MP-Framework)*, the theoretical aspects of developing the framework and its implementation in a real cloud platform in detail. The specifications for the management policies used by both rule language and domain-specific language (DSL) are explained in Chapter 4. Chapter 5 outlines the theoretical concept of modelling an *Automated Managed Cloud Platform* and *management policies* via Coloured Petri-nets (CPN). Chapter 6 explains the proposed Simulation-based Cost Calculation Method (SCCM) from the generated CPN models for a cloud platform. An extension of the SCCM method is discussed in Chapter 7. Finally, Chapter 8 comprises the conclusion of the thesis and a discussion of research limitations and future research.

CHAPTER 2

Research Background and Related Work

Various technologies and theoretical approaches have been analysed to solve the challenges addressed in this thesis. Designing an automated cloud platform for management energy consumption is the first aspect covered in this thesis. Therefore, this chapter provides an overview of the selected technologies related to cloud management systems and rule-based systems. Furthermore, we provide a review of the existing automatic management architectures used for large-scale systems. The review of such architectures is required to explain the Management Policy Framework presented in Chapter 3.

The second contribution of this thesis is proposing specifications for describing management policies using both a domain language and an executable rule language, which are described in Chapter 4. Therefore, this chapter includes an overview of the existing specifications of modelling rules. In addition, the chapter investigates the existing domain-specific languages (DSLs) designed for cloud platforms.

The third contribution of the thesis is concerned with the theoretical aspect of modelling an automated manageable cloud platform and management policies for energy consumption. Therefore, this chapter offers detailed background about Petri-net and coloured Petri-net modelling. Furthermore, this chapter includes an investigation into the current existing methods for cost analysis from Petri-net models. This background is required to clarify the modelling aspect and analysis method covered in Chapter 5, Chapter 6 and Chapter 7.

2.1 Cloud Management Systems

Cloud data centres naturally tend to deliver a vast amount of services to a large number of users. Thus, manual management in such an environment is difficult and does not scale to handle a larger number of services. As a result, a number of cloud management systems, such as HP Helion Eucalyptus [67], OpenNebula [105, 120], and oVirt [106], have been developed to include automatic functionalities such as service deployment, configuration, management, scheduling, and service termination. Such cloud management systems can govern heterogeneous cloud environments by offering several drivers suited to operate with various hypervisors. For example, OpenNebula [105] has multiple drivers connected to three different hypervisors which are Xen [106], KVM [84], and Vmware [14]. Our research deals with management using OpenNebula; therefore, the remainder of this section provides a detailed overview of OpenNebula and its architecture.

2.1.1 An Overview of the OpenNebula Cloud Management Toolkit

OpenNebula [105, 120] is a flexible toolkit that allows users to establish and to manage cloud services and virtualised environments [105]. The OpenNebula platform is enriched with a number of features that assist both cloud consumers and cloud operators. OpenNebula provides cloud consumers with flexible methods to access the virtualised platform, and to control and monitor their services using a web-based front-end module [105]. Cloud operators can deploy, configure, and manage both the cloud infrastructure and the virtualised environment using either a simple Unix-command line interface or a web-based front-end [105]. Furthermore, the OpenNebula solution is empowered with extensible Ap-

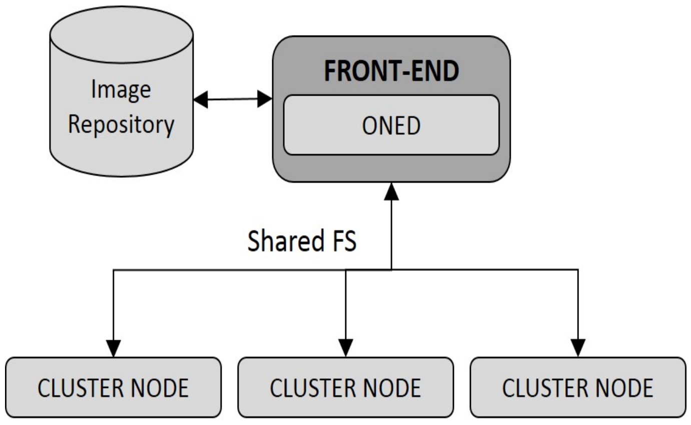


Figure 2.1 – A sample of a deployment architecture of OpenNebula to build a Private Cloud as shown in [120, p.26]

plication Programme Interfaces (APIs) that allow cloud operators to build customisable features suited to their configured platform [105].

The OpenNebula solution is scalable and supports multiple deployment models. OpenNebula can be utilised to control a large-scale platform consisting of up to 500 virtualised nodes. In addition, OpenNebula can be used to establish three different cloud deployment models: private, public, and hybrid models [120]. This can be achieved by configuring a suitable local driver for controlling the virtualised environment and also configuring the external driver for launching and administrating pay-as-you-go services such as Amazon EC2 [21]. An example of a deployment architecture that utilises OpenNebula is the establishment of a private cloud model, which is the implementation of a testbed used in the Energy Management Case Study explained in Chapter 3.

Figure 2.1 depicts a typical usage of OpenNebula. OpenNebula consists of a global manager component, which is called the ONED management daemon and is deployed on the front-end node. The ONED daemon controls a set of worker physical nodes, which are known as cluster nodes. Each cluster node runs a hypervisor that deploys virtual machines images. The communication drivers in the ONED daemon implement a Secure Shell (SSH) network protocol for securing the data transmission amongst cluster nodes. In addition, the ONED daemon uses an image repository to make the virtual machine images accessible for deployment and re-configuration. The architecture of OpenNebula can be scaled by organising the deployment architecture into a hierarchical structure [120].

The Architecture of the Core of OpenNebula

The flexibility provided by OpenNebula resulted from creating the ONED daemon which offers fixable features and a well-defined structure [120]. As shown in Figure 2.2 any configured ONED node is organised into three layers, each of which has a set of components. The layers are the drivers, the core, and the tools [120]. The drivers are designed to access a remote host for transferring, or deploying and running, or monitoring virtual machines. OpenNebula offers three types of drivers: *Transfer Drivers*, *Virtual Machine Drivers* and *Information Drivers*. The *Transfer Drivers* control the disk images on the current storage system (shared or non-shared file system). The *Virtual Machine Drivers* are hypervisor-specific components that are used for managing the deployed virtual machine instances on hosts. The final set of drivers are the *Information Drivers*, which collect the current status of virtual machine instances and hosts [120].

The core layer contains the brain of OpenNebula, which is written in highly optimised C++ code. The core layer consists of three separate managers for controlling data centre modules, which are *virtual machine managers*, *physical host managers* and *virtualised network managers*. Those managers communicate with remote drivers via the request manager which implements XML-RPC as inter-process communication protocol. This effective design for both hosts and virtual machine managers allows OpenNebula to govern up to 500 server hosts and up to 16,000 virtual machine instances [120]. OpenNebula Host Managers can be used to build hierarchical large-scale data centres. In addition, the core contains shared storage which can be used to locate the configuration and virtual machines' description files. Furthermore, the shared storage can be used to store the collected monitoring information, which can be either a simple SQLite database or a replicated MySQL database. In order to support interoperability, the core has a set of customisable APIs which can support Java and Ruby. In addition, the core is enhanced with a hook system which is implemented to give users the ability to execute customised scripts and configure a predefined set of events [120]. The tools layer includes a scheduler module and command line interfaces. The scheduler module can be configured to run allocation algorithms for the deployment of virtual machine images to available physical hosting nodes in the cloud-platform. The existence of such components simplifies the usability and the accessibility of OpenNebula for both cloud administrators and end-users [105].

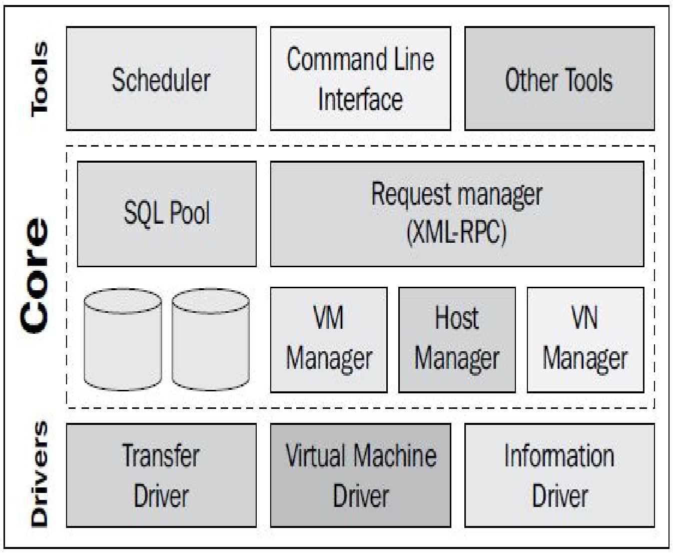


Figure 2.2 – The components of OpenNebula main core (ONED) as presented in [105, p.7]

2.2 Drools Rule Engine and Drools Rule Language

Drools [11] is a production rule system, which is a type of rule-based engine. The Drools Inference Engine is implemented using the enhanced version of the Rete algorithm for supporting Object Oriented Patterns [70]. Charles Forgy [53] introduced the Rete algo-

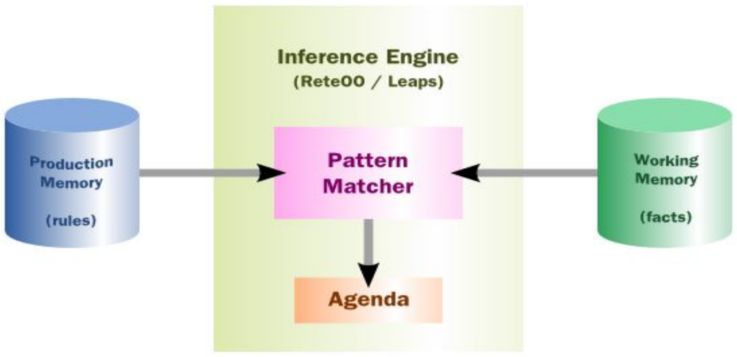


Figure 2.3 – The architecture of the OO-Rete engine in Drools, extracted from Chapter 1 in [70]

rithm, which is an efficient pattern-matching algorithm designed for comparing a large set of patterns to a large set of objects [53]. The Rete algorithm was specially developed to be used in the interpreters of production systems [53]. The algorithm represents rules as an acyclic graph to form a Rete network and provides a pattern-matching process [53, 70].

The Architecture of Rule Based System in Drools

Figure 2.3 presents a detailed description of the functionality of the rule-based system implemented in Drools. As illustrated in Figure 2.3, the Drools Rule Engine requires the inclusion of two types of memory, namely production memory and working memory. Production memory is a type of long-term memory in which rules are stored. Production memory is fixed during the run-time of the rule engine. Working memory is a type of short-term memory which contains facts that need to be evaluated by the inference engine. Facts are object models or the instances that contain attributes that illustrate a domain data for an application. During the run-time, the values of facts may change. As a result, the Rete algorithm has a strategy for triggering the changes in the values of the facts and performs a fast comparison process using the Rete network. The objective is to find the correct rules that can be fired by the agenda [53, 70]. The agenda is the place where a rule that has become an active rule is stored to be executed. The agenda uses a conflict-

```
rule "increase balance for credits"

when
  ap : AccountPeriod()
  acc : Account( $accountNo : accountNo )
  CashFlow( type == CREDIT, accountNo ==
$accountNo,
            date >= ap.start && <= ap.end,
            $amount: amount)
then
     acc.balance  += $amount;
end
```

Figure 2.4 – A sample of a rule for increasing the balance amount for a bank customer expressed in Drools Rule Language extracted from Chapter 2 in [70]

resolution methodology for ordering the execution of active rules [53]. More information on the basics of the Rete algorithm can be found in [48].

The Drools Tool [11] has an editor for authoring rules. The editor allows the rule developer to write a set of rules in a language called Drools Rule Language (DRL) and stores them into DRL format. DRL is a rule language developed by JBoss based on the Java programming language. The Drools Rules Language allows rules to be specified as a rule-set consisting of a number of conditions followed by a set of sequential actions in which the rule-set is expressed in the following format:

when (condition statements) **then** (action statements)

To present the basic block of a rule written in Drools Rule Language, Figure 2.4 illustrates an example of a rule which increases the balance amount for a credit card. Any rule written in Drools should be enclosed between the keyword $<$ *rule* $>$ and $<$ *end* $>$. The keyword $<$ *when* $>$ is an indication for the condition part of a rule whereas the keyword $<$ *then* $>$ is an indication for defining the action part of the rule. Drools Rule Language requires a data-domain model for writing conditions. In the illustrated example in Figure 2.4, the data-domain model consists of AccountPeriod, Account and CashFlow classes.

In Figure 2.4, the condition part consists of three compositional conditional statements. The statement *"ap : AccountPeriod()"* means assigning to a new object *ap* the extracted value of an object of type *"AccountPeriod()"*. Whilst, the statement *"acc : Account($accountNo : accountNo)"* for an object *"Account"* means to get the value of an attribute *"accountNo"* and assign it to a new defined object called *"$accountNo"*. The account is stored in a defined object called *"acc"*. The previously mentioned statements perform the job of selection statement as found in a database. The final statement in Figure 2.4 is a compositional statement which selects the amount *"$amount"* from *"CashFlow"* which is assigned to a valid account number [70]. On the other hand, in Figure 2.4, the action part is an update of the balance in the "acc" object with the value found in the selected *"$amount"* object. Therefore, the outcome of this rule is to increase the balance for all valid accounts [70].

In DRL, the action part can include a single statement or a number of statements. Furthermore, the action part in Drools Rule Language allows the execution of a set of statements written in Java code [20].

2.3 An Overview of Petri-nets and Timed Petri-nets

Throughout this thesis, we examined the problem of evaluating an automated cloud platform executing a number of *Management Policies* before executing them in a real platform. Chapter 5, Chapter 6 and Chapter 7 cover this problem. In this section and the remaining sections, we provide an overview about Petri-nets, in particular coloured Petri-nets.

Petri-nets (PNs) are graphical and mathematical tools that can be used for modelling and analysis of a wide range of systems [100]. Murata [100] noted that Petri-nets offer a promising method for describing and studying systems that are characterised by certain features, namely systems that are concurrent, asynchronous, distributed, parallel, non-deterministic, and stochastic [100]. In general, a Petri-nets model consists of a finite set of *Places and Transitions* which are connected by a finite set of *arcs* [47]. Therefore, in Petri-net tools, a system model is visually displayed to simplify its representation. Furthermore, in Petri-net models, tokens are applied for imitating the dynamic and concurrent behaviours of modelled systems [100]. Since Petri-nets are basically mathematical tools, the movement of tokens in Petri-net models can be controlled and governed using algebraic equations [100].

Petri-nets were introduced by A. C. Petri in 1962 for synchronising communicating automata. Afterwards, Petri-nets were extended to various versions which are employed with more definitions and capabilities to describe different and complex models [47]. The varieties in Petri-net families allow a system designer to study and validate qualitative and quantitative properties for the modelled system [47]. Some examples of Petri-net versions are place-transition Petri-nets, Timed Petri-nets, Priced Timed Petri-nets, Stochastic Petri-nets, and Coloured Petri-nets [47]. Since this thesis focuses on Coloured Petri-nets and Timed Petri-nets, we provide a definition and formulation of those types in the sections below.

2.3.1 The Description of Petri-nets

A Petri-net is a special type of bipartite directed graph consisting of three main elements, which are places, transitions, and directed arcs. The directed arcs connect places to transitions and transitions to places. Visually, places are represented by circles and transitions as bars [138]. Figure 2.5 provides an example of a Petri-net. In Figure 2.5, the Petri-net model consists of five places and four transitions. Two transitions are labelled with *Par Begin* and *Par End*. *Par Begin* indicates the start of execution of the parallel activities, which both start from P_1 and P_2. The *Par End* where the end of the execution of the parallel activities is reached and the tokens are placed in P_5.

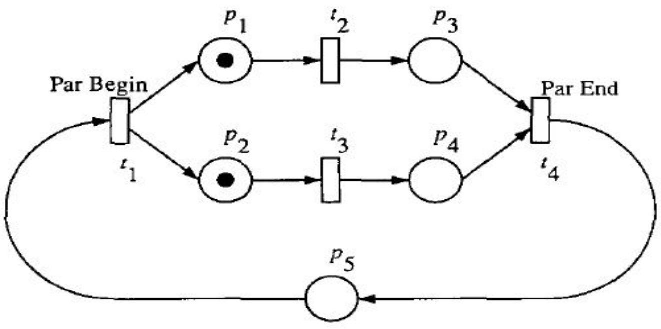


Figure 2.5 – An example of a Petri-net model executing parallel activities extracted from [100]

2.3.2 Timed Petri-nets

Timed Petri-nets are extended version of Petri-nets which associate a time with firing transitions and assign an age for a token. Diaz in [47] offered a simplified definition for timed Petri-nets:

Definition 1 *A classical timed Petri-net as defined in [47] is four tuples* (P, T, A, I) *[47] where:*

P is a set of **places**, T is a set of **transitions**, and $A \subseteq (P \times T) \cup (T \times P)$ is a set of **arcs**. I is time interval functions, which can be associated with any transition $t \in T$ in the Petri-net. The time interval function is bounded with minimum and maximum rational values in which $I(t) = [min, max]$ and restricted to be $0 \leqslant min \leqslant max$. The max might be infinite. For a transition t, the smallest of these times is called the earlier static date of firing t and is denoted as $Min(t)$. The largest one can be referred to as the later static date of firing t and is denoted as $Max(t)$ [47].

2.4 Coloured Petri-nets (CPNs)

A coloured Petri-nets (CPNs) as defined in [71, 73, 74] is "a graphical language for constructing models of concurrent systems and analysing their properties" [74]. A coloured

Petri-net is a modelling language that integrates the features of both Petri-nets and functional programming [74]. The CPN tool is empowered with graphical notions and a declaration of complex sets which make it easier to model complex and concurrent systems. Furthermore, CPNs use a programming language called CPN ML which was developed on the basis of the Standard ML functional programming language [74]. Standard ML is a language which mainly was developed for theorem proving [96, 108]. The CPN ML language is used to provide either primitive or complex declarations for data types. In addition, the language is used to build functions that manipulate the values which are stored into the CPN tokens. As a result, CPNs can be applied to model and verify dynamic behaviours for complex and distributed applications [71, 73, 74].

In a CPN, a token may have a complex data type as in programming languages. In addition, in a CPN model, each place has a correspondent data type. As a result, specifying a place with a data type restricts the types of the tokens that the place may receive [37]. Furthermore, CPN transitions process the values of the received tokens and create new ones, which can be from different data types. In a CPN, data types can be abstract or have a hierarchical structure. As a result, complex data types can be defined for places and tokens to describe complex structure. Hence, using CPNs can produce a concise model for systems [73]. The following section provides the mathematical description of CPNs.

2.4.1 The Formalism of Coloured Petri-nets (CPN)

Before presenting the mathematical syntax of CPNs, we briefly explain the definition of a multiset in a CPN. The multiset is composed of expressions that use the markings, steps, and occurrence of transitions in CPNs [71, 74]. Formally, Jensen and Kristensen [74] defined CPN multisets as below:

Definition 2 *Let $S = s_1, s_2, s_3, \ldots$ be a non-empty set. A multiset over S is a function $m : S \to \mathbb{N}$ that maps each element $s \in S$ into a non-negative integer $m(s) \in \mathbb{N}$ called the number of appearances of s in m. A multiset m can be written as a sum [74]:*

$$\sum_{s\in S}^{++} m(s)'s \quad = m(s_1)'s_1 + +m(s_2)'s_2 + +m(s_3)'s_3 + + \ldots$$

An example of the expression of a multiset is the multiset m_B that represents the colour set $NOXData$ such that the type of $NOXData$ is the product of INT and String which is written in CPN syntax as $colset\ \ NOXDATA = productNO * DATA$; [74]

$$m_B \quad = 1'(1, "COL") + +3'(2, "OUR") + +2'(3, "ED")[74]$$

To elaborate the previously mentioned expression, m_B is a multiset which has a one colour set of value $(1, "COL")$, three colour sets of $(2, "OUR")$ and two colour sets of $(3, "ED")$. In the expression of multiset, the symbol ++ means concatenation. In Section 2.4.3, there is an example which explains multiset in detail.

Mathematically, CPNs are described according to Jensen and Kristensen [74] as:

Definition 3 *A non-hierarchical Coloured Petri-Net (CPN) is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ where :*

P is a finite set of **places** and T is a finite set of **transitions** such that $P \cap T = \emptyset$. Places and Transitions are connected via a set of directed **arcs** denoted as $A \subseteq (P \times T) \cup (T \times P)$. Σ is a finite set of non-empty **colour sets**. V is a finite set of **typed variables** such that $Type[v] \in \Sigma$ which can be used on an arc expression to bind values. $C : P \to \Sigma$ is a **colour set function** that assigns a colour set to each place. In CPNs, a guard has a **guard function** that assigns $G : T \rightarrow EXPR_v$ to each transitions t such that $Type[G(t)] = Bool$, i.e. a Boolean value. $E : A \rightarrow EXPR_v$ is an **arc expression function** that assigns an arc expression to each arc a such that $Type[E(a) = C(p)_{MS}]$, where p is the place connected to the arc a. Examples of **arc expressions** that might be found in CPN models are $(var_a > 10)$ and $(var_b > 5 \ andalso \ var_b \leqslant 20)$. In CPN, $I : P \rightarrow EXPR_0$ is an **initialisation function** that assigns an initialisation expression to each p [74].

2.4.2 Markings in CPNs

In CPNs, a marking M consists of a combination of a mapped place p_i into a multiset of values $M(p_i)$ such that tokens in a place p_i represent each individual element in the multiset $M(p_i)$ [71, 74]. It is necessary that a multiset of tokens in a place p_i should have a similar type to the place p_i [71, 74]. The markings will be clarified via examples explained in Subsection 2.4.3. Semantically, the concepts related to markings in CPNs are described as follows:

Definition 4 *For a $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ both a marking and the initial marking are defined as:*

1. *"A **marking** is a function M that maps each place $p \in P$ into a multiset of tokens $M(p) \in C(p)_{MS}$. $C(p)_{MS}$ is the notion of multiset of tokens in a place p in a CPN model.*
2. *The **initial marking** M_0 is defined by $M_0(p) = I(p)$ for all $p \in P$ where $I(p)$ is an initialisation function of a place p in a CPN model" [74]*

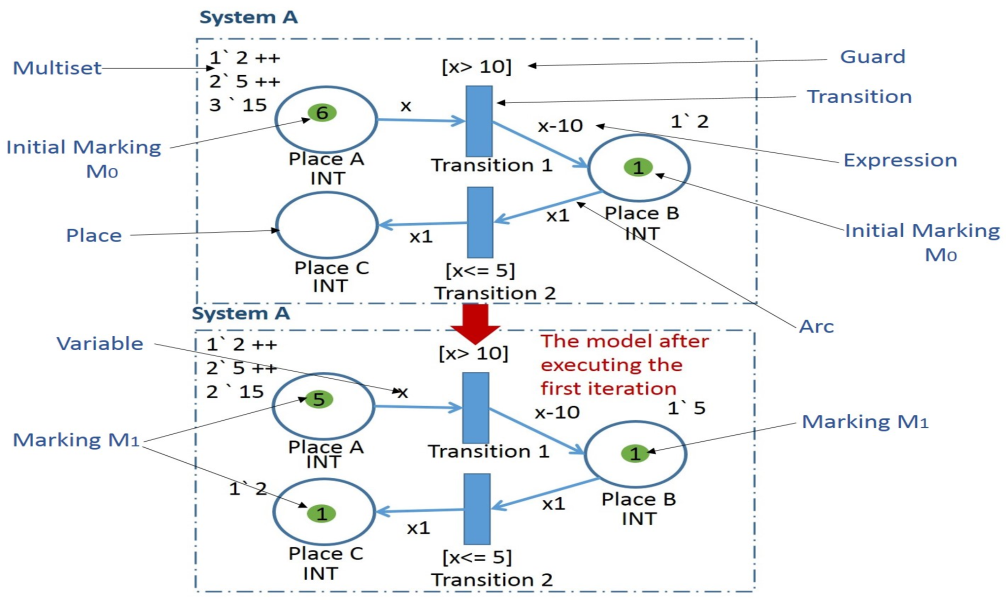


Figure 2.6 – Modelling **System A** with Coloured Petri-net

2.4.3 An Illustrated Example

To illustrate the CPN modelling concept, we provide the following example which is captured in Figure 2.6. Figure 2.6 illustrates a model of a CPN for System A, which performs calculations. In Figure 2.6, Place A has six tokens, one token with a value of 2, two tokens with values of 5, and three tokens with values of 15. In contrast, Place B has only one token with a value of 2, and Place C has no token. After the first execution for this model, the number of tokens at each place changes to be five tokens in Place A, one token in Place B, and one token in Place C. To illustrate an example of a multiset based on Definition 2 using Figure 2.6, the multiset which presents the values of the initial marking M_0:

$$m_A \;++\, m_B \;++\, m_C \;\; = 1'2 \;++\, 2'5 \;++\, 3'15 \;++\, 1'2 \;++\emptyset$$

However, for marking M_1 the values of the multisets change as follows:

$$m_A \ ++\, m_B \ ++\, m_C \ = 1'2 \ ++\, 2'5 \ ++\, 2'15 \ ++\, 1'5 \ ++\, 1'2$$

With each iteration, the number of tokens and their values will be changed. All possible values that each place can have are captured in a set of traces of execution. More examples presenting the functionalities and the capabilities for modelling with CPNs can be found in [71, 73, 74] and as well as on the official CPN website [5].

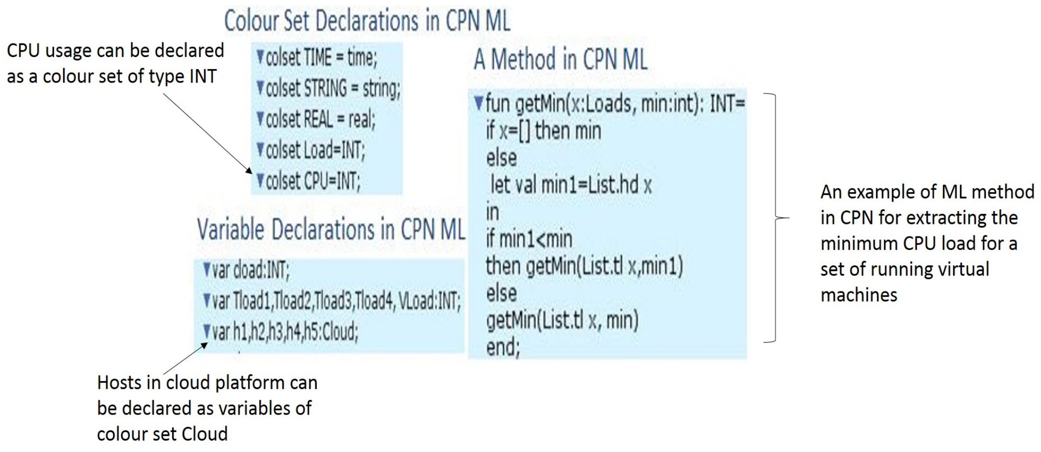
2.4.4 CPN ML Programming and CPN Tool

A CPN can be created using the CPN ML tool, which was developed by the CPN Group at Aarhus University [5]. The CPN programming environment uses the SML/NJ programming environment and extends it with constructs for defining colour sets and declaring variables [71, 74]. In addition, CPN ML allows users to write functions which can be used as arc expressions and guards in the CPN model. Furthermore, the CPN ML environment applies the concept of multisets and functions for the manipulation of multisets [71, 74].

Figure 2.7 captures some sample ML syntax used in the CPN ML tool. ML declarations for colour sets and variables are at the left hand side of the figure. A colour set is declared using the keyword “colset”. For example, “colset Load=INT” declares a colour set called “Load” which is of type integer. The right hand side of the figure provides a sample ML recursive function which returns the minimum load value from a list of type “Loads”.

2.4.5 The Formalism of Timed Coloured Petri-nets

To make CPN useful for testing the performance of a system and for supporting an accurate analysis, CPN has been extended to include the time concept [72]. A CPN model can be transferred using the CPN Tool [5] to a timed CPN model by initialising a global clock, which can be continuous or discrete [72], ages as stated in [74] or time-stamps


Colour Set Declarations in CPN ML
CPU usage can be declared as a colour set of type INT
A Method in CPN ML
Variable Declarations in CPN ML
An example of ML method in CPN for extracting the minimum CPU load for a set of running virtual machines
Hosts in cloud platform can be declared as variables of colour set Cloud

```
▾colset TIME = time;
▾colset STRING = string;
▾colset REAL = real;
▾colset Load=INT;
▾colset CPU=INT;
```

```
▾var doad:INT;
▾var Tload1,Tload2,Tload3,Tload4, VLoad:INT;
▾var h1,h2,h3,h4,h5:Cloud;
```

```
▾fun getMin(x:Loads, min:int): INT=
 if x=[] then min
 else
  let val min1=List.hd x
 in
 if min1<min
 then getMin(List.tl x,min1)
 else
 getMin(List.tl x, min)
 end;
```

Figure 2.7 – Samples of ML syntax used in CPN Tool

for tokens [72] and waiting time (or delays, as they are called in the timed PN definition) for firing some or all defined transitions in the CPN model (See Chapter 5 in [72] and Chapter 11 in [74]).

The values for both the global clock and timestamps of tokens are presented as a set of time values belonging to T. T is a set of non-negative integers which is declared in the CPN tool as type TIME [74]. The description of the syntax of timed coloured Petri-nets is similar to the definition of untimed coloured Petri-nets mentioned in Definition 6. Jensen and Kristensen [74] stated:

Definition 5 *A timed non-hierarchical coloured Petri-net is a nine-tuple* $CPN_T = (P, T, A, \Sigma, V, C, G, E, I)$ *[74] where:*

P is a finite set of **places** and T is a finite set of **transitions** such that $P \cap T = \emptyset$. Places and transitions are connected via a set of directed **arcs** denoted as $A \subseteq (P \times T) \cup (T \times P)$. Σ is a finite set of non-empty **colour sets** which can be either timed or untimed. V is a finite set of **typed variables** such that $Type[v] \in \Sigma$ which can be used on an arc expression to bind values. $C : P \to \Sigma$ is a **colour set function** that assigns a colour set to each place such that if $C(p)$ is timed, a place p is said to be a timed place. In

CPNs, a guard has a **guard function** that assigns $G : T \rightarrow EXPR_v$ to each transition t such that $Type[G(t)] = Bool$, i.e. a boolean value. $E : A \rightarrow EXPR_v$ is an **arc expression function** that assigns an arc expression to each arc a such that it can be either $Type[E(a) = C(p)_{MS}]$ if p is untimed or $Type[E(a) = C(p)_{TMS}]$ if p is timed, where p is the place connected to the arc a. $I : P \rightarrow EXPR_0$ is an **initialisation function** that assigns an initialisation expression to each p such that $Type[I(p)] = C(p)_{MS}$ if p is untimed or $Type[I(p)] = C(p)_{TMS}$ if p is timed [74].

Similar to the marking definition for CPNs explained in Subsection 2.4.1, the concepts of markings and initial markings in timed CPNs are specified as follows:

Definition 6 *For a $CPN_T = (P, T, A, \Sigma, V, C, G, E, I)$ both a marking and the initial marking are defined as:*

1. *"A **marking** is a function M that maps each place $p \in P$ into a multiset of tokens such that*
   - *$M(p) \in C(p)_{MS}$ in case p is untimed. $C(p)_{MS}$ is the notion of multiset of tokens in a place p*
   - *$M(p) \in C(p)_{TMS}$ in case p is timed. $C(p)_{TMS}$ is the notion of multiset of timed tokens in a timed place p*
2. *A **timed marking** is a pair (M, t^*) where M is a marking and $t^* \in \mathbb{T}$ is the value of the global clock. The global clock is a time assigned for a CPN model, or in other words the simulation time of the CPN model.*
3. *The **initial timed marking** is the pair $(M_0, 0)$ such that $M_0(p) = I(p)$ for all $p \in P$" [74].*

The formal definition for timed CPNs can also be found in [72] but we chose the formal definition stated in [74] since it uses notions similar to the one we applied in this thesis.

2.5 Integer Programming and Branch and Bound Algorithm

Linear Programming (LP) problems involve searching for an optimal value of an objective function. The optimal value can be either the minimum or the maximum. In the objective function, the decision variables are subject to a defined set of constraint equations. In LP, the decision variables must be non-negative. Similarly, Integer Programming (IP) problems are also concerned with finding the optimal value, but the decision variables defined in the objective function and the constraints are non-negative integers [29]. Therefore, an IP problem is defined as " any decision problem with an objective to be maximised or minimised in which decision variables must be discrete values" [118]. Formally, the integer optimisation is defined as follows:

$$z = \sum_{j \in N} c_j x_j \tag{1}$$

subject to:

$g_i(x1, x2, \ldots, x_m)\ \{\leqslant, =, \geqslant\}\ b_i\ \ , x_j \geqslant 0$

Equation 1 is an objective function that can be either minimised or maximised. c_j is a coefficient and x_j is a decision variable. g_i is a constraint function where $i = \{1, 2, \ldots, n\}$ and $j = \{1, 2, 3, \ldots, m\}$.

The IP problem can be solved using a branch and bound algorithm. The branch and bound approach is based on the following concepts:

1. *Branching:* Partitioning the continuous space into sub-problems for eliminating continuous spaces that do not include feasible solutions. From partitioning, we get subsets of optimal integer points that define a feasible solution of the original problem [118].

2. *Bounding:* Ranking or enumerating the obtained integer points extracted from the branching method [118].

The algorithm deals with continuous space which systematically subdivides the linear programming feasible region resulting in the creation of an *enumeration tree* and then makes assessments using the defined constraints based upon these subdivisions in order to round the feasible decision variables to be integer points. In the branch and bound approach, the number of sub-problems may grow exponentially [29] (more details about the algorithm can be found in [29] and in [118]). Excel Solver solves IP problems using the branch and bound combined with Simplex [8]. Simplex is an algorithm which is used to solve LP problems [117]. We used Excel Solver to find the integer points that provided us with the minimum energy consumption cost, which will be explained in Chapter 7.

2.6 Types of Virtual Machine Migration for Management

This thesis primarily deals with management energy consumption through triggering virtual machine migration action. In this section, we provide an overview of migration in a virtualised environment. Dynamic migration is the action of moving a virtual machine from one host to another while the virtual machine is executing. Dynamic migration has two types, which are live-migration and stop/resume migration. During the live-migration procedure, the memory of the migrating virtual machine is copied to the destination host without stopping its execution iteratively [88]. The virtual machine's execution is halted only to perform the final synchronisation, which is around $60-300$ ms at the configuration of data centres before beginning the running at the destination [88]. Live-migration gives the illusion that the virtual machine does not stop [88]. Live-migration is one of the functionalities that is embedded in a virtual machine manager (VMM) or cloud management system. For example, OpenNebula offers both types of virtual machine migration [120].

Live-migration is different from stop/resume migration. In a stop/resume migration process, the virtual machine is required to stop before migrating to a destination host. The migrated virtual machine continues its execution after it is completely and successfully transmitted to the destination [88]. Stop/resume migration is beneficial for migrating virtual machines that execute critical jobs, which might be sensitive to losing data during transmission time.

2.7 The Classification of Automated Management Architectures in Large-Scale Platforms

One of the contributions of this thesis is the design of an automated framework for management in a cloud platform considering management objectives assigned by the cloud provider, which will be discussed in both Chapter 3 and Chapter 4. Therefore, before discussing our management framework, we will provide a survey of the existing research in designing automatic architectures for managing and re-allocating resources in a large-scale platform which includes a distributed environment, a cloud platform, and a virtualised environment. Those architectures have an influence on how to develop our architecture which will be explained in Chapter 3.

The existing automatic architectures are classified into three groups based on the essential technique applied in each of the reviewed studies. The existing classified architectures are: two-level automatic architectures based on utilising the concept of control theory, automatic architectures based on heuristic approaches, and automatic architectures based on rule-based systems. The following subsections provide overviews of research related to each category.

2.7.1 Two-level Automatic Architectures Based on the Concept of Control Theory

The first type of automatic management architecture includes those that employ the concept of using control theory in designing an autonomic system or a self-managed system suggested in [135]. The suggested automatic architecture in [135] includes a number of components which are sensors, actuators or effectors, and a controller. Sensors are used to collect the control inputs which are parameters a controller needs to determine the state of the target system. The actuators or effectors are the components that apply the desired output in order to make the system reach the target objective. The controller is the main component that uses a dynamic model based on the received input and other entities that define the state of the self-managed system. The state of such a system can be computed by using either the concept of *Utility-Function* as applied in [63, 97, 121, 124] or the dynamic models used for describing the behaviour of the target system as suggested in [41, 58, 61, 111, 131].

A utility function is an objective function that assigns the state of the independent components of the target system to real scalar values which are expressed in a suitable unit [124]. The state of the system is expressed as a vector of attributes which are provided either directly by measurement instruments or by synthesised components plugged into the running system [124]. The usage of the utility function should be combined with optimisation techniques in order to provide a feasible solution [124]. Applying a utility function combined with the concept of control theory is one of the methods used for building autonomic systems [124]. The following section discusses the existing automatic architectures that use the concept of control theory based on utility functions.

Two-level Automatic Architectures Based on Utility Function

The usage of utility functions in large-scale and distributed systems is implemented for designing architecture of automatic managed systems. Examples of large-scale systems that employ a utility function are the distributed autonomic environment as proposed in [124] and cloud platforms as suggested in [58, 63, 97, 121]. Distributed autonomic environments share similar characteristics with cloud platforms, which have a large number of components that work independently in a heterogeneous platform. Therefore, we will first explain the work in [124] before explaining the applicability of utility function in cloud platforms.

In [124], Walsh *et al.* proposed a two-level architecture that uses two types of utility functions for managing a set of independent autonomic components. The automatic management aims at optimising the computational resource allocation for running various distributed applications' environments. An example of the distributed environment is running different web-based applications published in a cluster of web servers [124]. Two utility functions are defined by Walsh *et al.* which are *service-utility* and *resource-utility* functions. The *service-utility* functions are implemented locally at the application level, whereas the *resource-utility* functions are applied globally at the system level. The *service-level* function determines the business value to the user of the running service. In contrast, the *resource-level* utility function specifies the amount of computational resource allocated to run an application based on values of resource demand [124].

The approach of Walsh *et al.* is considered to be a two-level centralised architecture which can be applied in a distributed environment such as cloud platforms. In Walsh *et al.*'s architecture [124], the separation techniques for multiple environments and using two different utility functions allows the implementation of such an architecture into heterogeneous and complex environments such as a cloud infrastructure. The introduction of a level of abstraction applied in Walsh *et al.*'s [124] automatic architecture simplifies the modification of the design of the resource arbiter. All the complex functionality applied in the architecture is hidden from the higher level and handled by a local manager which is configured at the application level [124].

Similar to the adaptation of the utility function applied in [97], the application of utility functions has become more mature and enhanced to be suited to cloud platforms. The implementation in a cloud environment considers various management aspects which are business, technical, and environmental. In [97], a utility function is used for designing a resource manager for automatically allocating various types of resources using a feedback control loop as described in [77]. The main objective is satisfying the quality of service and increasing the resource utilisation. The utility function is modelled to consider the total shared resources (CPU , memory, network, and disk) of a virtual machine. Furthermore, the utility function includes a performance model of the running virtual machines which is mapped as measuring the response time for the submitted jobs (see the utility model in [97]).

In [97], the proposed architecture of automatic architecture is organised to use a different set of managers that implement a feedback control loop to control the level of resources provided to a set of running virtual machines. Each resource manager is applied to adjust resource utilisation among independently running virtual machines by maximising of the utility of each running virtual machine. The designed controller considers the summation of both shared resources and the response time of each virtual machine. The summation is used to determine virtual machines that have the highest utility function which are arranged in descending order [97] or placement into available hosts. The architecture proposed in [97] can provide flexibility at the node level. However, at the resource level, the architecture is complex and does not provide flexibility to support heterogeneous platforms.

In [121], there is a proposal for an architecture for a cloud manager that will automatically manage the provisioning and the placement of virtual machines. The automated manager is designed to consider several properties, namely the performance of the running application, SLA specifications, and both resource exploitation and operational costs. The architecture consists of two layers associated with two decision modules: an *Application Decision Module* and a *Global Decision Module*. The *Application Decision Module* uses an analytical performance model to analyse application response time based on long-term observation. In addition, the *Application Decision Module* consists of two levels of utility functions which are related to service-level and resource-level. Using measured performance information and SLA requirements, the *Application Decision Module* optimises the number of used virtual machines and the amount of the required resources by employing a constraint programming method [121]. Conversely, the *Global Decision Module* makes decisions related to allocating the correct set of virtual machines to each running application. In addition, the Global Decision Module packs the running virtual machines on physical machines via applying *live-migrate* action in order to reduce the number of running hosts. Therefore, the global controller solves two problems, which are referred to as VM Provisioning and VM Packing in [121].

The framework suggested by Van *et al.* in [121] has some shared properties with the autonomic architecture proposed by Walsh *et al.* in [124], which was previously explained. The similarities are in using two-level controllers which are application-specific and resource-specific managers. The application manager functionality measures the current resource usage and adjusts it based on the current application demands as in [124]. However, in [121], the application level uses two types of utility function compared to [124] which applied service-utility function only at the application level. We noticed that the suggested framework by Van *et al.* in [124] handles multiple problems which are validated through simulation. However, applying such a framework in real cloud platforms can become insufficient unless a single controller considers solving only one problem, such as the resource provisioning problem, rather than multiple problems. Furthermore, based on our experience with OpenNebula [105], completing the live-migration actions requires time which might increase to minutes. As a result, the global manager in [121] might introduce a performance issue if it is applied in real cloud platforms such as OpenNebula [105]. This is because the framework allows repeated triggering of live-migration actions for multiple virtual machines simultaneously, which has an effect on the quality-of-service of applications. This issue will be discussed in Chapter 3. Therefore, the automatic architecture proposed in [121] is complex in performing multiple operations randomly triggered in real cloud platforms. Thus, Van *et al.*'s automatic manager architecture [121] can only be useful for creating an off-line management plan.

In [61], Gueyoung *et al.* presented a centralised multi-level automatic management architecture for re-allocating migrated virtual machines. The techniques are based on using off-line modelling combined with an online prediction workload model to collect the information that is necessary for reallocating actions. Furthermore, the approach applies a multi-level adaptation hierarchy and scalable heuristic optimisation techniques based on re-allocating the migrated virtual machines. The main objective is to control the virtualised environment using cost, performance, and power consumption models. The introduction of the use of off-line modelling which can be profiled is a helpful method for evaluating long-term reallocation plans that depend on finding the average workload peak time [61].

Two-level Automatic Architectures Based on Dynamic Models

Gmach *et al.* in [58] provided a centralised resource management system which is capable of automatically organising a shared pool of servers. Like the previously mentioned architectures, the centralised resource manager in [58] is organised into levels. The local controller implements a feedback control loop which periodically measures the workload capacity. As with previous approaches, the local controller, which was specifically named in [58] as *Workload Management Service*, operates at the application environment level. On the top of the local controller, a system global controller is responsible for placing and deploying the correct set of virtual machines according to the data collected from the local controllers. In [58], the main objective is to keep the operating server numbers in line with workload demands for running services. This can be achieved by designing a workload adjuster which prioritises the received workload using a Supervise SLA Compliance associated with each running virtual machine. The local controller is implemented using various policies for adjusting assigned resources. These policies are explained in detail in [58].

In [58], there is a claim that this approach can control the energy consumption at the data centre, which can be accomplished by analysing historical workload traces for the various implemented policies using simulators. The outcome from applying such analytical models endows the global controller of the architecture with an ability to predict the workload peak time and to update the hosting routing resource allocation table [58]. It was found that the centralised architecture suggested in [58] can be applied for designing dynamic resource management that prioritises SLA parameters. Despite the introduction of SLA compliance and the use of the historical workload analytical model, this approach fails to provide the dynamic models used by the main controller, which makes the architecture more complex to be implemented in the management cases that are considered in this research.

In [41], Cunha *et al.* developed an automatic management resource capacity framework. The framework was described as a multi-tier performance model combined with a service-level pricing model and an optimisation model. The optimisation model is used to adjust the resource usage among applications. The framework runs at periodic times, where the measurement information is collected to forecast the expected workload on the system. The performance analysis is presented by transforming the measured performance metrics such as the response time to an estimated analytical model based on a queuing model. The optimisation is achieved based on current capacity information and the estimated performance model. The problem is formulated as mathematical models. The interesting aspect of this approach is the attempt to use performance models and workload prediction for automatic management [41].

In [83], Kusic *et al.* proposed an automatic architecture which implements a type of control theory called limited look-ahead control. The design is organised into multi-level controllers for resource provisioning in a virtualised environment. The controlling problem is decomposed into a set of smaller sub-problems and solved in a cooperative fashion by multiple controllers [83]. The level of controllers provides two types of information, which are virtual machines performance and power information for physical nodes. Each controller implements limited look-ahead control to predict and to determine management actions over a short slot time period (more details about the models used are found in [83]). In [83], the suggested approach is suitable for an architecture using a rule set or utility function for triggering a set of management actions. Although the introduction of limited overhead is novel in this approach, it is a complex model which is difficult to apply in practice.

In [111], the authors attempted to design an automatic management architecture for provisioning cloud resources to applications. The objective was to maximise the QOS for applications with respect to budget constraints. They defined a set of adaptive parameters used by an application. Those parameters are the input for a centralised manager that uses a feedback control loop to reallocate resources. The feedback loop controller uses a defined resource cost model, the virtual machine resource model (CPU and memory), and an analytical performance model [111]. The controller design is well-defined, and it can be useful for reactive automatic management systems using simplified performance model parameters. It is centralised and uses static scheduling.

Wood *et al.* proposed a centralised automatic architecture utilising a feedback control loop based on two strategies [131]. The purpose of the architecture is to dynamically control the triggering of migration of virtual machines. In [131], the authors introduced a novel Black-box and Grey-box of instrumenting strategies for monitoring and describing the resources-utilisation model [131]. The main objective of using the Black-box strategy is to monitor resource utilisation by observing the external behaviour of virtual machines with no attention to either application type or any dependencies of the virtual machine's hosting environment [131]. The suggested Black-box strategy can collect sufficient resource usage data to determine a detection of migration alarm [131]. However, the resource usage details provided by applying the Grey-box strategy presents more accurate data than the Black-box one, but the latter approach reduces the monitoring time intervals and the system overhead [131]. In [131], the feedback controller aims at detecting the migrated virtual machine by using the profiled information provided from the instruments and applying prediction techniques. The migrated machines are re-allocated using greedy heuristics [131].

2.7.2 Automatic Architectures Based on Heuristic Optimization Approaches

The second type of reviewed architectures are those that use heuristic approaches to help trigger a management action in a cloud platform. In this classification, Jing and Fortes proposed a two-level architecture for managing the mappings of the correct workloads to potential VMs as well as VMs to physical resources [76]. The VMs placing problem was formulated as a multi-objective optimisation problem which considers simultaneously minimising total resource wastage, power consumption, and thermal dissipation costs [76].

The placement optimisation problem is solved using an improved genetic algorithm with fuzzy multi-objective evaluation. The usage of a fuzzy logic approach is to provide an efficient search by resolving possibly conflicting objectives [76].

The proposed architecture is similar to the previously explained two-level architectures. The local controller which is assigned to each running virtual machine is located at the application level. The global controller runs using defined monitored parameters related to both the virtualised environment and the data centre level [76]. The monitored parameters are profiled for generating both suitable power and temperature models. The functionality of the global controller is to generate a new virtual machine placement and migration scheme [76] by implementing the Modified Genetic Algorithm. The local controller uses fuzzy logic-based modelling approaches to adaptively model the relationship between workloads and virtual machine resource demands [76]. In [76], the evaluation is done through a prototype demonstration which provides an estimation for resource demands responding to dynamically changing workloads [76].

The algorithm can be used to work with static resources requirement information or dynamic information. If static information is used, the system uses only instrument and usage profiles for creating resource usage information. In contrast, providing dynamic information requires the definition of a local controller that is used to periodically monitor information which is missing in the architecture suggested in [76]. However, we think that the algorithm can be beneficial for creating an off-line plan for migrating or placing virtual machines in order to reduce system overhead, but the suggested algorithm needs to be combined with a method for modelling performance for each running virtual machine.

Similar to the approach suggested in [76], a suggestion to use an ant colony optimisation algorithm to reduce the number of running hosts in order to reduce energy cost at a virtualised data centre was presented in [51]. The authors mapped the migration problem as a classical bin packing problem [25] with an objective of minimising the operational cost to fully utilise servers [51]. The use of an ant colony in this context needs to be combined with a multi-objective problem rather than a single one to be effective. Otherwise, a heuristic algorithm should be used because it is easy to implement and would be beneficial in practice. It is necessary for a data centre operator to maximise the profit which makes it too important to consider application requirements or higher level demands.

2.7.3 Architectures Based on a Rule-based System

The third existing automatic architectures are the architectures that employ rule-based systems for triggering various types of actions applied in a cloud platform, a virtualised environment, or a service-oriented architecture. Applying rule-based approaches is an effective component in designing an automated architecture in cloud platforms. The development of such architecture is found in [26, 91, 92].

In [91], there is a proposal to use a knowledge-based system in automatic management in cloud computing. The goal from the proposed architecture is to prevent the violation of SLAs. The implementation of a knowledge-based system is accomplished mainly by using case-based reasoning (CBR) combined with a rule-based system. In [91], the role of a rule-based system is to receive some measurable metrics important to the executed SLA assigned to the running virtual machines. These retrieved parameters are compared with a set of specified threat thresholds which are used to determine the state of the virtualised environment. The rule-based system would trigger the CBR system. The CBR system, using the measurable values of the SLA over a specified time interval, attempts to select a new case which has the highest utility function. Based on the case, the new reactive action can be selected and applied to the cloud system. Afterwards, the system is supposed to monitor the parameter again in order to evaluate the selected action after execution [91]. In [91], a rule-based system is used only to determine the state of the system whereas the main selected management action is decided using CBR. The architecture is executed in the monitor-analysis-plan-execute cycle. The approach suggested in [91] can be beneficial for achieving long-term management goals. However, the application of CBR is time-consuming for achieving short-term management as covered in this research.

The rule-based approach in [91] is enhanced in both [26] and in [92] to include either simple rules or default logic rules. The rule-based system is also used to trigger a set of reconfiguration or adaptation actions. In [92], the proper management action is identified based on defining the current state of the system and selecting the best policy model the virtual machine should apply. In contrast, in [26], the rule-based system uses simple rules based only on resource consumption to trigger virtual machine migration actions. Then, an allocation algorithm, which can be First-fit, RoundRobin, or Monto-Carlo, is used to reallocate migrated virtual machines [26].

The usage of a rule-based system in both [26] and in [92] has some similarities to the approach in my work, which is discussed in Chapter 3. However, the rules considered are low-level, neglecting high-level elements such as time, location, and SLAs. Furthermore, the rule-scheme provided in [92] is complex in its structure and does not have an effective data-domain model as we outlined in Chapter 4. Thus, it is concluded that the rule scheme suggested in [92] cannot be executed in architectures running in a real cloud platform due to the lack of a defined, well-presented architecture for the proposed system. Furthermore, the approach has an apparent misconception in the application of rules and rule-engine in performing autonomic management.

In [122], Vaquero *et al.* proposed an automatic architecture for reconfigurable cloud applications at run-time. The architecture is based on a customised rule-engine which enables the execution of a set of rules to govern the application behaviour. Application providers can update application behavioural policies during run-time, such as adapting new load conditions. The OVF description domain model language for virtualisation, which is composed of vocabulary descriptions for VirtualMachine, HardwareComponent, Service, VirtualDataCenter, etc., has been used for representing the domain knowledge that is to be used by the engine. Furthermore, the Semantic Web Rule Language (SWRL) is used to enable an easy definition of high-level policies for defining application behaviour on top of the static. It is argued that the architecture performance is based on the performance of the rule-engine [122]. The architecture does not provide a description of the types of policy or how rules can be expressed in the policy. The usage of the OVF domain description language can increase the probability of policies among various cloud-based applications. This architecture is different from the architecture presented in Chapter 3 since the generic architecture uses a rule-engine that controls the monitoring side and the management side. In addition, a classification for rules used in the architecture has been provided.

The rule-based system is a part of the automatic architecture applied in a service-oriented environment which has some similar aspects to a cloud platform in terms of an increase in dynamism as well as its complexity in the interaction among its running components or services [81]. In [114], Rosenberg and Dustdar developed an automatic architecture for business brokers in a service-oriented platform. The architecture is based on the deployment of various rule-engines. This approach provides a service layer interface for accessing and executing business rules from various knowledge bases. Furthermore, the business rule brokering layer allows heterogeneous rule engines to be encapsulated and used. Various rule-engines can be plugged in using the adapter pattern [116]. The rule-based knowledge is a web service which can be accessed remotely [114].

2.8 The Rules Modelling Languages and URML Specification

The second contribution of this thesis is to develop a specification for describing management policies in an executable rule language such as the Drools Language [11], which is explained in Chapter 4. As a result, this section includes an overview of some existing rule modelling languages and the specifications of the UML-based Rule Modelling Language (URML).

Rules can be described in a simplified manner by using modelling languages combined with a well-defined transformation methodology. There are a number of modelling languages for describing rules. Some of these languages are Semantics of Business Vocabulary and Business Rules (SBVR) [4], Simple Rule Mark-up Language (SRML) [4], UML-based Rule Modelling Language (URML) [112], and Business Process Modelling Notation (BPMN) [3].

SBVR is a language that attempts to provide a definition of a standardised rule modelling vocabulary [4]. SBVR presents a vocabulary that is intended to become a standard upon which many grammars can be based for specifying rules. SRML is also a descriptive language which can represent rule models but with a limited vocabulary [6]. On the other hand, URML [112, 90] is a graphical representation for rules, which supports modelling domain vocabularies (i.e., ontologies) and various types of rules. In URML, a rule is represented as a circle with identifiers, a condition arrow, and a conditioned model element. One benefit of URML is that rules can be translated to an event-condition-action rule structure [112].

In [46], there is an attempt to use BPMN, which is a graphical modelling language proposed by OMG [4]. BPMN is a collection of graphical representations that can be used to describe a business process. BPMN is used in [46] to provide a high-level graphical description for simple rule patterns. The objective is to simplify the expression of the rules used in business applications. The transformation of the graphical representation for rules in BPMN is accomplished by generating a methodology for mapping to Drools Rule Language [46].

All the presented languages can be used to provide a high-level description for the rules applied in the generic architecture MP-Framework explained in Chapter 3. However, URML was selected for defining the specification of management policies discussed in Chapter 4. Therefore, the following subsection explains the abstract specification of URML.

2.8.1 The Specification of URML

REWERSE (Rule Modelling and Markup group) [112] has published a meta-model of the URML language which classifies various types of rules that can be formulated to be executed with rule-engines such as Drools [11]. Figure 2.8 presents the structure of a rule in URML which can also be applied to a rule language. In this research, we are concerned only with the condition objectVariables as shown in Figure 2.8. Furthermore, the specification of production rules was also selected. Other rules existing in the models are beyond the scope of this research. It can be seen in Figure 2.8 shows that a production rule can consist of one or more conditions, zero or one post conditions, and one action ActionEventExpression which includes the statements of a rule action part. In URML, any formulated rule should be related to one or two ObjectiveVariable.

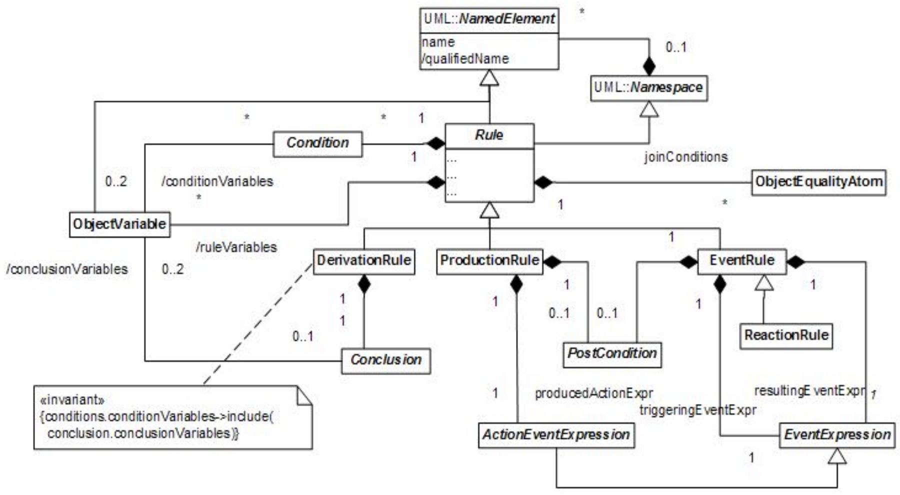


Figure 2.8 – The rules meta-model of UML-Rule Modelling Language (URML) as presented in [113]

2.9 The Existing Domain-Specific Languages (DSLs) Designed for the Cloud

A part of the second contribution covered in this thesis is related to designing a Domain-Specific Language (DSL) called CloudMPL for describing management policies during the design phase. Chapter 4 describes the language. In this section, we provide an overview of the existing domain-specific languages used in cloud platforms. DSLs provide special features in terms of the expressiveness and simplicity compared with general-purpose programming languages [94]. Using DSLs has several advantages. They can speed up the development time since the language is designed to be used in a specific environment. In addition, the language can assist in reducing the amount of domain and programming expertise required [94]. Furthermore, the domain language is extensible and machine-readable which allows users to build auto-code generation tools in order to reduce the development time [94]. To accomplish these features provided by DSLs, designing such languages requires experience in both domain knowledge and language development.

2.9.1 DSLs Used in Cloud Environment

Extensive research has proposed many DSLs for automating the deployment of applications into a cloud environment. One of these languages is Crawl, which is a part of the Cloud Crawler environment proposed for automating the execution of application performance tests in Infrastructure-as-a-Service (IaaS) used by cloud application developers [42]. Crawl is a declarative and extensible DSL to provide a high-level specification that captures all the important technical information for executing application performance tests [42]. Instances of this information are the configuration parameters and the quantity of the resources allocated to application components [42]. The language's textual notion is described via YAML. Furthermore, the language allows the use of XML and JSON to define new specifications of test scenarios [42].

Neptune is another DSL designed to automate the configuration and deployment of High Performance Computing (HPC) applications executed in the cloud [32]. The objective of Neptune is to provide portability and flexibility to the developers of HPC [32]. Neptune is a meta-programming extension of the Ruby programming language with the flexibility to run a large number of Ruby's libraries which are designed to communicate with a cloud infrastructure [32]. Neptune programs allow users to write Ruby scripting code. In addition, Neptune programs can also be used in Ruby programs using Neptune keywords. Neptune programs are composed of one or more invocations for jobs to be processed in cloud services [32]. The language is integrated to run in AppScale, which is an open-source cloud environment that uses Google App Engine APIs [32].

Pim4Cloud DSL is a platform-independent model for cloud-based applications which is designed using a component-based approach [30]. A cloud application-designer models the application by using Pim4Cloud DSL. Meanwhile, at the other side, the available resources for the modelled application are specified by the cloud provider [30]. Pim4Cloud has an interpreter which is used to match the assigned resources to the application's requirements. Pim4Cloud DSL is implemented into Scala, which includes different sets of codes for modelling different topologies for cloud applications [30]. The syntax of the Pim4Cloud DSL starts by defining the application as an abstract class which can factorise the shared entities. Each application topology can extend the abstract class. The Pim4Cloud DSL platform supports a static analysis for the modelled application and also allows the deployment of cloud components to be reused [30].

2.10 An Analytical Cost-Computing Method from Petri-nets

In Chapter 6 and Chapter 7, we propose two methods for computing the cost from the proposed CPN models for a cloud platform executing management policies. To develop such methods, we investigated the existing research on computing costs from Petri-nets. These include simple and optimal cost calculations from various Petri-nets which will be explained in the following sections.

2.10.1 The Cost Calculation Method from Petri-nets

The main cost analysis method which will be discussed in Chapter 6 is based on the method proposed for calculating the cost through Priced Timed Petri-nets (PTPNs) and Priced Petri-nets (PPNs). PTPNs have integer ages which represent the token creating time. Furthermore, transition arcs are assigned with time-intervals restricting the ages of the consumed and produced tokens. In [23], both PTPNs and PPNs are associated with multidimensional costs for discrete transitions and places in the models. In [16, 17], the discrete transitions are the ones that are triggered without time restriction. In both PTPNs and PPNs, the cost of discrete transitions is based on the assigned cost vector to the fired discrete transition. A PTPN has extra cost assigned to the timed transition which depends on the cost of the marking in trace σ [23].

From a trace σ that has the following format:

$$\sigma := \quad M_0 \longrightarrow M_1 \longrightarrow M_2 \ldots \longrightarrow M_n$$

Such σ consists of a set of markings and includes discrete and timed transitions. In σ, the cost of triggering a discrete transition t_{dis_i} is defined as $Cost(M_i \xrightarrow{t_{dis_i}} M_{i+1}) := C(t_{dis_i})$. Whilst the cost of triggering a timed transition t_{timed_i} is defined as $Cost(M_i \xrightarrow{t_{timed_i}} M_{i+1}) := \sum_{p \in P} M(p)| * C(p)$. Thus, the overall cost of a trace σ is the sum of all the computed cost of transitions which is defined as $\sum_{i=0}^{n-1} Cost(M_i \longrightarrow M_{i+1})$ [16, 17, 23].

In Chapter 6, we applied the previously explained method for calculating the cost and proposed alternatives suitable for the generated Coloured Petri-nets model for an automatic cloud platform (see Chapter 6).

2.10.2 The Optimised Cost Calculation Methods

Throughout our research, we needed to compute the optimal cost from traces of execution in CPN models of cloud platform. Therefore, a survey about computing optimal costs in Petri-nets models was required. The research provided in [16] and [17] was concerned not only with cost computation, but also with computing the minimal cost in both Priced Timed Petri-nets (PTPNs) and Timed Petri-nets (TPNs). They claimed that the minimal reachable cost may not exist for unbounded PTPNs and TPNs. Nevertheless, the minimal cost is computable for non-negative cost values if the problem is transformed to a cost threshold problem. The reachability graph is bounded with a set of final markings $M_f \in F$ and a vector of thresholds variables $v \in V$. Then, using the formula expressed in Section 2.10.1 such that the sum of cost values in a trace σ should be $Cost(\sigma) \leqslant [V]$. As a result, we concluded that the optimal cost can be computed in a PTPN if it becomes bounded, contains a set of reachable final markings, and also is associated with cost threshold variables. However, there is no such algorithm provided in [16] or [17] for retrieving the optimal cost from the markings, since their work only proves the existence of minimal cost using a reachability graph.

In [85], the problem of computing of the optimal cost was investigated from a different perspective. The objective was to develop an algorithm for computing the least cost plan for firing a transition sequence in a labelled Petri-net. The authors considered the reachability graph which is generated from firing labelled tasks as a trellis diagram with k length sequence [85]. The least cost planning is estimated by using a recursive algorithm. In [85], the reachability graph is transformed to a trellis graph. The trellis graph is a type of a state graph, which is a tool for the representation of finite state machines as ordered nodes based on the time occurrence of the state ([128], p.156). The algorithm developed by Li and Hadjicotis in [85] computes the cost at each node in the trellis graph at level i. Then, the node that has the least cost is determined at level i [85]. In [136], the recursive algorithm proposed in [85] was modified to find the least cost resource consumption sequence. The modified algorithm looks at transitions that have zero-cost which can be found in some Petri-net models [136].

The recursive algorithm proposed in [85] can be useful to compute the minimal cost in some Petri-net models. However, the algorithm would not be suitable for computing the minimum cost from the reachability graph generated from the CPN cloud models. The reason is that our CPN cloud model is a dynamic platform which does not have fixed cost values along the markings. In other words, if there is a marking M_a which has the lowest cost at $level_i$, this does not mean that the next followed marking at $level_{i+1}$ has the lowest value. Therefore, in our case, we have to compute the cost along the whole set of traces of the execution in the reachability graph in order to extract the traces which include the minimum total cost [85].

Similar to the least cost recursive algorithm, [45] developed a search-based algorithm for finding firing sequences from the initial state to the final state for a timed Petri-net model. For that algorithm, there is an assumption that a partially generated reachability graph is provided. The search process is guided by a heuristic function, which is based on firing count vectors of the state equation for predicting the total cost. Since this heuristic search exploits linear characteristics of the state equation, which contains sufficient global information; it can efficiently generate a near-optimal or optimal solution. However, we cannot use the heuristic search to find the optimal cost for the same reason that prevented us from using the recursive algorithm. Nevertheless, based on the study in [45], it was observed that the cost values in our model formulate a linear programming feature (this will be explained in Chapter 7).

2.11 The Existing Petri-nets Models in Cloud

Petri-nets and their extensions have been used in [87] and in [68] for modelling various aspects in cloud. Classical Petri-nets is proposed for modelling a charge model for Infrastructure as a Service used by a cloud provider [87]. Classical Petri-nets is extended to include a cost-profit function (CPPN). Both cost and profit are assigned as real fixed numbers, which are applied to transitions. Cost value is computed before firing a transition whereas profit is calculated after firing a transition. A cloud provider charges the users based on analysis of the convertibility graph. The method of analysis depends on counting the number of activities occurring during the renting session.

On the other hand, Deterministic Stochastic Petri-nets (DSPN) is proposed in [68]. Chen and Vandenberg modelled a reconfigurable protocol stack for a control system in the cloud. In the suggested model, DSPNs are used for analysis of the performance for an Ethernet network in the cloud when various configurations are applied. In the model, the places are mapped as network entities while protocol behaviour is modelled as transitions. Our work is different from the work in [87, 68]. We used CPN models for analysing which policy is suitable in terms of energy and migration costs as covered in Chapter 4, Chapter 6 and Chapter 7. Thus, our work is based on analysis and assesses policies for migration behaviour before implementation.

2.12 Chapter Summary

In this chapter, we provided a background for technological methods and theoretical aspects used to address the three challenges solved in this thesis. Furthermore, we also covered reviews about the existing related works, which are essential for Chapters 3, 4, 5, 6 and 7. The reviews mainly related to automatic management architectures in large-scale systems, the specification of rules-modelling languages, the existing domain-specific languages in cloud platforms, some of the existing cost-analysis methods used in Petri-nets models, and the current Petri-net models for cloud platforms.

CHAPTER 3

MP-Framework for Automatically Managed Cloud Platform

Management policies, which are expressed by cloud managers or non-technical users, are high-level. The continuous change in policies is based on the nature of the technical environment and changes in regulations and business requirements that the management policies might be applied to. As a result, as stated in the introduction, there is a gap between the level of describing management policies and executing them in an automatic manner. Although the previous research in developing autonomic architecture, as mentioned in Section 2.7 in Chapter 2, can be applied to automatically executing the expressed management policies, the software development and maintainability for such architectures are costly and time-consuming. This is due to the frequent modification of such policies. Therefore, another solution to bridge that gap is to design a generic architectural framework for executing management policies which is easily structured and uses enhanced technologies in its implementation. Hence, in this chapter, we explain the design of our proposed *MP-Framework* in detail. The *MP-Framework* is implemented to become an automatic controller that can be easily plugged in and executed with any cloud management system such as OpenNebula.

3.1 The Description of Management Policy Framework (MP-Framework)

Management Policy Framework (MP-Framework) is a generic framework for automatically triggering a management action into a cloud platform by executing various sets of management policies. These management policies are formulated as rule-sets which are enforced by a rule-engine that implements the Rete algorithm [49]. Examples of possible management actions would be triggering a live-migration action for the over-loaded virtual machine; notifying a cloud manager of the current status of energy consumption, or requesting to perform a management strategy to keep the amount of energy consumed by the nodes of the cloud platform within specified boundaries. Those management examples are extracted from our Management Energy Consumption Case Study that will be explained in Section 3.4 in this chapter.

The design of *MP-Framework* is considered to be generic, fine-grained, and easy to configure. Conceptually, the architecture of the framework is inspired by the design of self-managing systems that utilise control theory [135] combined with a rule-based system. These types of self-managing systems have essential components, namely a controller, a sensor, and an actuator or effector (for more information about using control theory in a self-managing system, see Section 2.7 in Chapter 2). Moreover, using a rule-based system in designing *MP-Framework* led to construct the framework as a fine-grained independent component. This fact becomes clear in the following section when we explain the components of the framework in depth. Therefore, producing such features allows the framework to be easily integrated with interfaces of any cloud management system such as OpenNebula [105]. Thus, *MP-Framework* can be help to solve an energy management

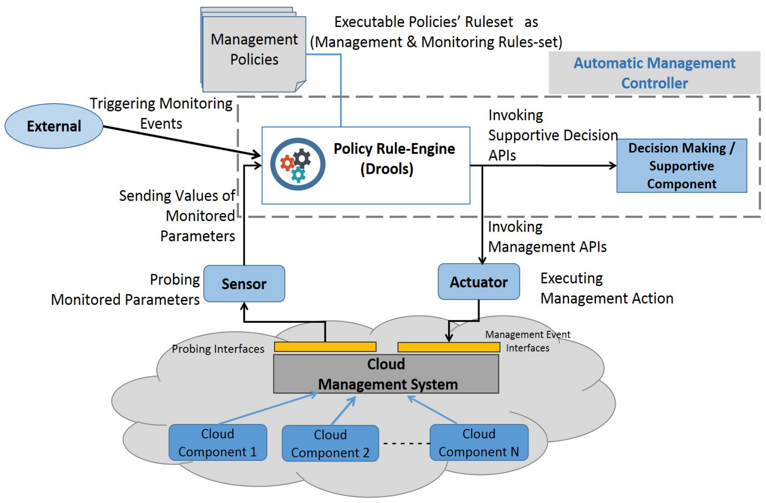

Management Policies
Executable Policies' Ruleset as (Management & Monitoring Rules-set)
Automatic Management Controller
External
Triggering Monitoring Events
Policy Rule-Engine (Drools)
Invoking Supportive Decision APIs
Decision Making / Supportive Component
Sending Values of Monitored Parameters
Invoking Management APIs
Probing Monitored Parameters
Sensor
Actuator
Executing Management Action
Probing Interfaces
Management Event Interfaces
Cloud Management System
Cloud Component 1
Cloud Component 2
Cloud Component N

Figure 3.1 – The architectural design of *MP-Framework* for executing Management Policies

problem in a cloud platform automatically. For example, *MP-Framework* can be used to manage energy consumption via migrating virtual machines, which is the case study covered in this thesis.

3.1.1 The Architectural Design of *MP-Framework*

Figure 3.1 illustrates the architectural design of *MP-Framework* which is used directly to launch management actions by triggering the satisfied management policy. The architecture consists of a policy rule-engine, sensors, actuators, and a decision-making or supportive component. These components should be interacting with a cloud manager through probing and management event interfaces. The policy rule-engine is the controller of the *MP-Framework* which is responsible for triggering management actions after analysing the monitored parameters provided through the sensors of the framework. The design of the policy rule-engine and its functionality are explained in more detail in Section 2.2 in Chapter 2. The management action selected by the policy rule-engine is executed by a Cloud Manager that is the main part of the cloud management system. As shown in Figure 3.1, the Cloud Manager interacts with a number of cloud components and has its own probing components for collecting the monitored parameters, such as resource usage, SLA-violation rate, and energy consumption from cloud nodes and running virtual machines at each cloud node. In addition, the Cloud Manager has its own event system that is also responsible for triggering the action which is received from the policy rule-engine (see Figure 3.1).

As presented in Figure 3.1, the communication between the policy rule-engine and the Cloud Manager is accomplished by interacting with two different components, which are a sensor and an actuator. Directly, the sensor is interlinked with the APIs of the cloud manager through probing interfaces. The sensor of the framework is responsible for requesting monitored parameters that are used for applying management policy periodically. In turn, the actuator has different types of interfaces which are called management-event interfaces. The management-event interfaces include set management action APIs of the Cloud Manager. The management actions APIs should directly execute the required management action, which is received as a message from the policy rule-engine.

As seen in Figure 3.1, the *MP-Framework* has also an extra component which is referred to as a decision-making or a supportive component. This additional component employs the policy rule-engine with an enhanced functionality for making decisions. Optionally, the supportive element can be configured to run search-based algorithms to provide the policy rule-engine with allocation solutions. For example, the supportive component can be set with the first-fit algorithm [64] to provide a re-allocation scheme for the migrated virtual machine which is configured in the case study explained in Section 3.4. Eventually, it is noticeable that *MP-Framework* has a centralised architecture due to the usage of a policy rule-engine that implements the Rete algorithm for object-oriented patterns. At the current stage, the implementation of a rule-based system such as Drools [70] has a centralised structure. This structure allows only one interaction with the Cloud Manager to take place at a time. However, enhancing *MP-Framework* to support a triggering action in a decentralised architecture will be discussed as a future work in Chapter 8. To make

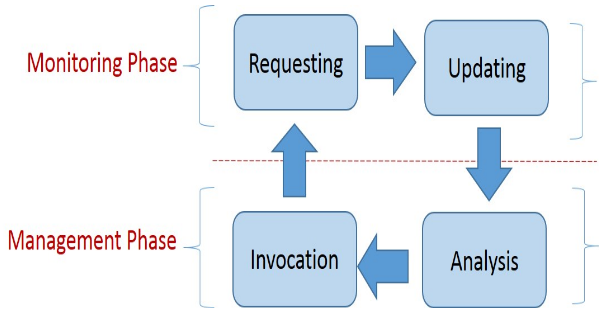


Figure 3.2 – The phases and the run-time execution states of *MP-Framework*

the *MP-Framework* perform automatic management, it should be executed in phases and across a set of states. These states are explained in the following section.

3.2 The Phases and the States of *MP-Framework*

The *MP-Framework* is executed in four cyclic states: Requesting, Updating, Analysis, and Invocation, which are presented in Figure 3.2. These specified states occur in two phases, which are the **Monitoring Phase** and the **Management Phase**. The execution of *MP-Framework* starts with the **Monitoring Phase**. During the **Monitoring Phase**, the policy rule-engine sends a request message to the sensor to provide the rule-engine with appropriate monitored parameters. During this period, *MP-Framework* is considered to be in the Requesting Phase. During the Requesting State, *MP-Framework* uses different types of rule-sets which are referred to as monitoring rules (see Subsection 4.1.1 in Chapter 4 for the definition of the monitoring rules and a sample of this scheme). The sensor of the framework sends a message to a Cloud Manager requesting the recent values of the monitored parameters such as resource usage and energy consumption. When the Cloud Manager receives the message, it uses its own probes to collect the required data and send them back to the sensor.

When the recent values of the required parameters are received, the state of *MP-Framework* changes to the Updating State. During the Updating Phase, no rule-sets are used. The sensor loads the recent values as a fact into the policy rule-engine. The rule-engine uses other types of rule-sets which are called the management rule-set or management policy (see Subsection 4.1.2 in Chapter 4 for a definition of these rules and a sample of this scheme). At the end of the Updating State, the framework will be in the Management Phase.

The **Management Phase** begins with the Analysis State. During the Analysis State, the rule-engine uses both the management rule-set and the recent updated values of the facts to perform the analysis process. The objective of the analysis process is to determine the status of the running cloud component and to find the stored management policy that should be activated. At the end of the Analysis Phase, the selected policy is loaded into the agenda for activation. If there exists an activated policy, the framework enters the Invocation Phase. Otherwise, the Invocation State is skipped, and the Requesting State begins again.

During the Invocation State, the agenda begins executing the action part of the activated policy. At the end of this state, the actuator should send action messages to the Cloud Manager. The end of the Invocation State is an indication to end the Management Phase. As a result, the Monitoring Phase starts again after a specified waiting time-window which is provided during the initialisation state of *MP-Framework*.

3.3 The Implementation of MP-Framework Using OpenNebula and Drools

The implementation of the architecture was done in two stages. The first stage involved building a cloud platform and configuring OpenNebula [105]. The designed cloud platform was built to include three physical nodes: a combination of two laptops and one desktop. The laptops were a Samsung and a Sony VAIO, both of which had similar sys-

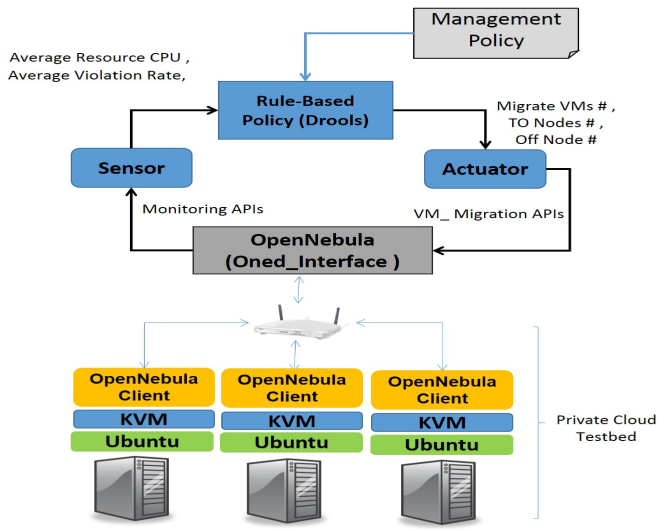


Figure 3.3 – The implementation of *MP-Framework* using OpenNebula [105] and Drools [70]

tem properties including a 2.4 GHz Intel(R)Core(i5) processor and 6 GB Memory. The system properties of the desktop were a 2.2 GHz Intel Core 2 Duo processor(vPro) and 4 GB Memory. All the three physical nodes ran Ubuntu version 11.0 and used KVM as a virtualisation technology. All three nodes are connected using a TP-LINK TL-SG1005D 5-port Gigabit Switch.

To complete the implementation of a private cloud platform, we configured the OpenNebula management system. This was achieved by deploying and running the OpenNebula client on each physical node. The OpenNebula Client controls running virtual machines by interacting with the deployed KVM manager. Due to the limited resources, we deployed the ONED daemon as the cloud manager on the Sony laptop. ONED remotely interacts with cloud nodes via Remote Procedure Call (RPC) interfaces. By using this configuration, our cloud platform testbed is considered to be a private cloud. The purpose of this cloud platform is to provide an infrastructure-as-a-service by deploying virtual machines used for processing computational jobs for the end-users who will possess these images during the validation of their service-level agreements (SLAs).

The *MP-Framework* was developed using Drools technology [20, 70]. In Drools, the domain data model and the implementation of the engine are based on an object-oriented framework; as a result, there is a simplicity in representing the formulated rules and data using Java classes. Furthermore, Drools can easily interact with the OpenNebula manager. Since OpenNebula uses RPC calls, we needed only to write simple wrappers to parse the sent/received RPC to/from Drools.

We developed the supportive component using Drools Planner (see Chapter 8 in [20] or Chapter 7 in [70]). Drools Planner is a library which is used to solve planning problems [20]. Since the objective with our framework is to migrate the overloaded virtual machines to other physical nodes, the planner assists in providing a reallocation scheme for the migrated virtual machine to the available running hosts. We configured the planner to use first-fit decreasing [64]. The planner is used if the management policy contains a statement invoking the supportive component, which is similar to the scheme captured in Section 4.1 in Chapter 4. Otherwise, if the supportive component is not used, the reallocation is done based on a scheduling algorithm configured in the ONED daemon on the OpenNebula side.

The final stage of the implementation is to develop both the sensor and the actuator. The sensor of the framework is developed as a software component which launches periodic requests. In our research scope, this component requests an update for values of three monitored parameters: resource usage, energy consumption, and migration violation rate. However, other types of monitored parameters can be added and configured by extending the monitoring APIs provided by the cloud management system. In this research, the implemented sensor uses two types of OpenNebula monitoring APIs which are built-in and extended monitoring OpenNebula APIs. The built-in monitoring APIs are configured to collect the values of resource consumption, whilst the values of energy consumption and migration SLA-violation rate (which are required in the case study and will be explained in Section 3.4) are gathered using the extended monitoring OpenNebula APIs. The following subsection will provide a detailed description of the implementation of the extended monitoring APIs.

The implemented sensor also uses an XML parser to parse the retrieved data from the RPC message and send data in XML-format to be loaded into the rule-engine. The operation of the sensor component is controlled by Drools, which uses a rule-scheme similar to the scheme presented in Section 4.1 in Chapter 4.

On the other hand, the actuator is implemented as a software component which also sends the activated management action as a request. The actuator deals with management action APIs, which are triggered when management messages are launched from the Drools engine to be received by the OpenNebula manager. Furthermore, the actuator also uses a parser to parse the management messages received from Drools and encapsulates the messages in RPC format before directly sending them to the OpenNebula manager. The actuator should trigger the OpenNebula migration APIs.

3.3.1 The Extension of OpenNebula Monitoring APIs

The case study used in this research depends on the values of energy consumption and migration SLA-violation rate which cannot be provided directly via built-in OpenNebula monitoring APIs. Therefore, we extended the OpenNebula monitoring component to include two extra monitoring features that would be suitable for the case study explained in Section 3.4.

Implementing Migration SLA-Violation Rate Monitoring APl

The Case Study expressed in Section 3.4 allows the running virtual machine to migrate among the cloud nodes of the implemented testbed. Since the migration of a virtual machine might cause a delay in delivering services [35], SLA-violation might occur. As a result, we extended the OpenNebula monitor to compute the violation rate associated with each migrated virtual machine. The computation of the violation is based on implementing the following model, which is applied to each running virtual machine at each running host in the cloud platform.

For each VM_i running at each $Host\_j$:

$$ViolationRate_{Migration} = \sum MigrationTime \times Price(SLA\_Violation_{min}) \tag{1}$$

The migration time represents the time for migrating a virtual machine from the source host to the destination host. The value of migration time is computed using the following equation:

$$MigrationTime = \frac{VM_{image\_size}}{Bandwidth_{available}} + VMTime_{Startup} \tag{2}$$

The first part of Equation 2 is extracted from the violation model proposed in [35]. Here, $VMTime_{Startup}$ represents the time required for resuming the operation of the migrated virtual machine on the destination host. This value is computed when the acknowledgement message for the arrival migrated VM to the destination is received.

Returning to Equation 1, the parameter $Price(SLA\_Violation_{min})$ represents the value of the minimum penalty price that the cloud provider would pay in the event of SLA-violation. We assigned the automatic triggering live-migration action to result in a violation in SLA due to the time for migration. This will have an impact on meeting the requirements of delivering services to the cloud consumer. As a result, we assigned that each migration of a virtual machine should use the minimum penalty price in order to prevent the occurrence of multiple migrations of the same virtual machine.

Implementing Energy Consumption Monitoring API

In the case study, it was necessary to obtain the value of energy consumption. To do so, we needed to have a hardware device plugged in to each node in the cloud platform to measure energy consumption or use the energy measurement tool suggested in [102]. However, due to the lack of resources provided and our objective to test the applicability of the framework in a real cloud platform, we built an energy consumption monitor that estimates the amount of energy consumption by a server running in the testbed using benchmark data. The implemented energy monitor is called Estimator Energy API.

The Estimator_Energy API is based on using the benchmark results published by Standard Performance Evaluation Corporation (SPEC) [13]. SPEC is a non-profit organisation that produces benchmark tools and allows hardware manufacturers to test and publish their system performance [13]. We selected the published data for SPECpower_ssj2008, which are the results of Third Quarter 2012. We examined the hardware specification closest to our platform configuration, namely the results published by HP data for multi-node servers. The specification of HP data mimics the system properties of the nodes used in our testbed (more information about SPECpower_ssj2008 results can be found in [13]). We designed a module that requests during each periodic time the workload of each running virtual machine. Based on the current status of the workload value on each node and using the benchmark results, we retrieved the average active power measured in watts that matches the workload. Then, we assigned a random value to the retrieved average active power. The obtained value would become an estimation of energy consumption for a running node in the testbed which would be sent to the sensor of *MP-Framework*. This extended energy consumption monitored API can be replaced with the suggested energy management method suggested in [102] which depends on computing the amount of energy consumption of each running process in a virtual machine based on the workload of a running application.

3.4 The Application of *MP-Framework* in a Case Study

The management of energy consumption in cloud platforms using dynamic and adaptive techniques has received considerable attention in research as in [24, 35, 95, 133]. Since the objective of designing *MP-Framework* is to automate the triggering of any management action, we applied the framework to demonstrate energy consumption management cases that can be found in private cloud platforms. Our objective from having conducted various experiments which include testing different set of Management Policies is a proof of concept that the framework can be applied to govern both energy consumption and the SLA-Violation rate in a real cloud platform. This will be explained in more detail in the following sections. Nevertheless, a description of the case study in our research will be presented in the following section.

3.4.1 The Description of the Energy Management Case Study

A cloud platform consists of hosting nodes and services. The services are VM images belonging to cloud consumers of that platform. For that platform a number of properties are required to be managed automatically. One of these properties is for the daily management of the energy consumption at each hosting node to be within certain values with a condition that the overall amount of energy consumed by the OpenNebula testbed is less than 5000 *watts* during the off-peak time. This objective can be accomplished, for example, by specifying a management action which triggers "the live-migration for running VMs". The live-migration action is launched automatically when the energy consumption value at a running host rises above the normal level which is between 100 *watts* and 135 *watts*. In addition, there is also another property that should be considered, which is the SLA-violation rate of the migrated virtual machine. This rate should also be kept at a minimum, not exceeding an overall £7.0 for the whole platform during the off-peak time.

Assumptions: We defined a number of assumptions for simplifying the implementation of the Energy Consumption Management Scenario. Firstly, we assumed that the measurement of resource consumption for each physical host and running virtual machine would be based on the average percentage of both CPU usage and memory usage. Secondly, the cloud platform uses the same virtualisation software. The objective is to reduce the complexity of the virtual machine migration operation and to avoid software incompatibility issues. Therefore, the cloud platform used in the case study has a homogeneous infrastructure. The final assumption is that there are no rules to restrict the migration operation of virtual machines. This is to simplify our explanation and does not affect the generality of our method.

The Required Monitored Parameters

Various monitored metrics are used as constraints for specifying the used management policies. The defined monitored parameters which can be represented as an extension of the fixed utilisation thresholds proposed in [35], are configured in the management policies subjectively. In other words, the threshold values for monitored parameters will be assigned by the cloud consumer during the design stage. Thus, the definition of each monitored parameter is as follows:

1. **Resource Consumption** is the value of CPU usage and memory usage for each single host in the cloud platform.
2. **Energy Consumption** is the amount of power consumption at each running server in the cloud platform.
3. **Migration SLA-Violation Rate** is the degree of violation rate which is associated with the VM as a result of triggering a migration action.

Suggested Policy	**Conditions For Each Running Host**	**Management Actions**
Management Policy A	Energy-Consumption $\leqslant 100$ *watts* or Energy-Consumption $\geqslant 135$ *watts* CPU-Usage $\leqslant 20\%$ or CPU-Usage $\geqslant 50\%$	**Migrate** a small-size Virtual Machine to Normal-loaded Hosts **Switch-off** Lower-loaded that do not run any Virtual Machines
Management Policy B	Energy Consumption $\leqslant 100$ *watts* **or** Energy-Consumption $\geqslant 135$ *watts* CPU_Usage $\leqslant 20\%$ **or** CPU_Usage $\geqslant 70\%$	**Migrate** if possible a small-size Virtual Machine to any available host
Management Policy C	Energy Consumption $\leqslant 100$ *watts* **or** Energy-Consumption $\geqslant 135$ *watts* CPU_Usage $\geqslant 50\%$	**Migrate** a small-size virtual machine running on Lower-loaded hosts to Normal-loaded hosts
Management Policy D	Energy Consumption $\leqslant 100$ *watts* **or** Energy-Consumption $\geqslant 135$ *watts* CPU_Usage $\leqslant 20\%$ **or** CPU_Usage $\geqslant 50\%$	**Migrate** a small-size virtual machine running to any available hosts for CPU_Usage more than 50% and Change VM_image configuration to use lower configuration for CPU_Usage less than 20%

Table 3.1 – The Suggested Management Policies Executed in OpenNebula and Drools Testbed and Demonstrating an Automatic Management in Energy Management Case Study

3.4.2 The Suggested Management Policies

In order to manage energy consumption, a set of management policies should be run during the off-peak time. We extracted four different management rules from [26] and [92] and modified these rules to be appropriate for execution in *MP-Framework* and to match the requirements of the previously mentioned case study. Table 3.1 presents these management policies. In Table 3.1, we have stated four types of policies. These policies are Management Policy A, Management Policy B, Management Policy C and Management Policy D. Each of these policies should trigger a different set of management actions. These actions are Migration virtual machines, switching off idle hosts or reconfiguration of services to use lower resources. The management actions are triggered as a response to the continuous measurement of the amount of Energy Consumption and the level of CPU_Usage from the running cloud platform. Triggering management actions is based

	CPU	Memory	Image_Type
Group A	2	1GB	Ubuntu_10.04Desktop.img
Group B	2	500MB	Ubuntu_10.04Desktop.img

Table 3.2 – Different Virtual Machine Images Used in OpenNebula and Drools Testbed

on specifying which running hosts in cloud platform have either above or lower levels of Energy Consumption within the allowed threshold values. In addition, the firing management action also depends on keeping the amount of Energy Consumption and the level of CPU_Usage within defined normal thresholds. The defined normal thresholds for Energy Consumption should be between 100 *watts* and 135 *watts*. Whilst the normal level of CPU_Usage varies as shown in Table 3.1. Each management policies defined in Table 3.1 is used to test each running physical hosing node in our cloud platform. For example, Management Policy A states that *If any running host in cloud-platform has an amount of Energy Consumption either at most more than* 135 *watts or less than* 100 *watts and that host is either overloaded or lower-loaded, then migrate any small size virtual machine for that host to any available normal loaded host. Furthermore, switch off if possible the lower-loaded hosts that have no running virtual machines.*

3.4.3 Executing the Management Policies in OpenNebula and Drools Testbed

Before executing each management policy suggested in Table 3.1, we configured the platform to run a fixed number of virtual machines during the execution time for running the management policies. The configured virtual machines belong to two different sets of VM images, namely Group A and Group B virtual images. The overall number of running virtual machines was 21. Table 3.2 presents the specification of the groups. Furthermore, the total number of rules assigned to each management policy was 6 rules and the total number of facts inside each management policy was 33 objects. Each group of VM images is configured to run stress [7, 57] during its execution time. Stress is a workload generator tool for LINUX systems used to randomly change CPU, memory, I/O and disk properties [7, 57]. The Stress tool generates the workloads by creating a number of worker processes and terminating them randomly [7, 57]. As a result, the value of CPU usage for each running virtual machine would vary when using the stress workload generator every ten minutes of execution. We configured the stress load generator to assign a random workload pattern to each running virtual machine during the execution of each management policy. Furthermore, the generated workload pattern used during the execution of Management Policy A is captured and used in the remaining management policies. Using a similar workload pattern allows us to identify which executed management policy is better in terms of having less energy consumption and a lower migration SLA-violation rate.

During the execution of policies explained in Table 3.1, we aimed to ensure that all the policies were running successfully and correctly in the OpenNebula and Drools testbed for nine hours with nine repetitions. We conducted the execution for nine hours since each management policy used in the case study should be applied during the off-peak time. As a result, we intended to closely simulate the situation in real scenarios. During the execution, we examined whether *MP-Framework* has the ability to achieve this energy management objective: *[The overall Energy Consumption by OpenNebula and Drools Testbed should be less than* 5000 *watts after nine-hours]*. Moreover, during the execution, we also observed the number of triggered management actions, whether VM live-migration or reconfiguration actions, performed correctly by the OpenNebula and Drools Testbed. Repeating each conducted experiment up to 100 times can provide concise results. However, we tested our framework in a real cloud environment which was a stable platform for nine-hours without stopping. Therefore, we repeated the execution for each policy only nine times. We would expect that the results would not make an unacceptable change when using the same configuration and using a similar number of Management Policies. However, if the configurations are used for a large scale platform with introduction to another set of policies, the results might change. This was beyond the scope of the conducted experiment since our aim is a proof of concept for having a framework that manages a platform using policies translated as rules. The results of the execution are discussed in the following section.

3.4.4 Results and Discussion

Figures 3.4 and 3.5 present the overall average of the amount of energy consumption and the cost of SLA-violation rate caused by triggering a live-migration action. We analysed both figures to identify the executed management policy that made the OpenNebula and Drools testbed successfully apply automatic management with the least energy consumption during nine hours. Since the experiment was repeated nine times, the difference in the obtained results from each execution was relativity small:5%-7.5%. Therefore, we obtained the overall average of the all nine executions and present them in Figures 3.4 and 3.5. The following subsections will offer a detailed discussion of the results obtained. In the figures, we focused on justifying why there was a difference in the amount of energy consumption obtained from each executed policy. In addition, we also traced the cost of SLA-Violation assoicated with each time the migration of virtual machine occurred in the testbed.

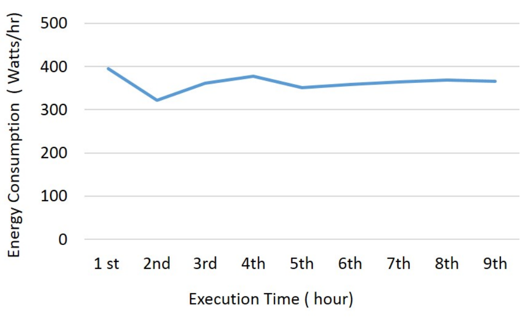
Analysis the Results in Terms of Energy Consumption

Figure 3.4 presents the overall amount of energy consumption from running OpenNebula and Drools testbed during the execution of each management policy mentioned in Table 3.1. Those figures are obtained after repeating nine times. As seen in Sub-Figure 3.4(a) the overall amount of energy consumed while executing Management Policy A is between 300 watts and 400 watts. In particular, if we focused on the period from the fifth hour of executing this policy until the end of the execution time, the overall energy consumption remained steady at approximately 300 watts. As a result, when applying a set of management actions such as the actions used in Management Policy A (see Table 3.1) as well as having a platform receiving random workloads as explained in Subsection 3.4.3, the amount of energy consumption remained within limited boundaries as presented in Sub-Figure 3.4(a).

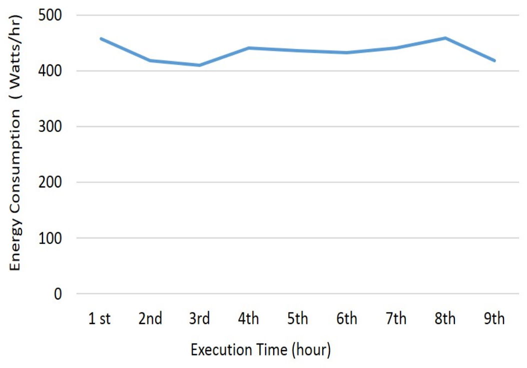
On the other hand, as shown in Sub-Figure 3.4(b) the overall average of energy consumption at the running testbed during the execution of Management Policy B was 420 watts - 460 watts approximately. The amount of energy fluctuated considerably during the nine hours. We noticed that a platform applying Management Policy B consumed more energy than the one that implements Management Policy A. To justify that, Management Policy A allows the migration within thresholds which are between 50% and 20% and also permits switching off idle hosts as a second action. These threshold values are considerably less than the ones specified in Management Policy B.

Looking again to Sub-Figure 3.4(c), the overall average energy consumption in the running testbed was stable throughout most of the hours of the execution. The stable values were roughly between 407 watts and 410 watts between the second and the fifth hour. However, the amount decreased to less than the values recorded in Sub-Figure 3.4(c) during the execution of similar hours. Management Policy C recorded lower energy consumption values than Management Policy B because of the usage of a single threshold. In contrast, Management Policy B allowed the migration action to be triggered when CPU usage for each running host was either above or below the upper and the lower specified values (see Table 3.1).

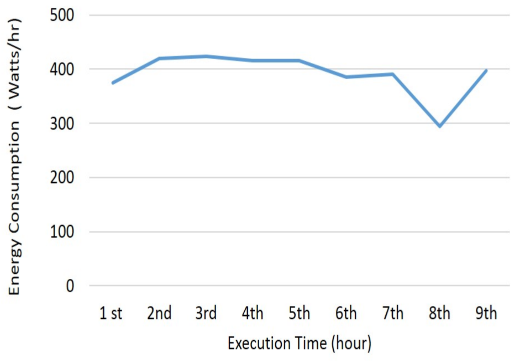
As shown in Table 3.1 Management Policy D has two types of management actions, namely those triggering live-migration and resource reconfiguration actions. Triggering two different management actions was reflected in the values of energy consumption, which are shown in Sub-Figure 3.4(d). In Sub-Figure 3.4(d), the overall amount of energy consumption began with 390 watts before the values increased. It is noticeable that Management Policy D consumed slightly more energy than Management Policy C during the execution of the second and the third hours. However, between the fifth hour and the ninth hour of executing Management Policy D, the values of energy consumption increased at a steady rate of about 400 watts.



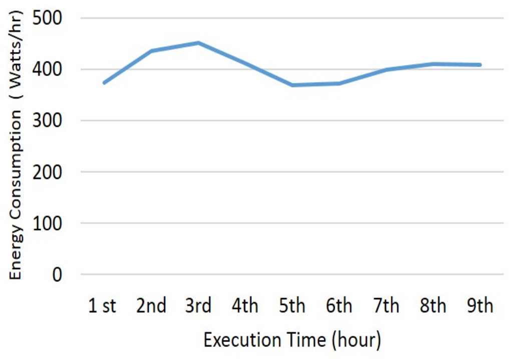
(a) The values of Energy Consumption of Management Policy A



(b) The values of Energy Consumption of Management Policy B



(c) The values of Energy Consumption of Management Policy C



(d) The values of Energy Consumption of Management Policy D

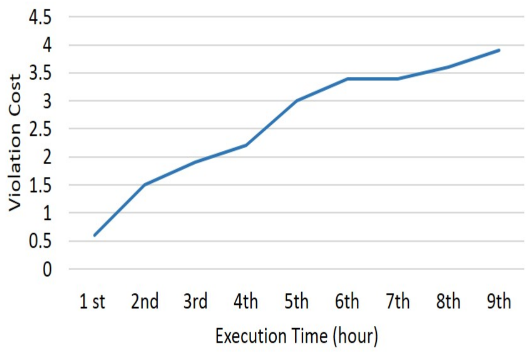
Figure 3.4 – The overall average amount of Energy Consumption recorded in OpenNebula and Drools Testbed during the execution of Management Policies mentioned in Table 3.1 for nine hours with nine repetitions

Analysis the Results in Terms of the Cost of SLA-Violation

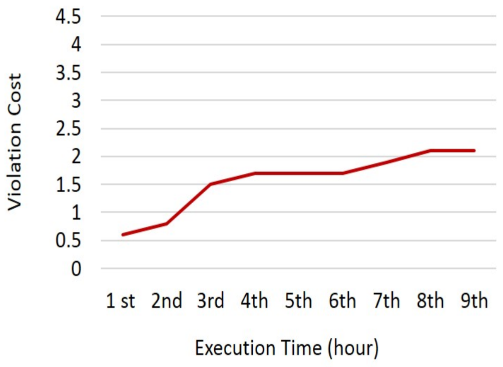
Executing management policies not only has an effect on the amount of energy consumption but there is also a cost property associated with the frequent triggering of live-migration actions. This cost property is the cost of SLA-violation caused by triggering live-migration action which it is important to investigate in our research. There is another cost property, triggering a reconfiguration action, which was ignored in our research because it was applied only in Management Policy D. In addition, the reconfiguration management action was triggered only when the CPU-load was low. Therefore, Figure 3.5 depicts the cost of SLA-violation resulting from the frequent triggering of live-migration actions during the execution of management policies mentioned in Subsection 3.4.2.

As shown in Sub-Figure 3.5(a) the cost of SLA-violation in a platform that executes Management Policy A caused by firing live-migration actions increased because that migration action occurred no more than twice in an hour. On the other hand, we noticed that the SLA-violation rate caused by triggering a migration action in a platform executing Management Policy B, which is presented in Sub-Figure 3.5(b), was lower than the SLA-violation values of Management Policy A. Due to the increase in value of the upper-bound of the specified threshold used in Management Policy B (see Table 3.1), the number of live-migration actions in each hour was considerably lower than similar triggered actions in Management Policy A. Thus, Management Policy B has a lower SLA-violation rate than Management Policy A (for more details about the number of triggered migration actions, see Sub-Figure 3.6(c)).

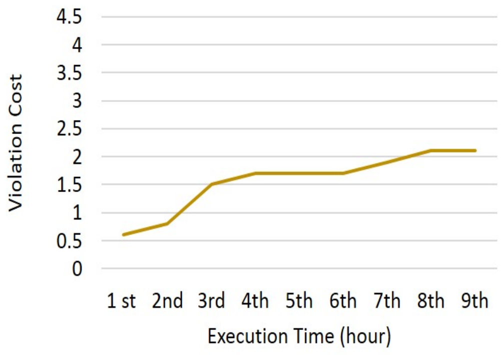
Similar to the reason for controlling the cost of SLA-violation values in Management Policy B, the cost of SLA-violation rate recorded during the execution of Management Policy C considerably increased at a rate close to the rate of Management Policy B. These values were relatively lower than the values achieved by Management Policy A. Similar to the figures presented obtained from Management Policy B and Management Policy C, the values of the cost of SLA-violation for also Management Policy D were lower than the cost of SLA-violation for Management Policy A (See Sub-Figures 3.5(c) and 3.5(d)). This is due to the lower frequency of firing migration actions during the execution of Management Policy C, which occurred only when the CPU usage was above 50%.



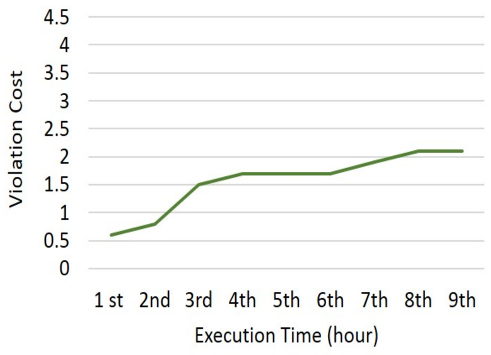
(a) The SLA-Violation Cost of Management Policy A



(b) The SLA-Violation Cost of Management Policy B



(c) The SLA-Violation Cost of Management Policy C



(d) The SLA-Violation Cost of Management Policy D

Figure 3.5 – The average cost of SLA-violation caused by triggering migration-action during the execution of management policies mentioned in Table 3.1 in OpenNebula and Drools testbed for nine hours with nine repetitions

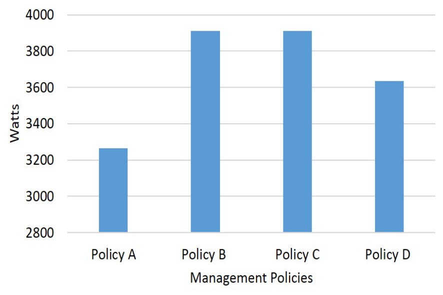
The Overall Analysis for the Results Obtained and The Response Time

Figure 3.6 shows the averages of the total amount of energy consumed, the total cost of SLA-violation, and the number of successful occurrences of management action for each of the demonstrated management policy after executing them for nine hours using loads generated from Stress tool [7]. From Sub-Figure 3.6(a), it is clear that Management Policy A had the lowest amount of energy consumption, whilst both Management Policy B and Management Policy C had the highest amount of energy consumption. Management Policy D had an energy consumption value which was less than the values of Management Policy B and Management Policy C. However, this value was not less than the amount of energy consumption of Management Policy A. These various energy consumption values resulted from the use of different threshold values and triggering various management actions.

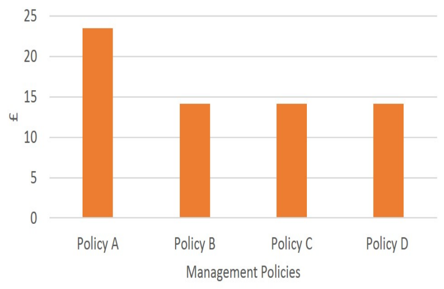
Moreover, we computed the overall total costs of SLA-violation rates, which are shown in Sub-Figure 3.6(b). As presented in this figure, the cost of the SLA-violation rate of Management Policy A was the highest amongst the other management policies, which all had similar SLA-violation costs. This result occurred because Management Policy A had the highest number of migration actions with 13 (see Sub-Figure 3.6(c)). In contrast, Management Policy D recorded the lowest number of live-migration actions with only 5 during its execution time. Management Policy B and Management Policy C had similar live-migration actions with 9 times and 10 times, respectively.

3.4.5 The Response Time for the Policy-Rule Engine

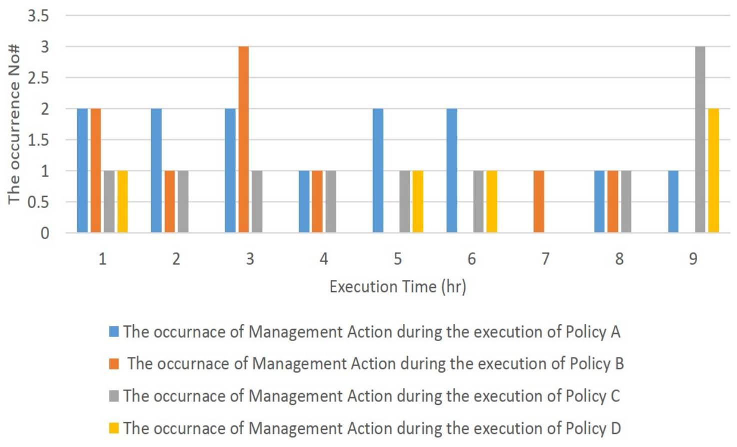
The response times for the Drools rule engine and the Drools Planner, the main components of the MP-Framework, were measured during the execution for 9 hours for each running management policy. Since the rule engine has about three rules for each running management policy, the pattern-matching time was measured, along with the time of ex-



(a) The total amount of Energy Consumption



(b) The total cost of SLA-Violation



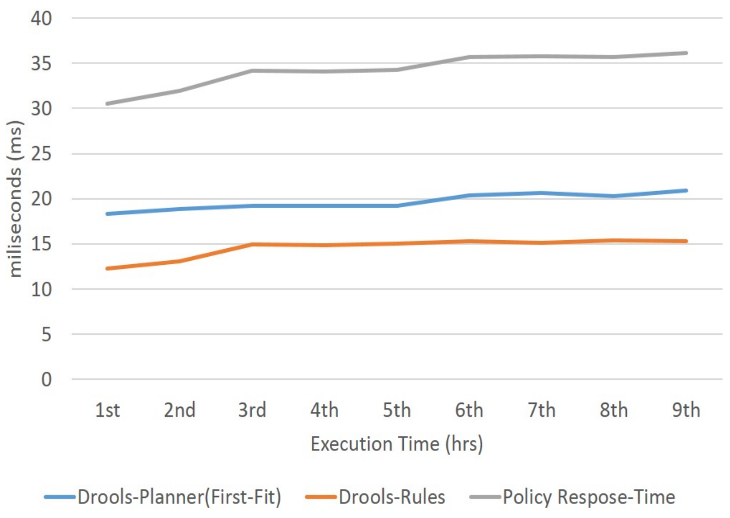
(c) The occurrence number of live-migration action for all executed Management Policies

Figure 3.6 – The total amount of energy consumption, the total cost of SLA-violation, and the number of occurrences of management actions for each management policy executed in OpenNebula and Drools Testbed after nine hours

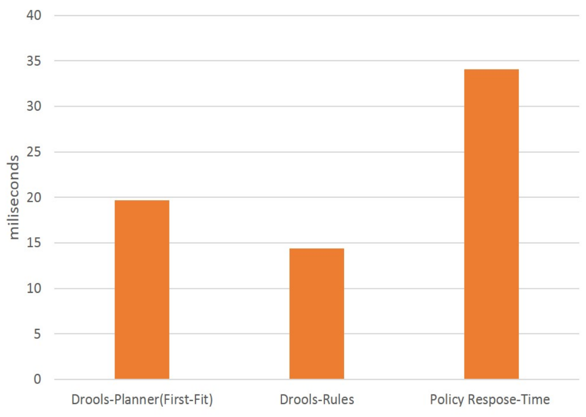
ecuting the first-fit algorithm implemented using Drools Solver. The detailed results for the policy response time are presented in Figure 3.7. Figure 3.7(a) shows that over 9 hours while the framework received similar inputs applied during experimental executions of all management policies, the average for the overall time response of the policy rule engines was between 30 and 37 milliseconds (ms). During the execution, the time increased in relatively small patterns . The overall policy response time results from the accumulation of the average response time records for executing Drools Planner and the Drools engine over 9 hours.

As shown in Figure 3.7(a) the average response time for Drools Planner implementing the First-Fit algorithm was between 19 ms and 21 ms. The values increased steadily because of the increased number of virtual machines that should be migrated, which needed to be allocated. In contrast, the response time of the rule engine was between 13 ms and 15 ms due to the fixed number of rules used for each running management policy. Overall, the results for the response time of the policy-rule engine are presented in Figure 3.7(b). At the end of the executions, the average response times of Drools Planner, the rule engine and the policy-framework were approximately 20 ms, 14 ms and 34 ms, respectively.

To summarise, based on the results obtained, *MP-Framework* is able to perform successfully the automatic management of energy consumption. This was shown through demonstrating various types of management policies (See Table 3.1). It was noticed that the overall amount of energy consumption for each management policy was less than 5000 watts over nine hours. However, the amount of energy consumption could be changed when various types of workloads are used. The average response time for the policy rule engines for all tested management policies was 34 ms. Moreover, we noticed that the migrating time for successfully moving a virtual machine between nodes is between 5 minutes and 7 minutes approximately. The delays for finishing the management action



(a) The Detailed Results Over 9 Hours



(b) The Average Response Time for The Policy-Rule Engine

Figure 3.7 – The results of the response time for the policy rule engine including (Drools-Engine and Drools Planners) for all running management policies for nine-hours

were not because of the design of *MP-Framework*; rather, they depended on other factors related to the implementation of *MP-Framework* such as the network speed, the size of the virtual machine, and the cloud management system used. As a result, a virtual machine that runs critical jobs or has a higher migrating rate should be restricted from multiple migration actions. Therefore, it is preferable to define a set of logical constraints for the running virtual machines which also can be specified in SLA before executing them and running them in *MP-Framework*. Allowing this feature can contribute to reducing the SLA-violation rate.

3.5 A Brief Discussion

In [126], the performance of Drools when deployed into one server running 1000 rules was nearly 1.4 seconds. This value increases to 31.2 seconds when the number of rules is 100,000. OpenNebula, the cloud management system used in our research, can manage up to 500 servers [105]. The response time of the *MP-Framework* could be affected by the number of the servers used, the deployed virtual machines and the time for triggering management actions. As mentioned in the previous section, the migration time of the virtual machine in our testbed was 5 to 7 minutes. If we compare that time in the data centre with highly efficient servers, as well as network communications, the migration time might be reduced.

In this chapter, we showed that the framework governs a small-scale cloud platform. However, for medium-sized cloud platforms, we suggest making an alteration to the method of deployment of the *MP-Framework*. This would involve duplicating the framework to be deployed to multiple cloud managers and separating cloud managers into clusters for increasing the scalability of the framework. Then, a communication channel should be opened between the deployed automatic controllers. All automatic controllers can execute either similar or different management policies based on the requirements of the data centre. Furthermore, the deployed automatic controllers need to have global knowledge about the overall data centre environment. Such an architecture deployment model opens new challenges in transforming the *MP-Framework* to apply to semi-decentralised or fully decentralised data centres. These challenges are beyond the scope of our research, and one researcher in our team has started investigating issues of duplicating the policy-rule engines. The focus in this thesis was on bridging the gap between the level of describing management policies related to control energy and implementation of the policies. In this section, we pointed out issues which might be found in large-scale data centres to show that we are aware that possible modification might be required for integration of the framework to make it executable for data centres with more than 500 or 1000 servers.

3.6 Chapter Summary

In this chapter, we proposed *MP-Framework* for triggering management actions into a cloud platform automatically. *MP-Framework* implements management policies described by cloud users during the design stage. The policy rule-engine is the main and most important component used in designing the framework. To show that the design of *MP-Framework* is applicable, the framework is implemented to govern the execution of triggering a live-migration of virtual machines and a reconfiguration of assigned resources for virtual machines. The implementation was used OpenNebula and Drools technologies and was applied in the Management Energy Consumption Case Study.

The cloud users can use *MP-Framework* to create an automated cloud platform that can do any automatic management scenario. Since *MP-Framework* depends on a policy rule-engine for triggering actions, cloud domain experts should specify a management policy that is mapped as a rule-set. In the next chapter, we will discuss the specification of management policies in the rule language in more detail.

CHAPTER 4

The Specification of Management Policies

In Chapter 3, we presented a framework for creating an automated cloud platform for energy management. The main aim was to provide the platform with the capability of responding to current changes with an effective management action. The management strategy executed in a cloud platform can be stored in a rule-based engine as a set of management policies. At the current stage, the challenge is how this set of management policies can be written as a rule-scheme using a rule language such as Drools. As a result, there is a requirement to formulate a generic scheme for specifying the two different rule-sets used in *MP-Framework* in the Drools Language.

Moreover, in our research context, management policies are an interpretation of a management objective assigned by cloud domain experts. Therefore, another dimension that is covered in this section is related to providing a method of describing management policies during development phases. This method is concerned with describing management policies at the authoring phase using a DSL and formulating the executable form of management policy into a rule language at the implementation level.

As a consequence, in this chapter, we will discuss in detail the management policies by outlining a process of designing such policies before they become executable in

MP-Framework. Furthermore, we will explain the classification and specification of management policies into an executable rule-scheme. We used the produced specification in designing management policies meta-model to develop a DSL called CloudMPL which is used for describing management policies at an early stage. The process of designing the language resulted from collaborating with researchers from Brazil and Aston University in Birmingham. Moreover, the specification of executable management policies proposed in this chapter is also applied in generating a conceptual mapping of rules from the CloudMPL to an executable rule language such as Drools. Both the specification of management policies and CloudMPL are applied to the Energy Management Case Study explained in Chapter 3 and Chapter 5.

4.1 The Generic Expression of the Rule Scheme Used in *MP-Framework*

As explained in Chapter 3, *MP-Framework* requires rules to be written in a certain structure. Section 3.2 in Chapter 3 described how the framework runs into phases which are monitoring and management phases. Each phase executes two different schemes written in a rule language such as Drools Rule Language [70]. The first scheme consists of a set of rules that are called *monitoring rules*. The functionality of the monitoring rules is to trigger the sensor of *MP-Framework* periodically. The second scheme includes a set of rules which are referred to as *management rules or management rule-sets*. The management rule-set is essential and must be carefully designed and developed. The management rules are those used by the policy rule-engine for executing the management action in *MP-Framework*.

During our design, we concluded that the expression of both monitoring rules and management rules are similar to production rules according to the specification of the UML-based Rule Modelling Language (URML) [112, 113]. Therefore, we are going to present a general expression for writing *monitoring rules* as well as *management rules* as UML activity diagrams for clarifying the structure of rules and for easily mapping high-level specified rules to a declarative rule-language, such as the Drools Language [70].

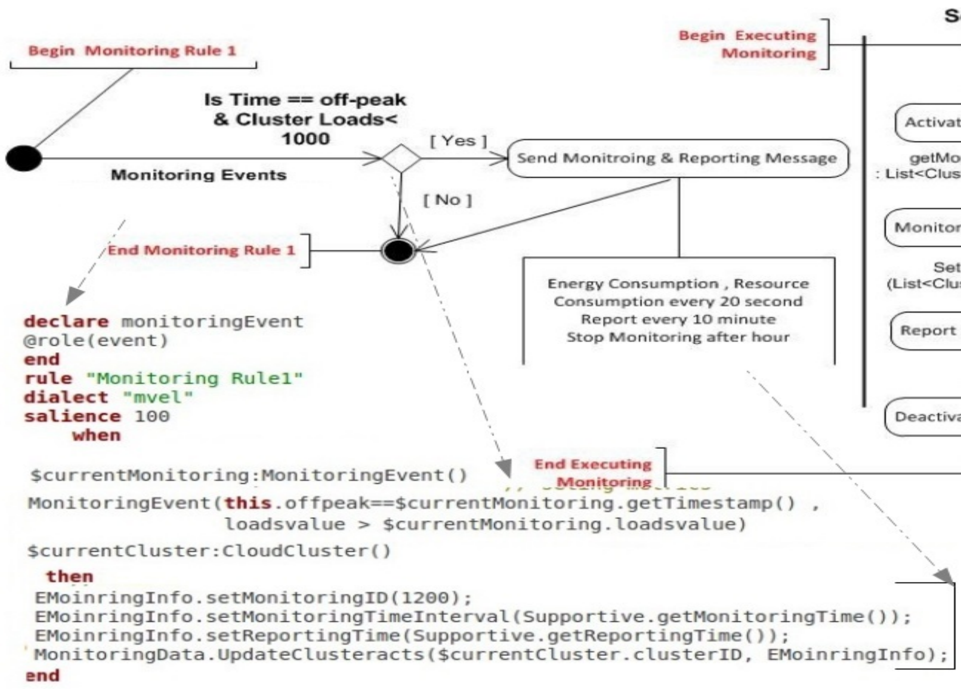
4.1.1 The General Expression of Monitoring Rules

A monitoring rule is a type of a production rule [112, 113] that triggers a set of action expressions for updating the setting of the sensor of *MP-Framework*. The rule structure of a monitoring rule includes definitions of event messages, which will be inserted as facts into the policy rule-engine of *MP-Framework*. Event facts are used in the statements of rule conditions; they state which parameters are required to be monitored by the sensor. We refer to events as monitoring events. For example, an off-time peak message and workload message can be considered as monitoring events in our Management Energy Consumption case study.

Figure 4.1 shows a UML activity diagram followed by the rule language for expressing a monitoring rule. We demonstrate one of the monitoring rules used in the Energy Management Case Study explained in Section 3.4.1 in Chapter 3. The diagram starts with a representation for monitoring events, which are an off-time peak message and a workload message. A branch maps a rule condition statement to check the time and workload values. The assertive arrow leads to the rule action part, which contains an invocation message to tell us which state to monitor, how long monitoring will last, and when to report the values to the policy rule-engine in *MP-Framework*.

4.1.2 The General Expression of Management Rules

A management rule is a kind of a production rule [112, 113] which consists of a set of conditions and post conditions. The outcome from triggering the management rule is to execute a set of action expressions (i.e. management actions in our research context) that

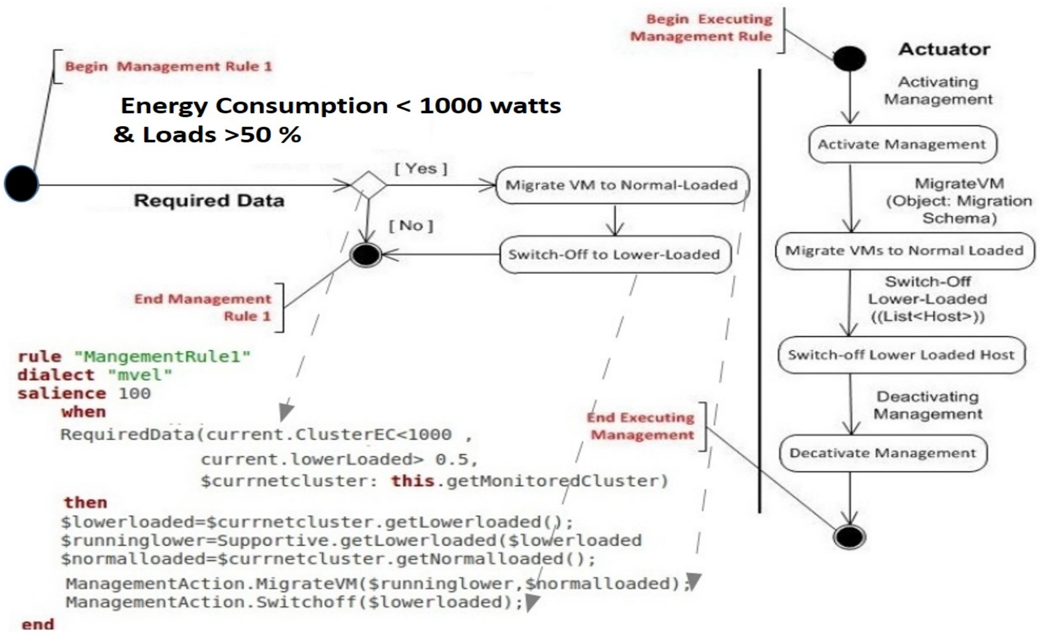

Begin Monitoring Rule 1
Is Time == off-peak & Cluster Loads< 1000
[Yes]
[No]
Monitoring Events
Send Monitroing & Reporting Message
End Monitoring Rule 1
Energy Consumption , Resource Consumption every 20 second Report every 10 minute Stop Monitoring after hour
Begin Executing Monitoring
Sensor
Activating Monitoring
Activate Monitoring
getMonitoring Data () : List<Cluster Requried Data>
Monitor Required Data
Set RequiredData (List<Cluster Requried Data>)
Report Required Data
Deactivating Monitoring
Deactivate Monitoring
End Executing Monitoring

```
declare monitoringEvent
@role(event)
end
rule "Monitoring Rule1"
dialect "mvel"
salience 100
    when

 $currentMonitoring:MonitoringEvent()
MonitoringEvent(this.offpeak==$currentMonitoring.getTimestamp() ,
                loadsvalue > $currentMonitoring.loadsvalue)
$currentCluster:CloudCluster()
  then
 EMoinringInfo.setMonitoringID(1200);
 EMoinringInfo.setMonitoringTimeInterval(Supportive.getMonitoringTime());
 EMoinringInfo.setReportingTime(Supportive.getReportingTime());
 MonitoringData.UpdateClusteracts($currentCluster.clusterID, EMoinringInfo);
end
```

Figure 4.1 – UML activity diagram and Drools language for structuring a sample of monitoring rules used in the case study mentioned in Chapter 3

change the state of the cloud platform. The condition statement of the management rule uses the reported monitored data, which are denoted as required data, and constraint values. An example of the required data are energy consumption, workload, and resource consumption values.

Depicted in Figure 4.2 is a UML activity diagram and Drools Language representation for one management rule expressed in Section 3.4.2 in Chapter 3. The rule starts by using the required data, which are the energy consumption measured in watts and the percentage of workload on hosts in the cloud platform. The assertive arrow leads to the rule consequence part, which consists of two sequential invocation management actions, which are VM migration and switching-off idle hosts. These messages are sent to the actuator of *MP-Framework* (see Figure 4.2).


Begin Management Rule 1
Energy Consumption < 1000 watts & Loads >50 %
Required Data
[Yes]
[No]
Migrate VM to Normal-Loaded
Switch-Off to Lower-Loaded
End Management Rule 1
Begin Executing Management Rule
Actuator
Activating Management
Activate Management
MigrateVM (Object: Migration Schema)
Migrate VMs to Normal Loaded
Switch-Off Lower-Loaded ((List<Host>))
Switch-off Lower Loaded Host
Deactivating Management
Decativate Management
End Executing Management

```
rule "MangementRule1"
dialect "mvel"
salience 100
    when
    RequiredData(current.ClusterEC<1000 ,
                 current.lowerLoaded> 0.5,
                 $currnetcluster: this.getMonitoredCluster)
    then
    $lowerloaded=$currnetcluster.getLowerloaded();
    $runninglower=Supportive.getLowerloaded($lowerloaded
    $normalloaded=$currnetcluster.getNormalloaded();
     ManagementAction.MigrateVM($runninglower,$normalloaded);
     ManagementAction.Switchoff($lowerloaded);
end
```

Figure 4.2 – UML activity diagram and Drools language for structuring a sample of management rules used in the case study mentioned in Chapter 3

4.2 The Definition and the Process of Designing Management Policies

Management policies are a set of compact statements which are expressed in the form of "if/then" sentences. These combined statements should be executed together to achieve a certain management objective assigned by cloud domain experts. Management policies should be easily translated into rule-sets written in a rule language in order to be executed in the proposed framework explained in Chapter 3 or similar frameworks as in [91] and [92]. We refer to the executable rule-sets as *Executable Management Policies* and similar policies written in "if/then" sentences as *Expressed Management Policies*. An example of *Executable Management Policies* is the rule-set explained in Figure 4.2.

For designing management policies and deploying them as a set of *Executable Man-*

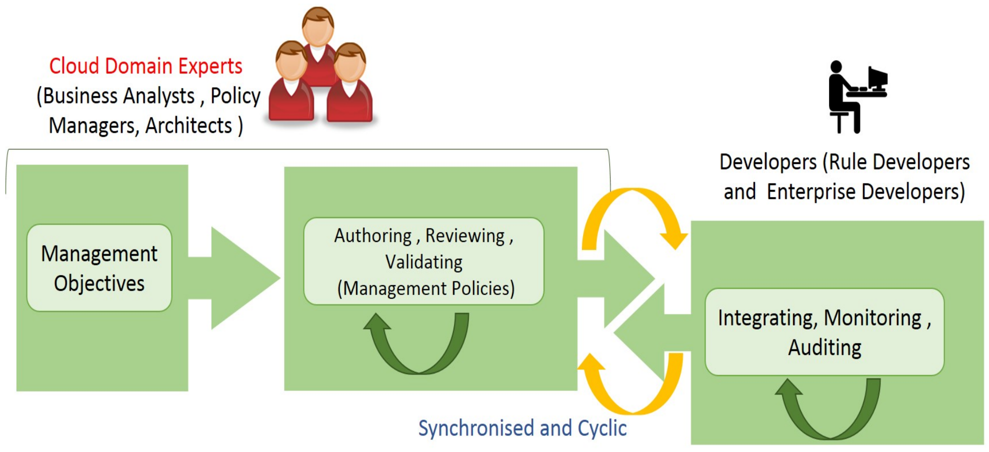

Cloud Domain Experts
(Business Analysts , Policy Managers, Architects)
Developers (Rule Developers and Enterprise Developers)
Management Objectives
Authoring , Reviewing , Validating (Management Policies)
Integrating, Monitoring , Auditing
Synchronised and Cyclic

Figure 4.3 – The phases and roles of designing management policies and integrating them in *MP-Framework* inspired by the model of designing Business Rules published by IBM in [10]

agement Policies, there are a few steps which are defined and assigned to different roles. These steps are inspired by the model proposed by IBM for designing and integrating *business rules* using ILOG JRules Business Rule Management System [10]. A business rule as defined by Ian Graham is "a compact, atomic, well-formed, declarative statement about an aspect of a business that can be expressed in terms that can be directly related to the business and its collaborators, using simple unambiguous language" [60]. An example of a business rule is *" if the Cloud Physical Host is idle and the time is night then Switch-Off Cloud Host"*. This example follows one of the defined pattern of business rules which is *If X happens then do action Y* [60].

In [10], the development of *business rules* involves a series of synchronised and cyclic steps which are performed by various roles. The first grouped steps for designing *business rules* are Authoring, Reviewing and Validating, which can be performed by business users such as Business Analysts, System Architects, Business Managers and Administrators [10]. Whilst the second grouped phases are Integrating, Monitoring and Auditing, the expressed *business rules* are accomplished by Developers [10]. We used previously mentioned steps and roles for defining the process of designing management policies. From the definition of management policies, we noticed that the forms of describing such policies, which are 'if/then' statements, are close to the specification of *business rules* as stated in [10]. Thus, in Figure 4.3, we presented the phases and the possible roles for designing management policies by integrating some elements from [10].

As shown in Figure 4.3, the process of designing management policies consists of three stages: the definition of management objectives; designing management policies, and deployment of executable management policies. Each stage of designing management policies is assigned to a different role as shown in Figure 4.3. In our research context, we refer to them as cloud domain experts who may have a little knowledge about a rule-based system. The cloud domain experts can be Business Managers, or System Architects, or Policy Managers (see Figure 4.3).

During the first step, which is the definition of management objectives, as shown in Figure 4.3, cloud domain experts assign a set of management objectives or a single management objective which can be achieved. An example of an objective would be as follows:

- The daily amount of energy consumption by the running platform must not exceed 5000 watts, and the amount of overall SLA-violation produced by running all services must not exceed £ 20.

The example above shows that the main objective is to keep both the amount of energy consumption and SLA-violation to a minimum rate. After the management objectives are defined, a series of synchronised and cyclic stages starts. At the authoring stage, cloud domain experts describe a possible set of management policies using both the objective definitions and the cloud platform domain model (the infrastructure of the cloud platform). System Architects define the essential parameters required to be monitored and a management action or a set of management actions that should be triggered (see Figure 4.3). Following that step, cloud-domain experts (System Architects or Policy Managers) also formulate a set of management policies using the defined monitored parameter(s) and management action(s). These policies can be written in simple plain English sentences which are "if/then" sentences. As a result, the defined objective(s) become *Expressed Management Policies*. To avoid any complexity that might be raised during the process of authoring policies, note that cloud-domain experts might not be experts in rules and rule-engines, but they should have a little knowledge about them. At the current step of the research, cloud-domain experts would write policies in natural language. However, in Section 4.4, we will discuss the design and the development of a DSL which can be used to describe policies at the authoring stage. The designed DSL was a result of collaborating with experts in design language(s).

During the management policies deployment step, the *Expressed Management Policies*, which were specified by cloud-domain experts, are encoded using an executable rule language such as Drools Language [70] to generate *Executable Management Policies*. Rule developers will take the role of mapping the written management policies in plain English to a rule language using the management policies meta-model and the model of the cloud platform which is referred to as the cloud platform domain model (both the management policies meta-model and the cloud platform domain model will be explained in the following sections). Following that step, developers integrate *Executable Management Policies* into *MP-Framework*. In order to accomplish this step, we assume that *MP-Framework* is created and configured to be running using any existing cloud-management system such as OpenNebula [105] which was explained in detail in Chapter 3. After the policies are executed in *MP-Framework*, the cloud platform is monitored and reviewed. The process requires both cloud domain experts and developers to evaluate the effectiveness of using the defined policies to reach the main management objectives assigned earlier. Therefore, we made the synchronisation and the repetition arrows in Figure 4.3.

4.2.1 The Classification of Management Policies

In [79], Kipp *et al.* provide a classification for collective green measurements and metrics used in a data centre. Particularly, such a classification is useful for building systems that are energy-aware. For a data centre, management can be viewed using either technical or organisational aspects or both. Management using technical parameters is related to the level of data centre's nodes and running applications such as (Resource Usage and Energy Consumption)[79]. On the other hand, management using organizational metrics considers economical, geographical and environmental aspects of the data centre [79].

Based on such a classification, we categorise the management policies which are related to governing energy consumption in a cloud-platform at low and high levels. Such a division targets different levels that might be found in the cloud platform. This classification can simplify the representation of management policies at the authoring and the deployment stages. Furthermore, the classification can assist us in creating the UML meta-model for describing a management policy which will be explained in the next section. Thus, our automatic energy management meta-model is based on a review of the existing work in a cloud-based automatic management architecture (see Chapter 2, Section 2.7); the management rules specified in [26, 92] and the classification suggested in [79]. Before presenting the meta-model for formulating management policies, an explanation of low-level and high-level management policies is covered in the following subsections.

Low-level Management Policies

Low-level or constraint-based policies use threshold rule schemes similar to the one explained in Figure 4.2. Such policies create an automatic management in a cloud platform which depends only on technical views. In other words, the measurable attributes of such policies are directly related to the platform parameters. For instance, these parameters can be resource usages such as CPU, memory, bandwidth, disk, and I/O, the cost of those resources, SLA-violation rate, or the amount of workload [79]. For such policies, the conditions of the policy can be composed of either a single or two threshold values which use comparative expressions, as will be shown in Section 4.3.1. The threshold values for both single and two threshold values can be assigned by cloud-domain experts. The possible action that might be triggered can be any management action defined by cloud-domain experts. An example of a management policy which uses only low-level attributes of the cloud platform is as follows:

1. RuleSet_1:

   When Violation_Rate is High or CPU_Usage is High or CPU_Usage is Low

   then Migrate Smallest VM Running in Host

This example is one of the advanced rule-sets used in the Energy Management Case Study mentioned in Chapter 5 in Section 5.1.

High-level Management Policies

The second type of policies are high-level; they deal with management from an organizational perspective. Such a view can be directly related to cloud consumers or the enterprises that own the cloud services. Executing services in a cloud-platform not only has an economic impact (revenues and costs) on cloud users, but also has an environmental effect which is related to the amount of CO2 emission produced by data centres that belong to a cloud consumer [79] (see the energy consumption figures of cloud-data centres in [129]). In our research, we applied the automatic management to use only one metric related to the organisational aspect. This metric is the impact on the cost of energy consumption based on changes in the time-zone and the geographical location of running services in the cloud platform. Such a metric will be used to formulate time-based and location-based policies. However, the other organisational aspect mentioned in [79] such as the economic view can follow a similar scheme using different measurable parameters and sensor interfaces. Thus, high-level policies can be extend to include features that are beneficial to the cloud consumers. For example, management policies that look at management from the economic view would have an *Executable Management Policy* that includes the concept of revenue or profit and uses a cost model for running services.

Similarly to the low-level policy, the conditions of the high-level policy might include single or two thresholds. However, the expression of the conditions of such a policy will be different and use operators different from the ones used in conditions of a low-level policy. The differences will be clarified when the management policy meta-model is explained in Section 4.3. The following management policy is an example of a time-based policy which is extracted from the case study mentioned in Chapter 5 Section 5.1.

1. RuleSet_2: (Applied between Private_Hosts)

   When Private_Host(x) can accept Migrated VM and Time is after 16.00 until 7.00 and Private_Host(y) can migrate VM

   then Allow only one VM to be migrated between Private_Host(x) and Private_Host(y) every Δ_{time}.

4.3 The Management Policies Meta-Model for Executable Rule Language

One of the challenges is creating an UML meta-model for formulating management policies. This is because we want to have a model which is enriched with enough operators, vocabularies, and expressions so that it can be used to formulate various sets of policies. The produced UML meta-model for management policy is based on the existing meta-model of URML [112]. The URML meta-model includes a specification for modelling production rules which is one type of modelled rules [112]. We noticed that the rule scheme generated from using the specification of production rules is close to the scheme for the management rule shown in Figure 4.2. Thus, any management policy (low-level and high-level policies) can be modelled as a production rule.

Figure 4.4 is an abstract meta-model for modelling a management policy. Each management policy should have a cloud platform domain model. The cloud platform domain model in Figure 4.4 is the one used in the advanced Energy Management Case Study explained in Section 5.2 in Chapter 5. This cloud platform domain model focuses on a target host and its set of monitored parameters. The target host has a set of running virtual machines which can be migrated during its life-cycle (for more explanation about this domain model, see Section 5.2 in Chapter 5). As presented in Figure 4.4, any management policy is encoded as a special form of production rule (more information about the productions rules can be found in [112] and [113]). These production rules follow a defined specification for formulating both condition and action parts in a rule language which can be applied to either Drools [70] or JRules [69]. Each production rule is composed of a set of conditions and a set of actions. The condition part of the rule can have none or more post conditions as well as a condition. This means that each management policy must have at least one condition.

Conversely, the production rule must have an action expression. The action expression is a statement that would be executed if its conditions are satisfied. The action expression can be of two types: *InvocationActionExpr* or *AssignVariableExpr*. The *InvocationActionExpr* is the statement that invokes an operation or a function. Here, the *InvocationActionExpr* can use the operations of the cloud manager which are shown in Figure 4.4. The *AssignVariableExpr* is a statement that would set the required variables such as the values of monitored parameters with numerical or real values.

In the following sections, we will discuss the specification for formulating conditions and action parts for a policy in an executable rule language. The specification includes the meta-model for conditions, the classification for operators used in conditions, and a

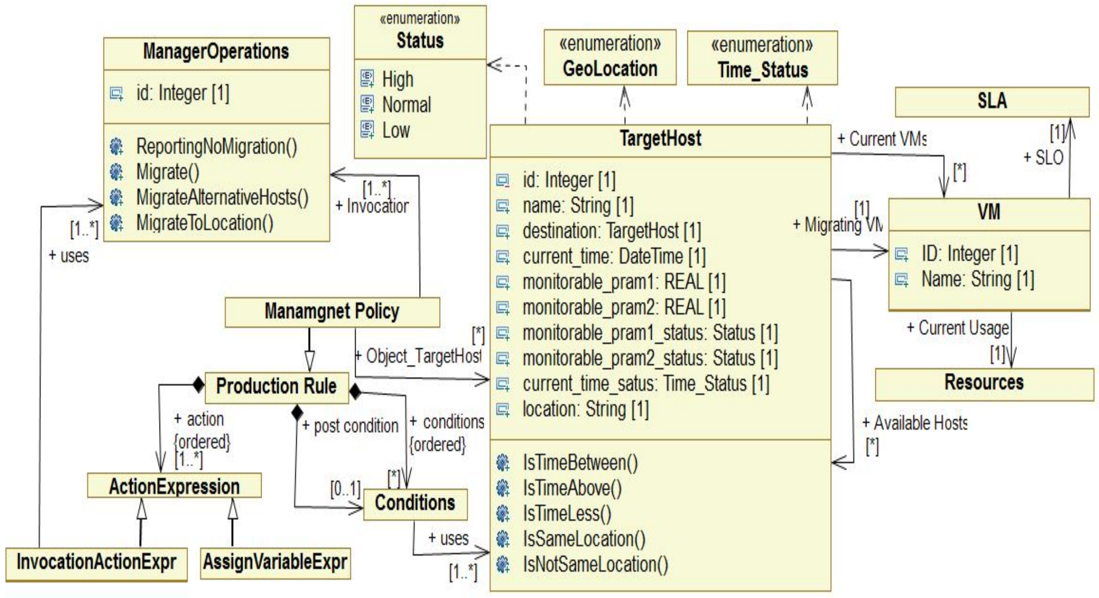


Figure 4.4 – The abstract UML meta-model for describing management policies

Expression	<property:>	<operator:>	<value_expr:>
Constraints-1:	monitorable_prams	Comparison	DataTerm
Constraints-2:	monitorable_pram_ status	Level Specification	DataTerm
SelectTargetHost:	id or name	TargetHost Selection	DataTerm
TargetHost Location:	location	Location	DataTerm
TargetHostTime:	current_time_status	Time	ObjectTerm

Table 4.1 – The Syntax for Conditional Expressions for a Management Policy

description of the actions used by a management policy.

4.3.1 Conditions Meta-Model for a Management Policy

Conditions for a management policy can be expressed through the meta-model presented in Figure 4.5 which was inspired by the URML meta-model [112]. This meta-model resulted from our classification for general rules used for management purposes in cloud platforms mentioned in Section 4.2.1. Particularly, we used constraint-based policies, time-based policies, and location-based policies.

In Figure 4.5, a single condition in a management policy is a Boolean expression

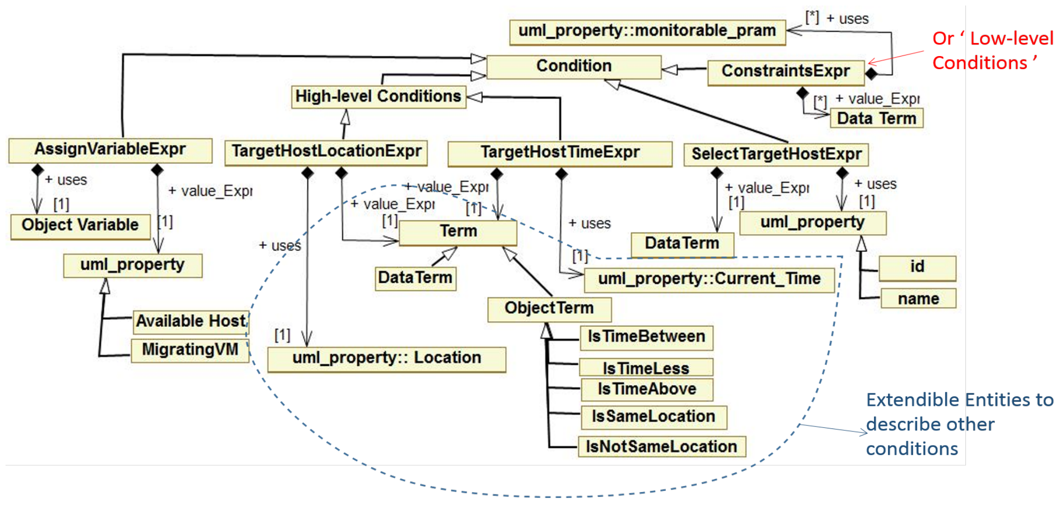


Figure 4.5 – The meta-model for conditions (low-level and high-level conditions) of a management policy in rule language (Drools and JRules)

which can be composed with other conditions by using *composition operators*. From the URML rule meta-model [112], we extracted some elements for modelling various conditions. These elements are *Term*, *DataTerm*, *ObjectTerm*, *uml_property*, and *Object Variable* [112].

In Figure 4.5, the conditional expressions are classified into five types. Each expression in the condition meta-model uses a property. The property is extracted from the target host that is running in a cloud platform. In addition, each expression also has a *value_expr* which might be of the following types: Data Term, Object Term, or uml_property. Furthermore, suitable operators are grouped to match each conditional expression type. Figure 4.6 presents the operators. Thus, referring to Figure 4.5 and Figure 4.6, each conditional expression and its syntax are explained as follows:

1. **ConstraintsExpr or Low-level Condition:** a comparative condition used to compare monitorable parameters against a specified threshold value or to specify the status of the monitorable parameters. Examples of monitorable parameters are CPU_Usage, Violation_Percentage and Energy_Consumption. ConstraintsExpr has two different syntaxes, which are presented in Table 4.1.

2. **SelectTargetHostExpr:** an identification expression, used to select a targeted host. The *SelectTargetHostExpr* expression must be included in a management policy. The expression syntax is shown in Table 4.1.

3. **AssignVariableExpr:** a selection expression, used to get values from some properties and to assign them to an object variable. This expression is an optional statement, which can be used in a management policy for extracting variables which are required by management APIs. The syntax for the expression is different from the syntax for other expressions. It is as follows:

   $< operator : Assignment \ \$ >< property : ObjectVariable >$

   $< operator : Assignment :>< property : ObjectTerm >$

4. **High-level Conditions:** an extensible component that is used to formulate organisational conditions. In our research context, we extended the component to include two types of expressions suitable for describing location and time conditions of running cloud services. Such expressions are directly related to the Energy Consumption Case Study used in this PhD thesis. The expressions are as follows:

   (a) **TargetHostLocationExpr:** a location-based expression, used to specify the geographical location of a target host. Since a physical cloud host can be located at any location around the world, the policy meta-model should allow an option for such a restriction. This expression is optional in the policy based on the requirement. The syntax for the expression is shown in Table 4.1, where its Data Term can be either a String type or a GeoLocation which is an Enumeration type. An example of TargetHostLocation is: $location == GeoLocation.Asia$.

   (b) **TargetHostTimeExpr:** a time-based expression that specifies the time status at a target host. Any target host in a cloud-platform has some operations to deal with time expression.

   These operations are $IsTimeBetween(< Time\_Begin >, < Time\_End >)$, $IsTimeLess(< Time\_Literal >)$, and $IsTimeAbove(< Time\_Literal >)$. Table 4.1 presents

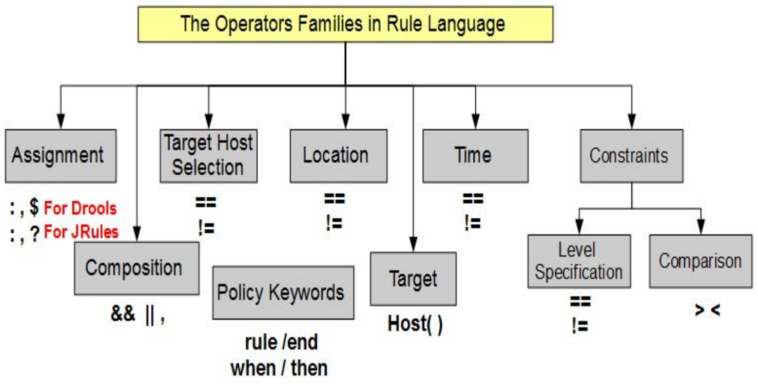


Figure 4.6 – The possible common rule language operator families for expressing a management policy

the syntax for the time-based expression. The following expression is a simple expression for checking time status:

current_time_status== IsTimeBetween(16.00,23.00)

As shown in Figure 4.5, the entities used for describing both TargetHostLocationExpr and TargetHostTimeExpr are extensible. Therefore, other high-level conditions can be described after considering the requirements of management assigned by cloud domain experts.

4.3.2 Policy Action Description

The action of a management policy in a rule language is expressed as an action expression. Action expressions can be either expressions for assigning values or expressions for invocation actions. To simplify the transformation process in the future, we use only invocation action expressions from the rule language, which is denoted as *InvocationActionExpr* in [112]. In any management policy, the invocation action expressions include calls for management APIs/Operations specified in a cloud manager. The syntax used for expressing *InvocationActionExpr* is:

CloudManager.Operation_Name(parameters);

In this syntax, *CloudManager* represents the instance of a cloud manager which has a

$< OperationName >$	$< Parameters : Type >$
Migrate	Original: TargetHost
MigrateAlternativeHosts	Original: TargetHost , Destination1:TargetHost, Destination2: TargetHost
MigrateToLocation	Original: TargetHost , LocationName: String
ReportingNoMigration	Original: TargetHost
Calculate	Original:TargetHost

Table 4.2 – Examples of Operations Used by Cloud Manager Instance in a Management Policy Described in Drools Used in Case Study in Chapters 3 and 5

number of management operations. Here, we specified that each defined operation for the *CloudManager* instance has a number of parameters which are necessary for the migration of virtual machines, reporting information and calculating service at the target host side in a cloud-platform. Table 4.2 shows each defined operation and its related parameters which are examples of operations are used in the Energy Management Case Study described in Chapters 3 and 5. Nevertheless, it is an extensible model which allows any management operation to be defined depending on specified management actions configured in *MP-Framework*.

4.4 The Usage of the Specification of Management Policies

The specification is used in two different domains. The first domain employs the specification for designing a DSL for authoring policies at the description level. The second domain applies the specification for developing mapping rules from the designed DSL at the policy authoring level to generate an executable management policy in the rule language. The whole process was achieved by collaborating with Andre Almeida, a PhD researcher at Federal Institute of Science of Education at Science and Technology in Parnamirim in Brazil, and Nelly Bencomo, a Lecturer at the School of Computer Science at Aston University in Birmingham. Together, we designed a language called CloudMPL.

The following section offers a brief description of CloudMPL, but more details about this language can be found in [19].

4.4.1 Using Management Policies Specification in CloudMPL

Almeida and Bencomo proposed a DSL called CloudMPL. Its main objective is to be the foundation of authoring management policies used by cloud-domain experts, who may have little knowledge about formulating rules. CloudMPL can be applied instead of using plain English sentences to write a management policy during the authoring stage [19]. CloudMPL is a textual language that was specifically designed to be used by cloud-domain experts to describe management policies before translating them to an executable rule language [19]. CloudMPL is enriched with domain vocabularies and expressive operators for expressing conditions and action parts which were partially inspired by the RELAX Language [130]. CloudMPL is tailored to the authoring of management policies which will be executed in an Infrastructure-as-a-Service (IaaS) cloud model [19]. Almeida and Bencomo designed the CloudMPL meta-model, its syntax, and its grammar using our specification of management policies explained in previous sections. The current implementation is available for download at `http://www.dimap.ufrn.br/splmonitoring/adaptmcloud/index.php`. We applied the CloudMPL language to describe some of the management policies used in the Energy Management Case Study, which will be explained in Section 4.5.

4.4.2 Designing Transformation Rules from CloudMPL to Drools

The specification of management policies described in Section 4.3 and the CloudMPL specification proposed by the collaborative group were used in developing the conceptual mapping rules from CloudMPL into Drools. The objective was to build the foundation for automatically generating an executable management policy from the policy authoring level. This step will be considered as a basic step for auto-generating higher-level management policies, which will be a future research.

The transformation process required us to design a set of mapping rules from CloudMPL to Drools for both conditions and action parts. To design these mapping rules, we used the CloudMPL meta-model as well as its syntax and the Drools specification mentioned in Section 4.3. Firstly, the mapping step began by presenting the mapping for the general syntax for a management policy and keywords in both CloudMPL and Drools.

Table 4.3 shows the mapping of generic syntax and special keywords from CloudMPL to Drools. It is noticeable from the generic syntax in Table 4.3 that any statement between *IF* and *THEN* is mapped as conditions in Drools, which should be enclosed with the target operator mentioned in Figure 4.6. Furthermore, any statement after *THEN* is mapped as an action in Drools. The mapping of both conditions and actions requires further explanation, which will appear in the following subsections.

Mapping Conditions

In CloudMPL, a condition block consists of one or more conditions. Therefore, any condition in CloudMPL can be structured as an attribute, an operator and a value. The attribute in CloudMPL is usually written before the CloudMPL operator. Whilst, the

CloudMPL Generic Syntax		Drools Generic Syntax
POLICY $< ID >$	$\rightarrow$	**rule** $< ID >$
IF	$\rightarrow$	**when**
$< CONDITIONS >$		$< Host(Conditions) >$
THEN	$\rightarrow$	**then** $< Actions >$
$< ACTION\_INVOCATION >$		
$\{< ACTION\_INVOCATION > \}$		end

Table 4.3 – Mapping Generic Syntax and Keywords for a Policy in CloudMPL to Drools

CloudMPL Expression	**Drools Expression**
<attribute:TIME> *AFTER* <value:threshold>	TargetHostTimeExpr
<attribute:TIME> *BEFORE* <value:threshold>	TargetHostTimeExpr
<attribute:TIME> *BETWEEN* < value: threshold_a > *TO* <value:threshold_b>	TargetHostTimeExpr
IN <value:ID> <value:location>	SelectTargetHostExpr + TargetHostLocationExpr
<attribute:monitorable> *FEW_AS* | *MANY_AS* <value:threshold>	ConstraintsExpr-1
<attribute:monitorable>*IS* <value:status>	ConstraintsExpr-2

Table 4.4 – Mapping CloudMPL Conditions to Drools Conditions Using CloudMPL Specification [19] and Table 4.3 in Section4.3

value is used after the CloudMPL operator. Thus, Table 4.4 shows the mapping for conditions that apply the following mapping rules:

1. The dot operator < . > in CloudMPL is mapped as '==' operator and <value:ID or Name> is mapped as <value_expr: DataTerm>.

2. *After* operator is mapped as '==' combined with <ObjectTerm:IsTimeAbove> in Drools.

3. *Before* is mapped as '==' combined with <ObjectTerm:IsTimeLess> in Drools.

4. *BETWEEN*,*TO* operator is mapped as '==' combined with <ObjectTerm: *IsTimeBetween*>.

5. < value: threshold > is mapped as <Time_Literal> parameter for both IsTimeAbove and IsTimeLess in Drools.

6. < value: threshold_a > and <value:threshold_b> are mapped as <Time_ Literal_ Begin> and <Time_Literal_End> parameters for IsTimeBetween in Drools.

7. <attribute:Time> is mapped as <property: current_time_status>.

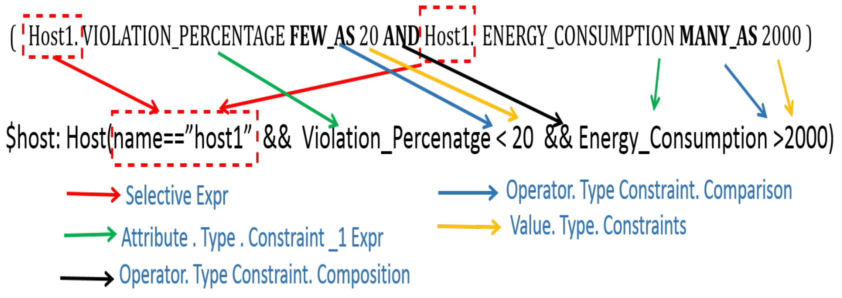

(Host1. VIOLATION_PERCENTAGE FEW_AS 20 AND Host1. ENERGY_CONSUMPTION MANY_AS 2000)
$host: Host(name=="host1" && Violation_Percenatge < 20 && Energy_Consumption >2000)
Selective Expr
Attribute . Type . Constraint _1 Expr
Operator. Type Constraint. Composition
Operator. Type Constraint. Comparison
Value. Type. Constraints

Figure 4.7 – Using mapping rules for mapping CloudMPL condition block to Drools condition part for a policy

8. *IN* is mapped as '==' operator and <value:location> is mapped as <value_expr: DataTerm> which can be either String or Enumeration.

9. *FEW_AS* or *MANY_AS* are mapped as Comparison operators.

10. <attribute:monitorable> is mapped as <property: monitorable_parms> and <value: threshold> is mapped as DataTerm.

11. *IS* operator is mapped as '==' operator and <value:status> is mapped as <DataTerm: Status>.

12. <attribute:monitorable> in *IS* expression is mapped as <property: monitoriable_ Parms_ status>.

13. *AND* and *OR* is mapped as && and || operators in Drools.

To elaborate the mapping of a condition of a management policy, we provide a sample of conditions written in both CloudMPL and Drools in Figure 4.7. These conditions express one of the management policies extracted from the Energy Management of the Running Example presented in Section 4.5. In Figure 4.7, the first statement is a CloudMPL expression for three conditions, which are Violation_Percentage *FEW_AS* 20, Energy_Consumption *MORE_AS* 2000, $host1$. These conditions are composed in CloudMPL by the *AND* operator. The same figure also includes conditions expressed in

Drools which map CloudMPL conditions. In Figure 4.7, the arrows represent the types of the mapping rules that can be applied.

Mapping Actions

In CloudMPL, any action statement is mapped as a call method for management operations in a policy expressed in Drools. The mapping rules for actions are:

1. Action $< ID >$ in CloudMPL is mapped as the Name of the operation in CloudManager (see Figure 4.8).

2. The parameters List, which includes a set of Parameter Expression, is mapped as the operation parameters in CloudManager.

3. In CloudMPL, Parameter Expressions consists of $< TypeID >$. Type is mapped as either $< ObjectTerm >$ or $< getObjectRef >$ in Operation. Whilst ID is the mapped as the name of the parameter.

4. In CloudMPL , if Type is Host and it is the first parameter in the statement, then it is mapped as the Original and its type is TargetHost in Drools.

5. Conversely, in CloudMPL, if Type is Host and is not the first parameter, then it is mapped to be either Destination1 or Destination2 based on ordering parameters in Drools.

Figure 4.8 illustrates the application of the mapping rules for an action from CloudMPL to Drools. The figure includes an operation defined in Table 4.2. The operation has three parameters which are Host1, Host2 , and Host3. In Drools mapping, Action ID is mapped as Migrate_Alternative_Host, whilst the first parameter is mapped as $\$host1$. Both Host2 and Host3 are mapped as $\$host1.getHost2()$ and $\$host1.getHost3()$, respectively.

The previously mentioned mapping rules for both conditions and actions parts will be used to design an interpreter to automatically or semi-automatically generate Drools

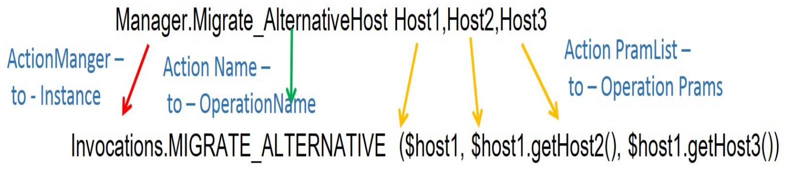

Manager.Migrate_AlternativeHost Host1,Host2,Host3
ActionManger – to - Instance
Action Name – to – OperationName
Action PramList – to – Operation Prams
Invocations.MIGRATE_ALTERNATIVE ($host1, $host1.getHost2(), $host1.getHost3())

Figure 4.8 – Using mapping rules for mapping a CloudMPL action block to a Drools action part for a policy

codes for policies that would be executed into *MP-Framework* as explained in Chapter 3. The Drools code generation will be a future project.

4.5 The Application to the Energy Management Case Study

We applied CloudMPL to express a number of management policies extracted from the Management Energy Consumption Case Study presented in Chapters 3 and 5. We took some management policies encoded into Drools and used CloudMPL to write them. Figure 4.9 shows both CloudMPL declarations and CloudMPL policies for the case study, which were implemented using XText [132].

Figure 4.9 shows that there are six policies expressed in CloudMPL which demonstrate the usage of all operators suggested by the language. Policy 1 is a constraint-based policy, which requires monitorable parameters and uses the CloudMPL operators *FEW_AS* and *MANY_AS*. In turn, policy 2, policy 3, and policy 4 are composed of both time and constraint expressions. Both policy 2 and policy 4 use the operator *After*, whereas policy 3 includes the ClodMPL time operator *Between / To*. The constraints operator used in these policies is *IS*. Policy 5 has only a single time expression which uses the CloudMPL time operator *Before*. The final policy, which is policy 6, contains a location expression besides the time and constraint conditions. The location condition uses the CloudMPL

```
Define Resource Host as
begin
Violation_Percentage as Number
Violation_Status as Status
CPU_Usage as Status
Energy_Consumption as Number
Current_Time as Time
End
DEFINE ACTIONMANAGER Manager AS
BEGIN
ACTION Migrate_Alternative_Host Host origin,
Host alternativeA , Host alternativeB
ACTION NoMigrate
ACTION MigrateLocation Host origin, Location location
END
CREATE host1,host2,host3 AS Host
```

(a) CloudMPL Declarations

```
POLICY policy1
IF ((host1.Violation_Percentage FEW_AS 20) AND (host1.Energy_Consumption MANY_AS 2000))
 THEN
Manager.Migrate_Alternative_Host host1 , host2, host3

POLICY policy2
IF ((host1.Current_Time AFTER 23:00) AND ((host1.Violation_Status IS HIGH) OR
(host1.CPU_Usage is HIGH)))  THEN  Manager.NoMigrate

POLICY policy3
IF ((host1.Current_Time BETWEEN 01:00 TO 07:00) AND ((host1.Violation_Status 20 IS HIGH) OR
(host1.CPU_Usage is HIGH)))
THEN Manager.Migrate_Alternative_Host host1 , host2, host3
```

(b) CloudMPL Expressions_1

```
POLICY policy4
IF ((host1.Current_Time AFTER 23:00) AND ((host1.Violation_Status IS HIGH)
OR (host1.CPU_Usage is HIGH)))  THEN
Manager.NoMigrate

POLICY policy5
IF ((host1.Current_Time BEFORE 09:00))  THEN
Manager.NoMigrate

POLICY policy6
IF ((host1.Location in 'ASIA') and (host1.Current_Time Between 16:00 to
23:00) and ((host1.Violation_Status IS HIGH) OR (host1.CPU_Usage is HIGH)))
Service.MigrateLocation  host1, 'ASIA'
```

(c) CloudMPL Expressions_2

Figure 4.9 – A sample of CloudMPL (XText) for management policies used in the case study

operator *IN*.

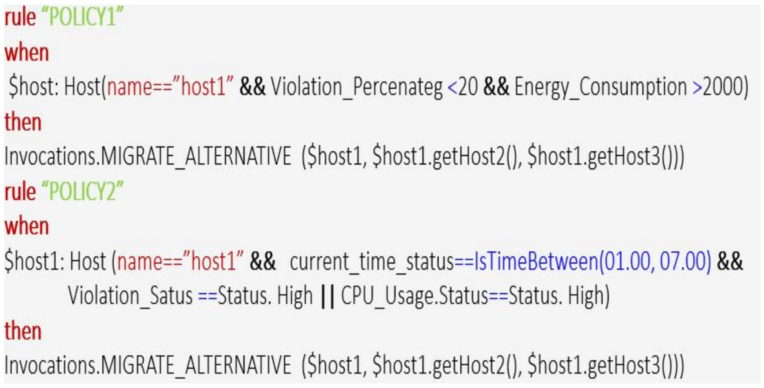
4.5.1 The Interpretation of CloudMPL Policies to Drools

We applied the mapping process introduced in Section 4.4.2 to the policies of the case study. Using the designed mapping rules explained in Section 4.4.2, we generated Drools code manually. To test the mapping rules, Figure 4.10 presents a sample of Drools code for policy 1, policy 3, policy 5, and policy 6, which are presented in two groups. The important Drools operators used in the conditions are highlighted in blue.

Taking policy 3 as an example, this policy is mapped as a combination of Time and Constraints_2 Expressions which are shown in the management policy conditions meta-model mentioned in Section 4.3. The mapping for the condition part applies the rule numbers 1, 4, 6, 7,11, 12 and 13 as explained in mapping conditions in Section 4.4.2. We applied all rules proposed for mapping the action part expressed in Section 4.4.2. As a result, policy 3 will have a rule code similar to these illustrated in Figure 4.10. We applied this method is applied to all remaining CloudMPL policies captured in Figure 4.9.

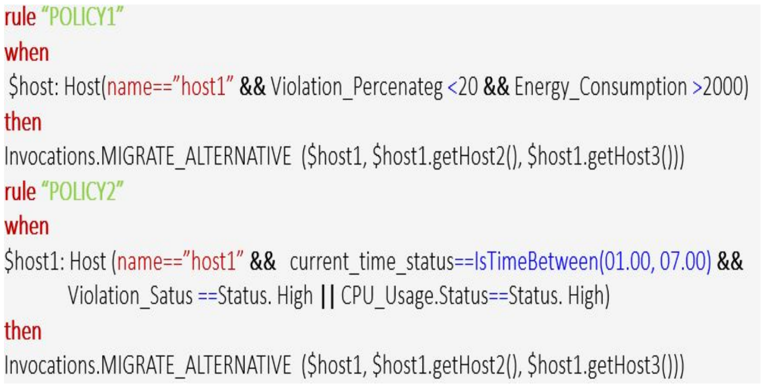
4.6 Chapter Summary

In this chapter, we classified management policies into low-level and high-level. In addition, we provided a specification for formulating the executable form of these policies in rule-language using URML specification. The generated specifications were applied in designing the CloudMPL language and the mapping rules from CloudMPL to Drools. CloudMPL is a DSL which aims at narrowing the existing gap between the specification of the *Expressed Management Policies* in the policy authoring step and the implementation of similar policies via a rule language. CloudMPL establishes a set of operators that deal with several kinds of constraints, from ordinal through to time and location constraints, that can be applied to a specific or a set of cloud resources. In addition, CloudMPL



```
rule "POLICY1"
when
 $host: Host(name=="host1" && Violation_Percenateg <20 && Energy_Consumption >2000)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
rule "POLICY2"
when
$host1: Host (name=="host1" &&  current_time_status==IsTimeBetween(01.00, 07.00) &&
        Violation_Satus ==Status. High || CPU_Usage.Status==Status. High)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
```

(a) Drools Rules "Group_1"



```
rule "POLICY1"
when
 $host: Host(name=="host1" && Violation_Percenateg <20 && Energy_Consumption >2000)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
rule "POLICY2"
when
$host1: Host (name=="host1" &&  current_time_status==IsTimeBetween(01.00, 07.00) &&
        Violation_Satus ==Status. High || CPU_Usage.Status==Status. High)
then
Invocations.MIGRATE_ALTERNATIVE  ($host1, $host1.getHost2(), $host1.getHost3()))
```

(b) Drools Rules "Group_2"

Figure 4.10 – The generated Drools rules from CloudMPL policies

supports user-defined actions to deal with the consequences of conditions specified by the managers of cloud computing infrastructure. The usage of both CloudMPL and the automated approach, which is based on designing mapping rules between CloudMPL and Drools, was illustrated with the help of an example related to the management of energy consumption by migrating virtual machines. Both the specification of management policies and CloudMPL can be extended to include other types of high-level or low-level parameters related to management energy in a data centre such as the one defined in [79]. By fully implementing both CloudMPL and the automated code-generation for creating management policies, we can decrease the amount of time required in implementing such policies which is considered as a future step. Due to the closeness of CloudMPL to natural languages and its declarative nature, it is possible for the language to be used by cloud domain experts for specifying policies.

Using both the *MP-Framework* and the specification of management policies covered in this chapter, an **Automated Managed Cloud Platform** can be created. A cloud user will have a platform that can be easily configured and integrated to operate with any cloud management system such as OpenNebula. The integration requires developers to only develop suitable interfaces for probing and triggering management actions. The challenge was how management policies were designed and developed in a simple executable form in a rule language. We tackled this challenge with the specification proposed in this chapter. The other issue solved in this thesis is how to model the *Automatically Managed Cloud Platform* executing management policies to evaluate the effectiveness of the designed management policies before executing them in a real cloud platform. This challenge will be discussed in the following chapters.

CHAPTER 5

Modelling Management Policies and Cloud Platforms via Coloured Petri-nets

The cost values associated with triggering any management actions caused by the execution of management policies in a cloud platform can be unpredictable. As discussed in Section 4.2.1 in Chapter 4, management policies can be in the form of either low-level, high-level policies, or a combination of both. Low-level policies as constraint rule-sets might involve the comparison of the values of variables with given threshold values. On the other hand, both time-based and location-based rule-sets, which are considered as high-level, are formed from Boolean constraints. As a result, management policies that would run on an automatically managed cloud platform can be very complex and might interact with each other.

Not only do the interactions and the complexity of management policies affect the cost of triggering dynamic actions in a cloud platform, but the massive architecture of the cloud environment could also be considered highly dynamic. The complexity of a cloud platform is due to the usage of virtualisation technology [81]. In turn, the increases in the dynamism of a cloud platform are caused by variability in the received workload and the complexity in the underlying interaction between the components of the platform [81]. As a result, triggering any management action resulting from running management policies has an impact on certain cost values which might be crucial to cloud users. For example, if the management policy triggers a migration action during its time execution, cost values would include the energy consumption cost and the virtual machine migration cost. Thus, there is a clear need to identify the most appropriate management policy from a set of potential ones. Cloud-domain experts and rule developers can know the effect on the cost of energy consumption of using various set of management policies using real observations after implementing the policies to run in the cloud-platform. However, such a method is time-consuming and increases the maintenance and the reviewing cycle of enforcing management policies. Instead, we are looking to find a method for estimating costs of implementing two different set of policies, such as p_1, p_2 before executing them in a real cloud platform. Hence, the remaining chapters make a contribution in modelling and analysing management policies that would run in *MP-Framework* or similar frameworks such as those in [26] and [92].

The suggested modelling approach is applied by following a systematic strategy for analysing run-time management policies. This can be accomplished by modelling both a cloud platform with dynamic management behaviours and a run-time policy via CPNs. An example of such behaviours is the monitoring and the management processes which are the run-time phases found in *MP-Framework*. The generated CPN models can be used to compute maximum and minimum costs associated with automatically triggering a management action. Specifically, we investigate the effect of repeating the triggering migration action of a virtual machine on both energy consumption cost and transmission cost or migration cost. The modelling and the analysis strategy can be done before implementing management policies in a real cloud platform.

In this chapter, we will explain the modelling concept of a cloud platform with dy-

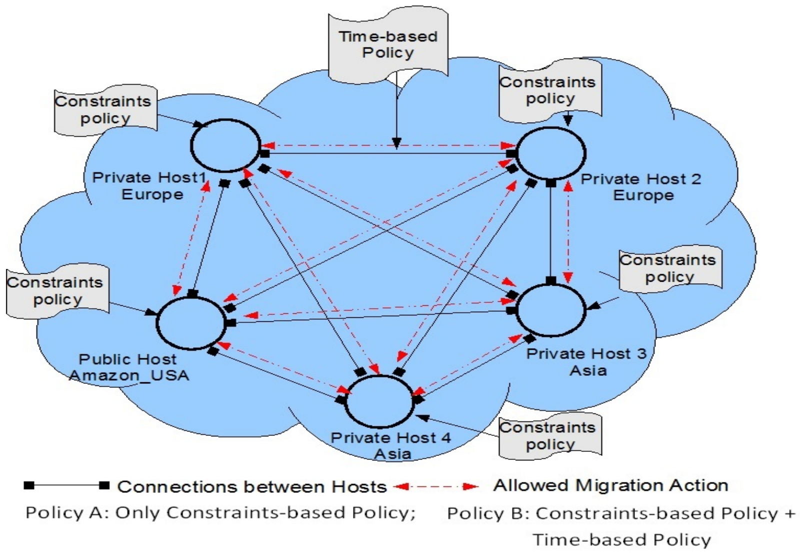


Figure 5.1 – The case study description with possible implementation for Policy A or Policy B

namic migration behaviour and run-time policy using CPNs. We will provide a formal mathematical definition for a cloud platform consisting of two dynamic behaviours: monitoring and migration. Furthermore, we will present a method to model both low-level and high-level policies in CPNs. The modelling concept was applied in an advanced Energy Consumption Case Study using various types of management policies, which will be presented in the following section.

5.1 Advanced Energy Management Case Study

This case study is an extension of the ones explained in Chapter 3. To clarify the Petri-net formalism for both a cloud platform and a set of management policies, let us consider the scenario presented in Figure 5.1.

Figure 5.1 shows the architecture for a platform containing four Private Hosts in different locations and one Public Host. Two Private Hosts are located in Europe whereas the two remaining Hosts are in Asia. The Public Cloud is using Amazon instances, which are located in the USA. All Hosts in the platform are connecting together. Furthermore, a dynamic migration action for moving VM images is allowed among all Hosts. The Cloud domain experts that own this architecture want to restrict the live-migration action among hosts using two suggested policies, which are Policy A and Policy B. Policy A is a Constraint-based Policy applied at each node in the cloud platform (see Figure 5.1), whilst Policy B is a Constraint-based Policy applied at each node alongside a Time-based Policy which is described later.

A Constraint-based Policy uses threshold values for some low-levels which are written as rule-sets described as follows:
Constraint-based Policy:

1. RuleSet_1:

   When Violation_Rate is High or CPU_Usage is High or CPU_Usage is Low

   then Migrate Smallest VM Running in Host

2. RuleSet_2:

   When Violation_Rate is Normal and CPU_Usage is Normal

   then Accept Migrated VM

Time-based Policy is based on using the node timestamps and migrating only during off-peak time. The policy is explained as:
Time-based Policy

1. RuleSet_3: (Applied between Private_Hosts)

   (a) **When** Private_Host(x) can accept Migrated VM and Time is after 16.00 until 7.00
   and Private_Host(y) can migrate VM
   then
   Allow only one VM to be migrated between Private_Host(x) and Private_Host(y) every Δ_{time}.

2. RuleSet_4:(Applied between Private_Host and Public _Host)
    (a) **When** Public_Host(x) can accept Migrated VM and Time is after 18.00 until 23.00 and Private_Host(y) can Migrate

        then

        Allow VM to be Migrated to Public_Host(x) every Δ_{time}.
3. RuleSet_5: (Applied at Public _Host)
    (a) **When** Time is after 23.00 at Public_Host(x)

        then

        Allow VM to be Migrated to Private Hosts from Public_Host(x) every Δ_{time}.

A cloud consumer or a cloud domain expert aims to know which Policy A or Policy B is better in terms of Energy Saving before implementation.

5.2 The Formalism of Cloud Platform and Management Policies in Coloured Petri-nets (CPNs)

Before explaining the formal method for modelling both cloud platform and management policies in CPN, we created a class diagram that captures the necessary components of a typical cloud environment similar to the one shown in Figure 5.1 to assist us during the modelling process (see Figure 5.2). We developed the class diagram based on our experience with the OpenNebula toolkit [105] and used a data model suited for constraint rules suggested in [26] and [92].

Looking at Figure 5.2, a typical cloud-platform consists of three main components: Host, VM images and SLA classes. The Host class is one of the cloud components that run various VM instances. The Host class has a number of attributes used as measurement parameters during the monitoring process. Examples of these parameters are Total_CPU_Usage and Average_Violation. The Total_CPU_Usage attribute represents

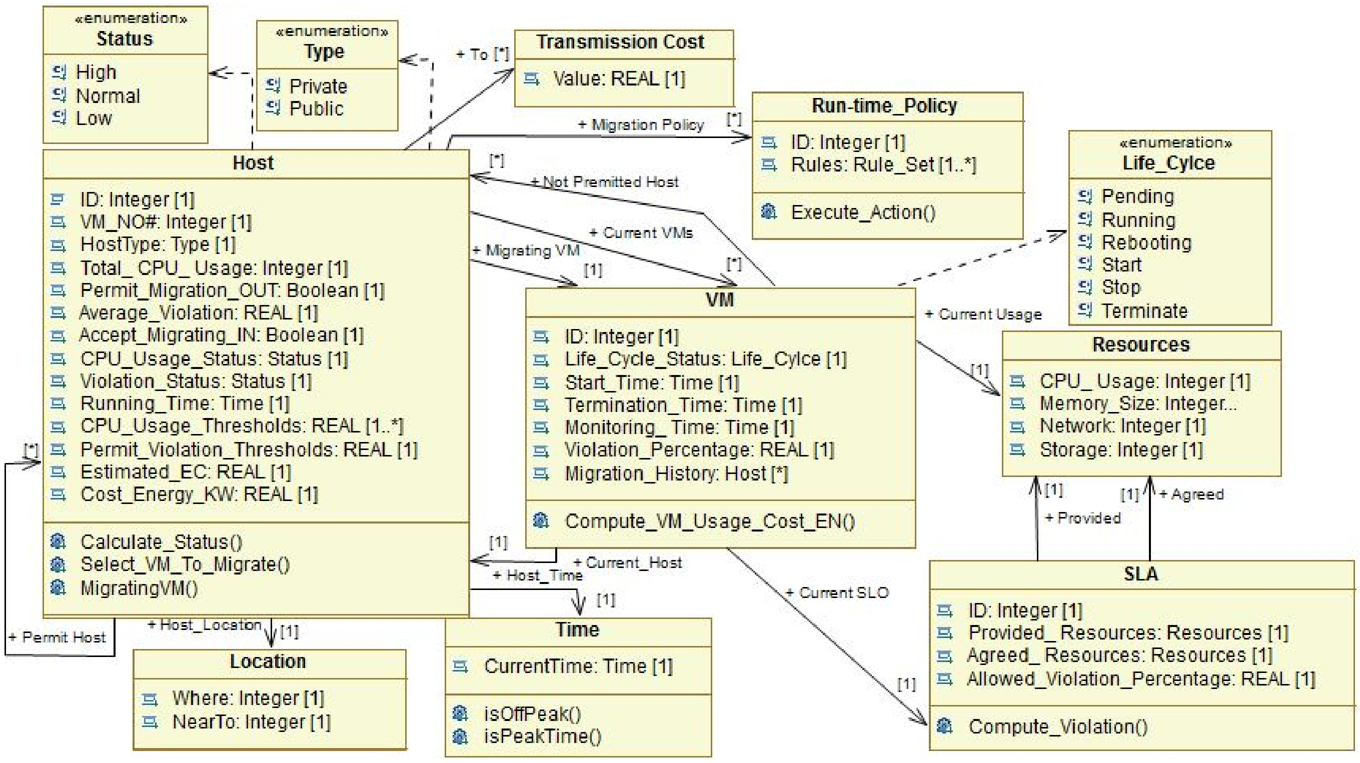


Figure 5.2 – The class diagram of a cloud platform with a management policy similar to the one presented in Figure 5.1

the average of shared CPU use among running VM instances on the host. The Average_Violation is a parameter that represents the percentage of violation in the amount of resources provided for each running virtual machines on the host. The Host has threshold parameters for both CPU Usage and Violation attributes. Furthermore, in Figure 5.2 each Host has a set of Current_VMs and a Migrating_VM. Migrating_VM instance is a representation of the selected VM image, which will be migrated to another host at $time_i$. In addition, the Host class has cost-related parameters and a location attribute, which represents the geographical location for the host.

We assume that each running VM image in a cloud platform belongs to only one cloud consumer. Thus, each running VM instance is associated with only one SLA and also has a life cycle. The attribute life_cycle of a virtual machine can have various states after being assigned to a cloud user (see Figure 5.2). Here, we assume that the VM life_cycle is running because we model the live-migration behaviour (for more information about the migration of VM, see Section 2.6 in Chapter 2). Each VM instance has a current SLA with time-limits. Time-limits are computed after the deployment of the virtual machine and assigned to a cloud consumer. Time limits are specified with two timestamp attributes in the VM class, which are Start and Termination. For example, the start time for VM1 is 7.00, and the termination time is 23.00. In Figure 5.2, the SLA class includes a description of the agreed amount of resources. The agreed attribute represents the amount of resources allowed to be consumed by a cloud consumer. SLAs are often agreed during the contract establishment process. In addition, the SLA also has the currently provided resources parameter, which is the amount of currently allocated resource to a VM image during its execution time. Those attributes are based on the case study mentioned in [26, 91, 92].

The Runtime_Policy class contains a set of migration rules and methods for executing actions supposed to be performed by the rules. Actions in our context enforce the migration behaviour among all running hosts.

5.2.1 Modelling a Cloud-Platform in Coloured Petri-nets

Since CPNs are designed for modelling concurrent systems [73, 74], a CPN is ideal for modelling the cloud platform as well as run-time policies (see Sections 2.3 and 2.4 in Chapter 2 for detail about Petri-nets). The dynamic migration behaviour which occurs between hosts in a cloud can be modelled as transitions in a CPN. Each place p in a CPN is modelled as the host of the platform which has three defined states: **Monitoring State**, **Permit Migration State** and **Accept Migration State**. The **Monitoring State** is an indication that the place Host is in the monitoring process. The **Permit Migration State** is an indication that the place Host is allowing a virtual machine to be migrated to another host, which is in the **Accept Migration State**. In turn, the **Accept**

Migration State is an indication that a place Host can accept a virtual machine from hosts that are in the **Permit Migration State**. The token of moving virtual machine is the one that is allowed to move across the CPN model of the cloud which is represented as the Colour Set named VM, shown in Table 5.1.

Each place p_i has two types of transitions: monitoring transition and migration transition. The monitoring transition is used to model the monitoring behaviour of the place p_i, which will be explained later. This transition has an arch to and from the same place. In contrast, the migration transition is used to model the live-migration action which will be from a place p_i to another place p_j.

5.2.2 The Formal CPN_{Cloud} Definition and Colour Sets Declarations

Based on the CPN definition in Section 2.4.1 in Chapter 2, our CPN model is formally denoted as a CPN_{Cloud}. The CPN_{Cloud} has the following elements:
$CPN_{Cloud} = (P_{Hosts}, T, A, \Sigma_{cloud}, V, C, G_{policy}, E)$, where

1. P_{Hosts} consists of a finite set of running hosts with three possible states which are Monitoring state, Permit Migration state and Accept Migration state.

2. $T = T_{Monitoring} \cup T_{Migration}$ such that $T_{Monitoring} \cap T_{Migration} = \emptyset$, where $T_{Monitoring}$ is a finite set of *monitoring transitions* which are fired during the monitoring process of *MP-Framework*. $T_{Migration}$ is a finite set of *migration transitions* between places which are enabled during the management process of *MP-Framework*.

3. Σ_{cloud} Cloud contains of all colour sets (see Table 5.1 for more information about the names and the declarations for colour sets in Σ_{cloud} and part of CPN ML in Figure 5.2).

4. G_{policy} is a finite set of guards which represent modelling High-level Policy (more

details about the modelling policies will be explained in the following sections)

5. A, V, C, and E are the original CPN elements as explained in Section 2.4.1 in Chapter 2.

5.2.3 Modelling Dynamic Behaviours of MP-Framework

The CPN_{Cloud} should mimic the functionality of *MP-Framework*. The functionality of *MP-Framework* during both the Monitoring and Management phases is explained in Section 3.2 in Chapter 3. This functionality is modelled into two dynamic behaviours: the Monitoring and Management behaviours in CPN_{Cloud}.

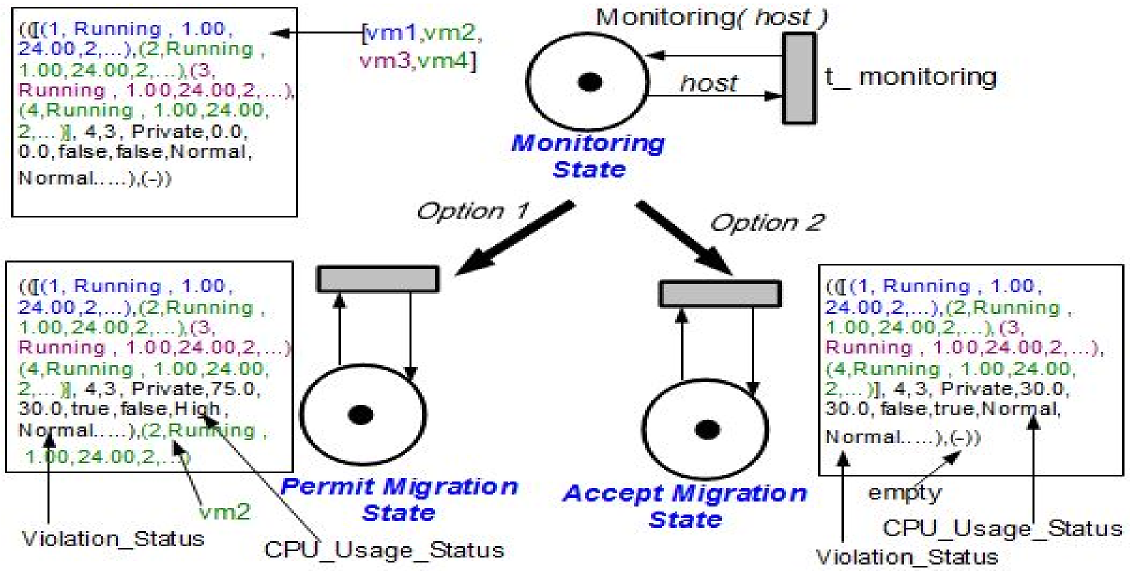
Modelling the Monitoring Behaviour

In our modelling context, each place has to monitor then migrate depending on the values of the monitored parameters. In CPN_{Cloud}, we can say that p has a monitoring state when the values for both Permit_Migration_OUT and Accept_Migrating_IN are false (see Table 5.1). The monitoring process is done by firing $t_i \in T_{Monitoring}$ at each host place with the time delay denoted as @MonitoringTime. The time delay is specified as a guard at any $t_i \in T_{Monitoring}$.

Figure 5.3 depicts a single transition representing a monitoring activity in a host to describe changes in hosts' colours after firing the transition. It begins with p place on the top, which has four running virtual machines and host information. The Host information includes CPU_Usage, Violation_Percentage, and other information which are shown as numbers. Furthermore, p place also has Migrating VM, which is empty at the beginning of each monitoring phase. When $t_{monitoring}$ is enabled all the information is bound as a host variable. The host is passed to the Monitoring(host) expression. The Monitoring(host) expression is an ML function that implements measurement for vm parameters such as CPU_usage and Violation_Percentage. Furthermore, the ML method also defines the state

Σ **Name**	**Attributes**
lifeCycle	with Pending—Run—Reboot—Start—Stop—Termination
status	with High—Normal—Low
type	Private—Public
Resources	CPU_ Usage:INT Memory_Size:INT Network:INT Storage:INT
transmissionCost	value:REAL
location	Where:INT NearTo:INT
SLA	ID:INT AllowedViolation:REAL
VM	ID:INT Start_Time:Time Termination_Time:Time Monitoring_ Time: Time Violation_Percentage: REAL Migration_History: list ID
Host	ID:INT VM_NO:INT HostType:TYPE Total_ CPU_ Usage: REAL Average_Violation: REAL Permit_Migration_OUT: BOOL Accept_Migrating_IN: BOOL Running_Time: TIME CPU_Usage_Thresholds:product REAL*REAL Permit_Violation_Thresholds: REAL Estimation_ECost: REAL Estimation_ EC: REAL Pre_Estimation_EC:REAL Cost_Energy_Per_KW: REAL Watt_Per_Usage: REAL

Table 5.1 – Declarations of Colour Sets in Σ_{cloud}



(([(1, Running , 1.00, 24.00,2,...),(2,Running , 1.00,24.00,2,...),(3, Running , 1.00,24.00,2,...), (4,Running , 1.00,24.00, 2,...)], 4,3, Private,0.0, 0.0,false,false,Normal, Normal.....),(-))
[vm1,vm2, vm3,vm4]
Monitoring(host)
host
t_ monitoring
Monitoring State
Option 1
Option 2
(([(1, Running , 1.00, 24.00,2,...),(2,Running , 1.00,24.00,2,...),(3, Running , 1.00,24.00,2,...) (4,Running , 1.00,24.00, 2,...)], 4,3, Private,75.0, 30.0,true, false,High, Normal.....),(2,Running , 1.00,24.00,2,...))
Violation_Status
vm2
CPU_Usage_Status
Permit Migration State
Accept Migration State
(([(1, Running , 1.00, 24.00,2,...),(2,Running , 1.00,24.00,2,...),(3, Running , 1.00,24.00,2,...), (4,Running , 1.00,24.00, 2,...)], 4,3, Private,30.0, 30.0, false,true,Normal, Normal.....),(-))
empty
CPU_Usage_Status
Violation_Status

Figure 5.3 – Modelling monitoring behaviour in $p \in P_{Host}$ with two options to occur

for the host at the end of the monitoring phase.

At the end of executing the ML method, the state of a place can be either Permit Migration or Accept Migration. When the value of the Permit_Migration_OUT colour is true and the value of the Accept_Migrating_IN colour is false, then p has a Permit Migration state. On the other hand, the state of p is Accept Migration when the value of the Permit_Migration_OUT colour is false and the value of the Accept_Migrating_IN colour is true. For example, we are using the scenario in Figure 5.3. If CPU_Usage_Status is High and Violation_Status is Normal, then the host has a Permit Migration state. Thus, the Migrating VM colour set in the host place will have the information about the selected VM for migration. Furthermore, in our modelled scenario in Figure 5.3, if the CPU_Usage_State is Normal and the Violation_State is Normal, then the state of the host place is Accept Migrating. Thus, the Migrating VM colour set in the host place will be empty.

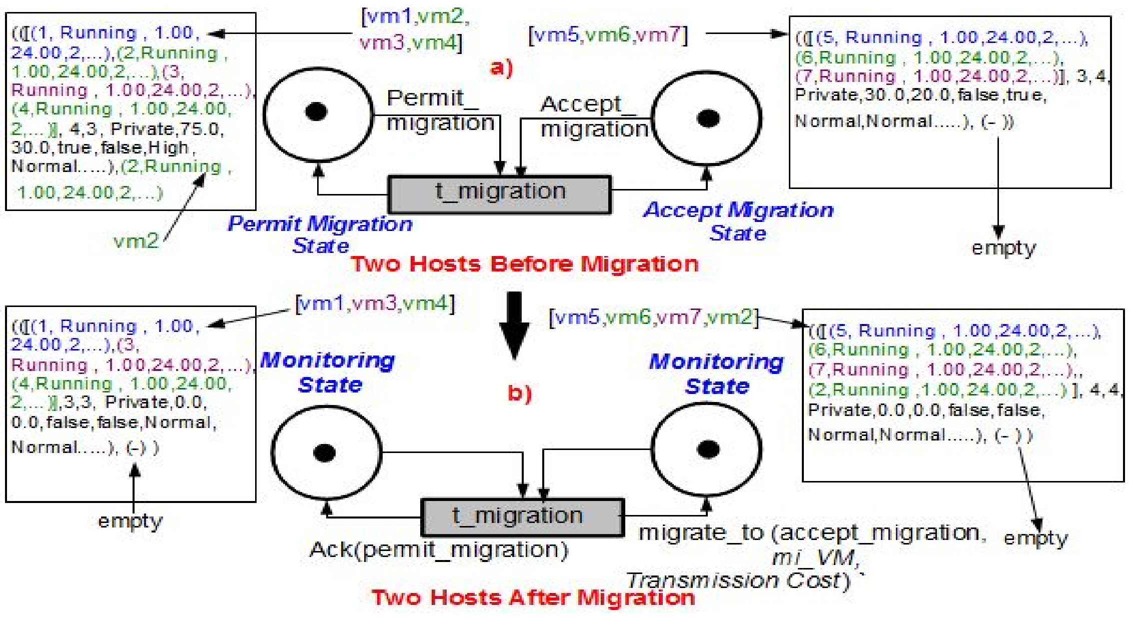


Figure 5.4 – Modelling migration behaviour in CPN_{Cloud} before and after firing $t_{migration}$ between two hosts with different states

Modelling the Dynamic Migration Behaviour

CPN_{Cloud} models the migration behaviour as a management action at the end of each monitoring cycle. The behaviour is modelled by moving the values of the Migrating VM colour from a place with the Permit Migrating state to any place which has an Accept Migrating state (see Figure 5.4 for a model of the complete behaviour).

Figure 5.4 shows the colour set values and status in both a place with the Permit Migration state and a place with the Accept Migration state before and after the migration process. Looking at Sub-Figure 5.4.(a), the Migrating_VM in a place with Permit Migration has a value, which is vm2, whilst the Migrating_VM colour in the other place is empty. To fire the migration transition, the place with the Permit Migration state has to send the value of the Migrating_VM colour as well as the parameters required by the high-level policy. At the opposite side, the place with the Accept Migrating state has to send only the parameters which are required by the high-level policy.

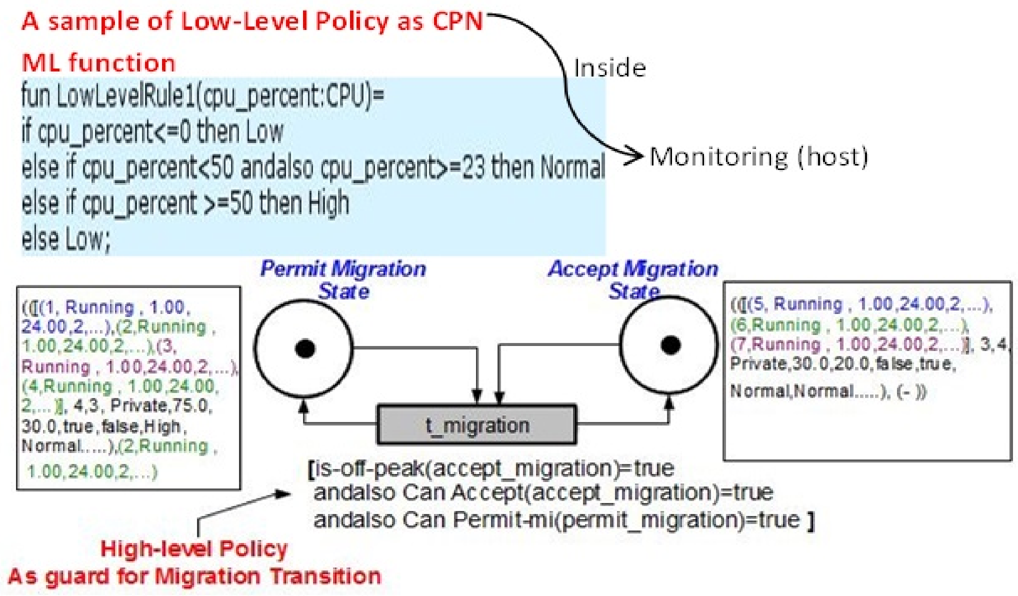
Sub-Figure5.4.(b) captures the values of colour sets in a place with the Permit Migration state and a place with the Accept Migration state after completing the migration process. After migration, the place with the Permit Migration state receives the Ack() message which it leads to set the value the Permit_Migration_OUT to be false. As a result, the place with the Permit Migration state will transfer the Life_Cycle_Status of the Migrating VM in the Current_VM list to Stop and remove it from the list. The Migrating_VM colour in the place will become empty. Meanwhile, the place with the Accept Migration state receives the values of the Migrating_VM colour set and the Transmission Cost. At this place, the received Migrating_VM colour set will be added to the Current_VM list colour and its Life_Cycle_Status will be changed from Pending to Running. At the end of the migration stage, the state for both places will be transformed to the Monitoring state. This means that the monitoring phase starts again.

5.2.4 Modelling Management Policies in CPN

As explained previously, the CPN model has to impose two types of Management policy, namely low-level, and high-level (see Section 4.2.1 in Chapter 4 for more information about the classification of Management Policies). In the class diagram in Figure 5.2, there is a Run-time policy class. This class is where rule-sets for low-level and high-level policies are defined.

Modelling Low-level Management Policy

Since the low-level policies such as Constraint rules are applied at the host level, we modelled a low-level rule-set as a function written in CPN ML language called low-Level-Rules. The Low-Level-Rules function is executed when $t \in T_{Monitoring}$ is fired. Figure 5.5 presents a sample of one of the Low-Level Rules expressed in the CPN ML function, which is used to identify the state of CPU_Usage on a host in the model.


A sample of Low-Level Policy as CPN ML function
Inside
Monitoring (host)
Permit Migration State
Accept Migration State
t_migration
High-level Policy As guard for Migration Transition

```
fun LowLevelRule1(cpu_percent:CPU)=
if cpu_percent<=0 then Low
else if cpu_percent<50 andalso cpu_percent>=23 then Normal
else if cpu_percent >=50 then High
else Low;
```

```
([(1, Running , 1.00,
24.00,2,...),(2,Running ,
1.00,24.00,2,...),(3,
Running , 1.00,24.00,2,...),
(4,Running , 1.00,24.00,
2,... )], 4,3, Private,75.0,
30.0,true,false,High,
Normal.....),(2,Running ,
1.00,24.00,2,...)
```

```
([(5, Running , 1.00,24.00,2,...),
(6,Running , 1.00,24.00,2,...),
(7,Running , 1.00,24.00,2,...)], 3,4,
Private,30.0,20.0,false,true,
Normal,Normal.....), (- ))
```

```
[is-off-peak(accept_migration)=true
 andalso Can Accept(accept_migration)=true
 andalso Can Permit-mi(permit_migration)=true ]
```

Figure 5.5 – Modelling time-based policy as a guard function in $t_{migration}$ transition

Modelling High-level Management Policy

In CPN_{Cloud}, any High-level policy is applied between any place with the Permit Migration state and any place with the Accept Migration state. As a result, the action part in a high-level rule-set is modelled as the migration transition $t_{migration}$. Whilst any condition in a high-level rule-set is described as a guard at any migration transition $t \in T_{Migration}$. These guards are denoted as G_{policy}.

Any $g_a \in G_{policy}$ consists of two or more conditions, which should be evaluated as true (see Figure 5.5). Our proposed Run-time Policy consists of a set of conditions. These conditions could be modelled as "if statements" in guards. Modelling rules as a set of "if statements" in guards of CPN_{Cloud} would reduce the readability of the model and might increase the errors in the model. This is because of the complex expression of compositional conditions which can be found in the high-level Policy when the number of Host places is increased. Therefore, we model each condition as a CPN ML function. Each function can return either true or false but in order to execute the high-level Policy, all used CPN ML functions in g_a should be true. All conditions in the guard are separated by either *andalso* or *orelse* CPN ML keywords used (see modelling a time-based Policy in Figure 5.5).

5.3 Chapter Summary

In this chapter, we investigated the possibility of applying a model-based approach as a modelling tool which will be used as a first step for assessing two different sets of management policies. In this research context, we focused on studying the effect of triggering a migration action when it is enforced using *MP-Framework*. By using CPN, we successfully modelled a cloud platform with dynamic migration and policies related to control of such behaviour. Any CPN_{Cloud} model consists of places as running hosts and migration transition between host places. Furthermore, low-level policies become a part of a monitoring transition, which is applied to each host place. On the other hand, high-level policy is modelled as a guard at each migration transition in the CPN_{Cloud} model. The generated CPN model has become a graphical simulator for a cloud platform, which can assist in analysing the costs of both energy and transmission related to migration policies before implementation. Any generated CPN_{Cloud} model will be simulated to produce a finite set of traces of execution. These traces should be analysed via the Cost Calculation Method which will be explained in detail in the following chapters.

CHAPTER 6

The Simulation-based Cost Calculation Method (SCCM) for Analysing Management Policies

The proposed CPN_{Cloud} model explained in the previous chapter can become a tool that can be used to analyse management policies. The modelling approach will be an alternative for evaluating potential running policies either by observing their execution in a real cloud platform or by testing them using existing cloud simulators such as CloudSim [36], icanCloud[104] and GreenCloud [80]. Because the CPN tool is empowered with a simulation for generating models and state graphs, the effect of system dynamic behaviours can be studied in depth [73, 74, 75]. Literally, the CPN tool can generate traces of execution that can form either the whole or partial reachability graph for the modelled system. Any traces of execution of the CPN model contains states that show the occurrence of each dynamic behaviour of the cloud platform [74, 75].

In our investigated case, we were concerned with computing the cost of triggering migration actions from each produced trace. Therefore, we investigated two given sets of such policies: p_1, p_2. For example, p_1 is a constraint-based Policy and p_2 includes constraint-based as well as time-based Policies. How can we analyse and identify which one is suitable in terms of energy savings? This can be achieved by applying the following steps:

1. Model both p_1, p_2 as CPN models which generate CPN_{Cloud} models.
2. Run the simulator of each CPN Model and generate various execution paths.
3. Each produced execution path, the Cost Calculating Method, which will be explained in this chapter or expressed in Chapter 7, is applied to compute the cost of both the energy consumption and the transmission of virtual machines along the execution path.
4. Get the average of the computed cost values for each CPN model. Then, compare the produced results produced from the cost calculation method which will be explained later.

In this chapter, we will explain the Simulation-based Cost Calculating Method (SCCM) used to compute costs from a single trace. The method is based on an application for Cost Computing from Timed Petri-nets proposed by Abdulla and Mayr in [15, 16, 17] which is modified to be applicable in a cloud platform that might trigger a number of migration actions during the execution of a management policy. We did not model cloud and policies using similar Petri-nets models suggested in [15, 16, 17]. These models are Priced Timed Petri-net (PTPN) and Priced Petri-nets (PPN). The reason for not using PTPN and PPN models is that these models have simple the expressions for describing both tokens and places. A cloud platform has a complex structure and requires a model allowing complex data-types to be defined. Furthermore, we have a set of policies, which consist of logical constraints and would be difficult to model in simple PTPN or PPN models.

Both the modelling aspect proposed in Chapter 5 and the SCCM are applied to evaluate two sets of management policies to determine which one is better in term of saving energy.

6.1 The Method for Calculating the Cost

Any single trace of executions produced by the CPN model contains markings generated from firing transitions in the model. Therefore, an execution trace extracted from any CPN_{Cloud} contains the markings resulting from firing both monitoring and migration transitions. To assess costs between two policies, we simulate the CPN model and extract several traces of execution from each CPN model for policies run for 24 hours. Then, we computed the cost across each trace. We can identify which traces have the maximum and the minimum cost. Before explaining the Cost Calculating Method, we assume the following:

1. The host type can be either Private or Public.
2. A host has a power model for estimating energy consumption. In our work, we implemented the power model proposed by [26], but any alternative models can be used. The value computed from the power model will be stored at the colour #Estimation_EC in a place. The formula for the power model is as follows:

$$E_i = min(E_i) + CPU\_Usage_i * (max(E_i) - min(E_i)) \tag{1}$$

   Where E_i is the amount of energy consumed in watt per hour unit by a host. CPU_Usage_i is the average of shared CPU usage among virtual machines running in the host. $max(E_i)$ is the maximum amount of energy consumed by the host when it has a maximum load; and $min(E_i)$ is the minimum energy consumption by the host when it is in the idle state [26].
3. Transmission cost is the cost of live-migration of a virtual machine between hosts which depends on migration duration time and the cost of SLA-violation [35]. The migration time is based on the total amount of memory used by the VM and the

available network bandwidth [35], which is computed as:

$$MigrationTime = \frac{VM_{image\_size}}{Bandwidth_{available}} \tag{2}$$

4. We simplify the calculation by assuming that there is no delay between hosts; therefore, we did not include in the transmission cost the cost of the delay. It is straightforward to add the delay cost and extend the cost model.

Based on these assumptions, the calculation method uses the approach for computing cost from timed Petri-net proposed in [15, 16]. Therefore, for a given a trace of execution:

$$\begin{gathered}\sigma := \quad M_0 \overset{(t_0,\theta_0)}{\longrightarrow} M_1 \overset{(t_1,\theta_1)}{\longrightarrow} M_2 \ldots \overset{(t_{n-1},\theta_{n-1})}{\longrightarrow} M_n \\ 0 \leqslant \theta_0 \leqslant \theta_1 \leqslant \theta_2 \ldots \leqslant \theta_n \leqslant 24\end{gathered} \tag{3}$$

We divided the cost in σ into two parts:

1. The cost when virtual machines are running on hosts denoted as $ECost$.
2. The cost of migrating a virtual machine from one host to another is denoted as $TCost$.

We made a simplification in the model that each virtual machine is sent instantaneously but we can add the delay cost for future work. Suppose that both $ECost(M_i)$ and $TCost(M_i)$ are known when a transition t_{i-1} at time θ_{i-1} was fired. Therefore, there are two cases at the markings $M_i \overset{(t_i,\theta_i)}{\longrightarrow} M_{i+1}$ when t_i at time θ_i is fired:

Case 1: When t_i is a monitoring transition. In this case the cost had already been calculated before θ_{i-1} and stored at $ECost(M_i)$. During the firing of a monitoring transition, the amount of energy consumed during time period $[\theta_{i-1}, \theta_i]$ on the basis of virtual machine loads is calculated and stored as a value in the colour $\#Estimation\_EC$ at a place

p. Thus, the cost of energy at the marking M_{i+1} is as follows:

$$ECost(M_{i+1}) \quad = \quad ECost(M_i) + \sum_{p \in P_{Host}} \#Estimation\_EC(p) \times Price(p) \qquad (4)$$

Where the sum is taken by multiplying the amount of energy consumed stored $\#Estimation\_EC$ at a place over period of $[\theta_{i-1}, \theta_i]$ with local $Price(p)$ for all hosts $p \in P_{Host}$. The price of the energy per watt is given by a cloud provider. Naturally, since no migration happens, the cost of migration at M_{i+1} is the same as M_i.

$$TCost(M_{i+1}) = TCost(M_i) \qquad (5)$$

Case 2: When t_i is a migration transition. In this case, a virtual machine moves between hosts. Therefore, the transmission cost at the marking M_{i+1} is the value of $TCost$ in previous marking M_i added to the cost of transmitting a virtual machine.

$$TCost(M_{i+1}) = TCost(M_i) + MigrationTime \times Price(SLA\_Violation_{min}) \qquad (6)$$

In which $MigrationTime$ is computed using Equation 2 whereas the price of SLA_violation is the minimum price that should be paid. The value of the minimum price is specified in the SLA.

Moreover, during the firing migration transition t_i between the period of $[\theta_{i-1}, \theta_i]$, the $ECost(M_i + 1)$ is calculated using Equation 4.

By applying this method, the costs of energy consumption, transmission cost and the value of estimated energy consumption can be calculated along an execution path σ for 24 hours. Similarly, if we have a number of execution paths $\sigma_1, \sigma_2, \ldots, \sigma_n$, we can apply the cost calculation method proposed for σ at each one. Thus, the paths that have the maximum and the minimum energy cost values can be identified.

6.2 Evaluation of CPN Cloud Model and Simulation-based Cost Calculating Method (SCCM)

We evaluated both the modelling of CPN_{Cloud} and the Simulation-based Cost Calculation Method (SCCM) using the advanced management energy consumption case study mentioned in Section 5.1 in Chapter 5, which involved a comparison between two different Management Policies. The policies would be applied to a management system for five cloud hosting nodes (See Section 5.1).

6.2.1 Simulating the Case Study with CPN_{Cloud}

We applied our proposed Petri-net model CPN_{Cloud} to generate two CPN_{Cloud} models for the described management policies for the case study described in Section 5.1 in Chapter 5. One model implements Policy A whereas the other model uses Policy B. We configured both models to have five places. These places represent the host nodes mentioned in the case study (see Figure 5.1 in Chapter 5). Each host place except the Amazon place has five virtual machines, whereas the Amazon place has only three virtual machines. In both models, we created ML methods to randomly generate loads at virtual machines which randomly change loads after triggering monitoring and migration transitions. It is possible to replace the load generator ML method with other load generators such as JMeter [119].

As explained in Section 6.1, some values required for calculation are stored in the markings. These values are Estimated_EC, Migration Time and Cost_Energy_Per_Kwh which is $Price(p)$. We used the CPN ML function to compute and store the values at the marking. We started the experiment by implementing only Policy A. We applied the Constraint-rules at all five places. Then, we started the Petri-net simulator and analysed the migration behaviour. We extracted the execution path and collected the data for 24

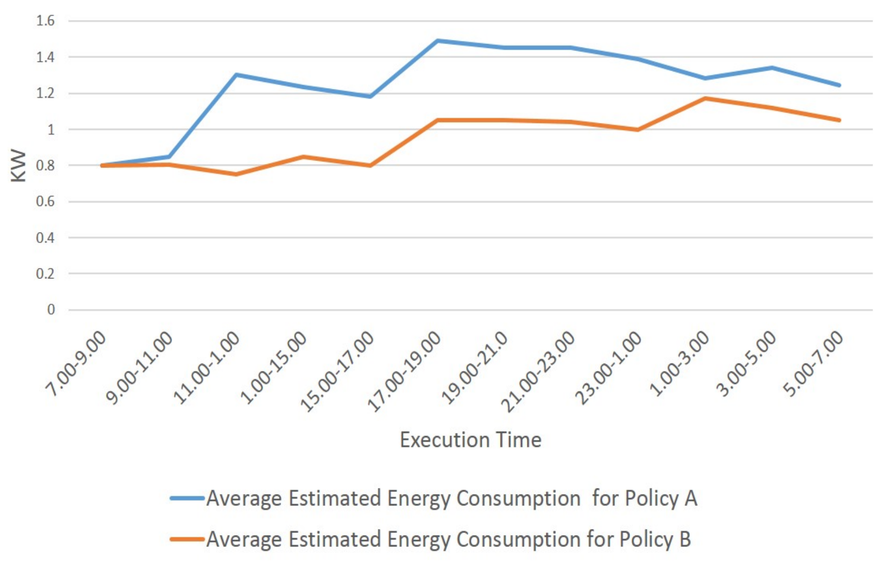


Figure 6.1 – The average of Estimated Energy Consumption for both Policy A and Policy B during 24 hours

hours. We repeated the iteration nine times, and we computed the values of estimated energy consumption, energy cost and transmission cost by applying the cost calculation method at each path. Using the CPN_{Cloud} model for Policy B, we repeated similar steps applied for analysing Policy A. The experiment was conducted on a Samsung laptop which has a 2.40GHz Intel(R)Core(TM) processor and 6GB memory

6.2.2 The Results and Discussion

From all nine execution paths from both models, we calculated the averages for the results obtained. We present the results for the averages as detailed in three graphs, which are illustrated in Figures 6.1, 6.2, and 6.3. Furthermore, we present bar charts that demonstrate the average of the total Energy Cost, Transmission Cost and Estimated Energy Consumption from modelling both policies. From the obtained graphs from the CPN_{Cloud} models, we can analyse whether Policy A or Policy B is more efficient in terms of Energy saving.

Figure 6.1 illustrates the average Estimated Energy Consumption by hosts in the CPN

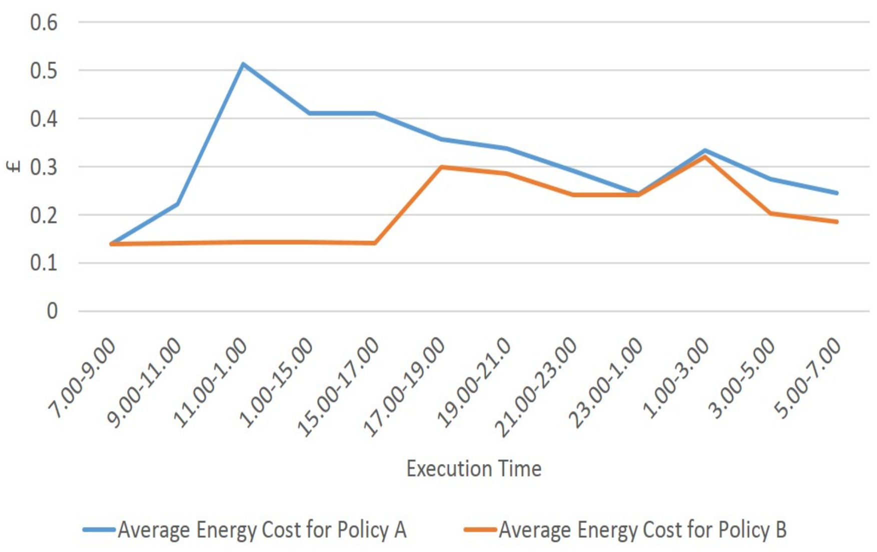


Figure 6.2 – The average of Energy Costs for Policy A and Policy B during 24 hours

models for 24 hours. Looking at the starting point for both executions, we notice that both Path A and Path B have the same amount of energy consumption which is roughly 0.8 kwph. The reason is that both the Policy A model and the Policy B model have the same number of virtual machines and similar loads at each place. However, in Path A, the amount significantly grows until reaching 1.2 kwph during the peak-time, which is from 9.00 until 11.00. In contrast, Path B has stable Estimated Energy Consumption values, which are about 0.8 kwph from 9.00 until 11.00. The increase in Energy Consumption is because Policy A allows migration at any time when values of both CPU_Usage and Violation_Rate exceed the allowed thresholds. Therefore, if the VM is moved to another host, which suddenly has an increase in its loads, this might lead to an increase in Energy Consumption. On the other hand, the migration is not permitted during peak-time in path B; therefore, the amount of Estimated Energy Consumption does not increase relatively. However, the average of Estimated Energy Consumption in Path B slightly grows between 15.00 and 17.00 because the policy allows migration after peak-time.

Figure 6.2 represents the Energy Cost for all the host places in the CPN models for

Policy A and Policy B. Looking at the starting point for both Policy A and Policy B executions, we notice that both Path A and Path B have the same Energy cost which is nearly £0.15. The reason is that both the Policy A model and the Policy B model have the same number of virtual machines and similar loads at each place between 7.00 and 9.00. After that, we notice from the graph that the Average Cost of Energy for Path A dramatically increases during peak-time. The reason for the increase in the cost value is that Policy A allows a virtual machine to migrate with no time restriction. Therefore, when the loads increase at any virtual machine in the model, the places that have the highest loads will trigger migration. As a result, the virtual machines will be allowed to move to places that might have energy costs higher than the previous places. Thus, the average of energy costs at all places in the model might increase during peak-time. On the other hand, looking at Path B, we notice that during peak-time the energy cost remains steady because the policy does not permit migration during peak-time. However, the cost grows after peak-time when migration is allowed. We notice that the Energy Cost of Path B during off-peak time is relatively close to the values of Energy Cost for Path A due to the increase in the migration times and the loads on virtual machines in both paths.

Figure 6.3 is the third graph that we produced from the model. The graph presents the amount of transmission cost while executing Policy A and Policy B. We notice from the graph that during peak-time Path B does not record any transmission cost until the end of the peak-time period. This figure is opposite to Path A figures. During peak-time, Path A has fluctuations in the transmission cost which continuously increase until reaching above £1.2 before 1.00 and then dramatically decrease between 1.00 and 15.00 to record nearly £0.5. The transmission cost at Path A after peak-time is decreasing which means that the number of places that are in the Permit Migration state at that time is lower than places in the same state during peak-time. We notice that transmission Cost figures for both Path A and Path B during off-peak time are close. The reason is that both policies

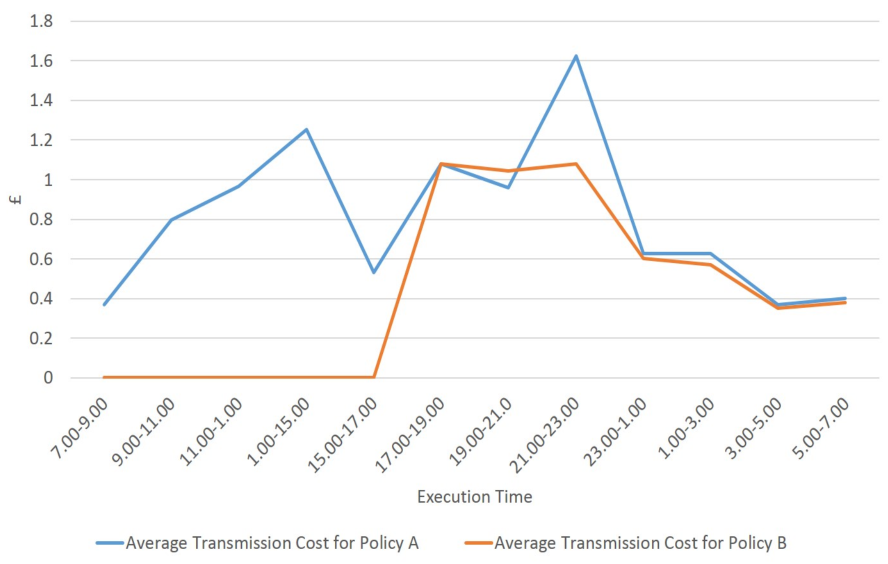
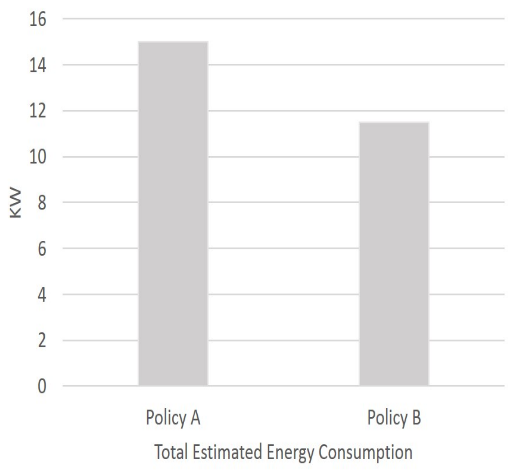


Figure 6.3 – The average of Transmission Cost for Policy A and Policy B during 24 hours

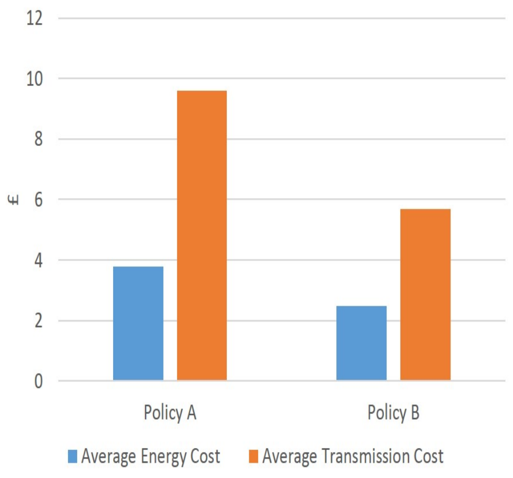
A and B allow multiple migrations to any available host during off-peak times.

Figure 6.4 has two bar charts that show the total of Average energy consumption, energy cost, and transmission cost after 24 hours from executing Policy A and Policy B. From the chart, we can clearly see that the totals of energy consumption, energy cost, and transmission cost for Policy B is less than the totals of these values for Policy A. The reduction percentage that we might achieve in both energy consumption and energy cost when the high-level policy is combined with the low-level policy is about 23.55% and 35% respectively.

From the *CPN_Cloud* modelling and analysis results, we determine that if cloud consumers or cloud-domain experts allow dynamic migration using Policy A, which implements only the low-level policy, they might consume more energy and have higher costs than using Policy B, which combines the low-level and high-level policies. Thus, the result that we achieved from CPNs can assist cloud-domain experts to assess which policy is suitable in terms of energy saving.



(a) The total amount of Estimated Energy Consumtion for both Policy A and Policy B



(b) The total Energy Cost and Transmission Cost for both Policy A and Policy B

Figure 6.4 – The total estimated energy consumption, energy cost, and transmission cost for executed management policies during 24 hours

6.3 Chapter Summary

In this chapter, we proposed a Simulation-based Cost Calculation Method (SCCM) for computing the cost from a single trace generated from simulating CPN_{Cloud} model. We applied the SCCM to compute the energy cost and migration cost associated with triggering a migration action. The method is an application of a proposed cost calculation from timed Petri-nets [15, 16]. By using the SCCM, we can obtain the total cost values from a number of traces of execution generated from simulating CPN_{Cloud} models. The outcome of the analysis approach is to provide an estimate for cost values for two different management policies and to select the appropriate one before implementation. The application of the suggested modelling and analysis approach in the Energy Management Case Study allowed us to define which management policy was better in terms of cost savings. In the next chapter, we will improve the SCCM to provide more accurate estimation by looking at two different cases that we observed while applying the SCCM.

CHAPTER 7

The Optimised Cost Calculation Method (OCCM) for Management Policies Including Time-Intervals in a Modelled Cloud Platform

In Chapter 5, we proposed a model-based approach using coloured Petri-nets (CPNs) to model both cloud platform and management policies. The proposed method is aimed at defining a formal model for an automatic cloud platform which includes a migration action as a dynamic action. The formal description model for both cloud platform and management policy is denoted as CPN_{Cloud}. Each generated CPN_{Cloud} model is simulated to produce a set of traces of execution for 24 hours. At each sampled trace, the cost calculating method for computing the costs of both energy consumption and the migration of a virtual machine is applied to assess two sets of management policies (see Chapter 6 for a detailed explanation of the method). We noticed that in some CPN_{Cloud} models that used time-intervals to fire migration transitions, the traces of executions for such models might have complex structures and might also include various loops. However, the cost calculating method proposed in the previous chapter does not consider complex

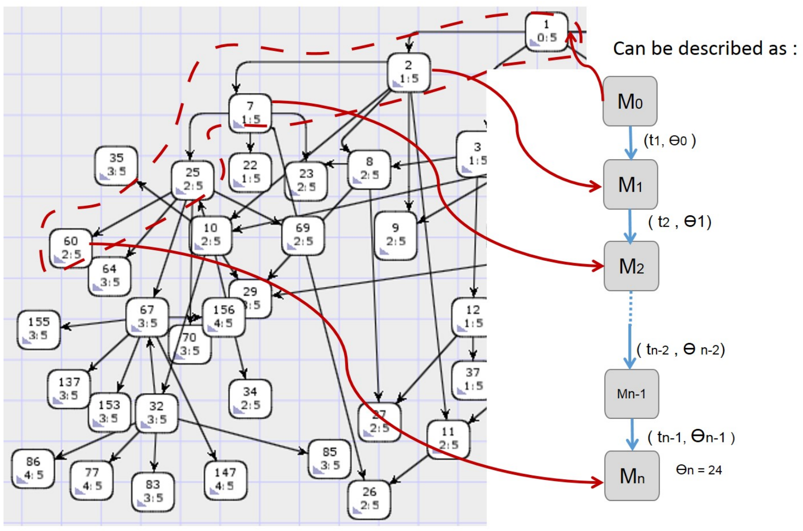


Figure 7.1 – A sample of a reachability graph (left-side) and a sample of a trace extracted (right-side)

cases.

Therefore, in this chapter, we extend our previous method to consider the time-intervals by formulating a set of integer programming equations solved by the Simplex algorithm which uses the Branch and Bound Method [8]. The purpose is to provide cost estimation values which cover nearly all possible cases that might be found in a trace generated from a CPN_{Cloud} model.

Furthermore, we solved the problem of including loop traces inside traces of execution. This can be accomplished by proposing a theory which relates the migration cost to the loop traces. The objective of this theory is to find the loop traces that would be discarded during the process of the cost calculation. Hence, this chapter is an extension of the Simulation-based Cost Calculation Method (SCCM) proposed in Chapter 6.

7.1 The Description of the Problem

To clarify the problem, which led to our extension of the previously proposed Cost Calculation Method, let us consider a partially generated reachability graph produced from a CPN_{Cloud} which is presented in Figure 7.1. In Figure 7.1, these traces represent the possible execution during 24 hours which is generated using the CPN Tool [5]. Focusing on the highlighted trace, any single trace can be described in a format as presented on the right hand side of Figure 7.1. In a trace σ, the squares represent the markings which can be either a marking resulting from triggering a monitoring transition or a marking generated from firing a migration transition. These markings contain the computed cost values for all running hosting nodes in a cloud platform. In addition, the trace σ also has arrows which are annotated with (t_i, θ_i). In this form, the notation θ_i represents the time unit for firing a transition. Formally, the trace can be described as follows:

$$\sigma := M_0 \xrightarrow{(t_0,\theta_0)} M_1 \xrightarrow{(t_1,\theta_1)} M_2 \xrightarrow{(t_2,\theta_2)} \dots \xrightarrow{(t_{n-1},\theta_{n-1})} M_n$$

$$0 \leqslant \theta_0 \leqslant \theta_1 \leqslant \theta_2 \dots \leqslant \theta_{n-1} < 24 \tag{1}$$

By applying the SCCM described in Chapter 6, we can obtain the estimated cost values of Energy Consumption and Migration Costs. However, some CPN_{Cloud} models have migration transitions restricted to time intervals, such as the model of the CPN_{Cloud} that includes timed-based policies (examples of such models are explained in the case study in Section 5.1 in Chapter 5 as well as the ones presented in Table 7.1). This means that in a trace σ, a migration transition t_i is fired with a time delay d_i which is between the allowed time-interval $[D_{Mini}, D_{Maxi}]$. The previous Cost Calculation Method relies on using a CPN simulator to randomly select the time delay for firing a migration transition. However, when using a time-interval $[D_{Mini}, D_{Maxi}]$ for firing a migration transition t_i, there might be a time delay d_i where the cost of energy consumption between the marking M_i and M_{i+1} could have minimum values. As a result, the SCCM proposed in Chapter 6, can be extended to compute the minimum energy consumption cost between the markings. This can be achieved by finding the optimal or near-optimal time delays for firing migration actions in traces associated with time-intervals. The solution will be explained in the following section.

7.2 The Optimised Cost Calculation Method (OCCM)

The solution starts by computing both cost values which can be Energy Cost and Migration Cost at each marking M_i in a trace σ using the second parts of both Equations 4 and 6 mentioned in Section 6.1 in Chapter 6. Then, we formulate a set of Integer Programming equations for obtaining the minimum energy consumption cost.

7.2.1 Computing the Overall Cost in the Trace

Let us consider that the generated trace σ has the following format:

$$\sigma := \quad M_0 \xrightarrow[d_0]{(t_0,\theta_0)} M_1 \xrightarrow[d_1]{(t_1,\theta_1)} M_2 \xrightarrow[d_2]{(t_2,\theta_2)} \ldots M_{n-l} \xrightarrow[d_{n-1}]{(t_{n-1},\theta_{n-1})} M_n$$
$$0 \leqslant \theta_0 \leqslant \theta_1 \leqslant \theta_2 \ldots \leqslant \theta_{n-1} < 24 \tag{2}$$

In which each M_i is a marking resulted from firing t_i with a delay time unit d_i. Each M_i happens at time θ_i. To compute the total cost in trace σ such that each ti is fired with time delay d_i which has the minimum $ECost(M_i)$ and an overall total cost which is less than or equal to the daily assigned budget b, we define the following objective function:

$$OverallCost(\sigma) = \quad Minimize(\sum_{i=0}^{n-1} d_i * \ ECost(M_i) + \ TCost(t_i)) \tag{3}$$

subject to the following constraints:

- $d_i * \ ECost(M_i) + \ TCost(t_i) \leqslant AllowedBudget$

$$d_i = \begin{cases} D_{Mini} \leqslant d_i \leqslant D_{Maxi} \text{ if the firing transition } t_i \text{ is a migration transition} \\ @MonitoringTime \leqslant 24 \quad \text{if the firing transition } t_i \text{ is a monitoring transition} \end{cases}$$

- $\theta_{i+1} = d_i + \theta_i \leqslant 24$
- $\theta i \ , \theta_{i+1} \ , d_i > 0, \ \ \theta_i < \theta_{i+1}$
- $\theta_0 = 0$ and $\theta_n = 24$

Such that $d_i, \theta_{i+1}, \theta_i$ are integers. In our solution, we consider computation for a day (i.e., 24 hours). Yet, the computation can be easily adjusted for days, weeks or seasons. *AllowedBudget* is the value of the maximum allowed assigned budget by a cloud domain expert for both Energy Consumption and Migration Cost. Its computation will be explained in the following section. Both D_{Mini} and D_{Maxi} are integer values extracted from the management policy used to limit the variable d_i. The variable @Monitoring-Time represents the delay-time unit for monitoring transitions which will be assigned in a CPN_{Cloud} model before extracting the traces. θ_i is the time for firing the transitions in a trace σ and θ_{i+1} is the next time unit after firing a transition t_i. In this method, we assume that each fired monitoring transition has no cost. As a result, in Equation 3, the obtained cost value for any monitoring transition will be equal to 0.

The value of $OverallCost(\sigma)$ is obtained using Simplex Solver which uses the Branch and Bound Method implemented in Excel [8]. The objective is to find the feasible integer values for time-delay d_i, since the CPN_{Cloud} model considers discrete time rather than continuous time. Using this algorithm, the total cost along σ can be computed by finding the best time-delay d_i for firing each migration transition t_i. In case the solver [8] does not find a feasible solution in a trace σ, we assign for each d_i the value D_{mini} (a brief description about Integer Programming and Branch and Bound Method is provided in

Section 2.5 in Chapter 2).

7.2.2 Computing the Assigned Budget

In Equation 3, the first constraint-equation restricts that the total cost at a marking M_i should be less than or equal a value of the $AllowedBudget$. In our CPN_{Cloud} model, we obtain this value by using the following Equation:

$$AllowedBudget = \sum_{a=1}^{n} x_a + \sum_{b=1}^{m} y_b \tag{4}$$

such that:

- $x_a =$ is the maximum cost of energy consumption for $host_a$ during 24 hours
- $y_b =$ is minimum amount of penalty caused by violating SLA

The value of each x_a is obtained from the configuration of the cost of Energy Consumption for a host which is based on the SPEC Benchmark results shown in [13]. On the other hand, the value for each y_b is extracted from the SLA Configuration for each running Virtual Machine in the CPN_{Cloud} model.

7.2.3 A Special Case: Handling Traces with Loops

In the previous section, we mentioned a method of calculating the minimum cost for a given finite trace σ. If a Petri-net results in a finite number of possible reachable markings, the method suggested above would be sufficient for obtaining the minimum amount of energy associated with a policy that has time intervals. In such cases, we need to obtain the Integer Programming equations corresponding to *all* traces and use a solver for IP such as the one in [8] on each trace to find the trace which has the minimum cost. However, it is possible that some traces involve periodic behaviours which appear in these traces as loops. As depicted in Figure 7.2, from the trace shown in the figure, infinite traces can be obtained by repeating the loop involving the markings M_2, M_3, and M_4.

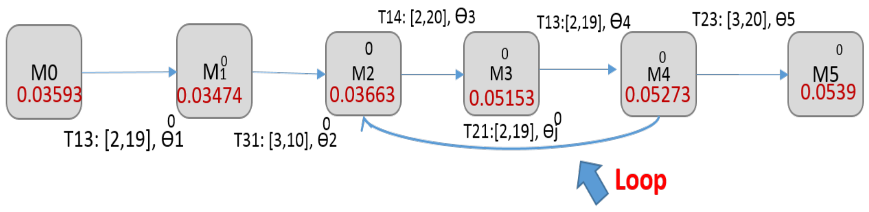


Figure 7.2 – An example of a trace with a loop

It is possible for each given number of iterations of the loop to obtain a trace and apply the method of the previous section to calculate the minimum cost. The trace would repeat the markings involved in the loop. Clearly, the length of the traces can increase as we can include an arbitrary number of repetitions of each loop. At first glance, it might be the case that we need to identify the minimum cost over an infinite number of traces. However, with each iteration on a loop, there are costs associated with the traces. This is because the migration of the virtual machine accumulates cost. For a sufficiently large number of migration costs, the trace will be large enough to be discarded from the calculation, when we are looking for a solution with a minimum cost.

Lemma 1 *Assume that σ is a trace of execution such that σ has a loop. i.e. σ has the following sub-sequence:*

$$\lambda \quad = \quad M_{k-1} \xrightarrow[d_{k-1}]{(t_{k-1},\theta_{k-1})} M_k \xrightarrow[d_k]{(t_k,\theta_k)} M_{k+1} \dots \xrightarrow[d_{l-1}]{(t_{l-1},\theta_{l-1})} M_l$$

such that $M_l = M_{k-1}$ and $Cost(\sigma) \geqslant N \times LoopCost(\lambda)$ in which N is the number of repetitions of the sequence $(*)$ and $LoopCost(\lambda) = \sum_{i=k}^{l} TCost(t_i)$

Sketch of The Proof: The amount of energy associated with λ consists of the amount of energy consumed by running virtual machines at all hosts in CPN_{Cloud} model. We mean that the cost at M_i where $k-1 \leqslant i \leqslant l-1$ plus the cost of the migration when the transitions $t_{k-1}, t_k, \dots, t_{l-1}$ are fired. $LoopCost(\lambda) = \sum_{i=k}^{l} TCost(t_i)$ captures only the cost migration. If there are N repetitions of the loop, we end up with $N \times \mathit{LoopCost}(\lambda)$ with at least the cost associated with the migration for N iterations of the loop.

Theorem 1 *Assume L represents the set of all loops with at least one migration transition with a non-negative cost. Suppose $C_{min} = \min\{\mathit{LoopCost}(\lambda) | \lambda \in L\}$ repeating the smallest value for all the cost of migration within a loop. Suppose that σ_s is an arbitrary finite trace of execution starting from the initial marking and executing for 24 hours. If q is the smallest number that $q \times C_{min} \geqslant TCost(\sigma_s)$, then the minimum cost will be*

$$Min\{Cost(\sigma) | \sigma \text{ is a trace with at most } q \text{ repetitions of each loop}\} \tag{5}$$

Proof 1 *For any trace σ with more than q repetitions of a loop L, the cost of σ will be greater than or equal to $Cost(\sigma_s)$. Hence, traces will be discarded as these will result in a minimum cost of energy.*

Using the above theorem, we need to calculate the minimum cost with the help of a finite number of traces. As a result, if any graph resulting from the CPN_{Cloud} model consists of a set of traces including loops, a finite set of traces executing the loops should be generated. This can be achieved by repeating the loops N repetitions. Then, the cost is computed using the Optimised Method explained in Section 7.2. In addition, the cost of migration for traces with loops should be considered. We stop computing the cost for traces consisting of loops when the cost values become greater than the cost of a trace with the smallest repetition number for the loop sequence. The following example will demonstrate the implication on the values of migration cost in a trace with a loop sequence.

7.2.4 An Example Demonstrating a Loop Case

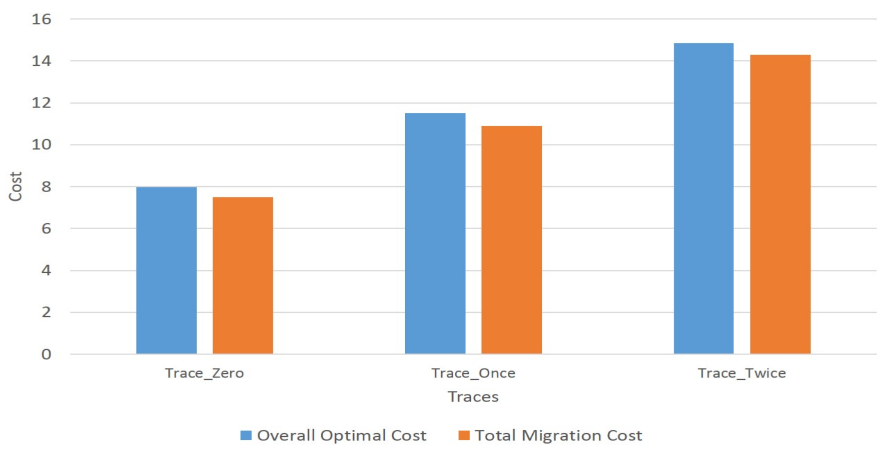
Figure 7.3 demonstrates an example of the loop case in a trace extracted from the graph of Policy A mentioned in Section 7.3 (see the graph in Figure 7.4(a)). Figure 7.3(a) presents the cost values of both Migration Cost and Overall Cost in a trace σ_1 with and without the execution of loop sub-traces. From the generated loop traces, the Cost Calculation Method explained previously is applied to these traces. From the result, the optimal total cost values for traces with loops are more than similar traces with zero repetition for the loops, due to the increase in the cost of migration for loop traces. As a result, it is preferable to discard these traces before the calculation.

By applying the theorem explained previously, a detailed analysis about the changes in the values of Migration Cost, denoted as $TCost(\sigma_1)$ during 24 hours is presented in Figure 7.3(a). In Figure 7.3(b), the value of $TCost(\sigma_{Zero})$ increases when the migration is allowed until it reaches 7.5 after 24 hours execution. One the other hand, the values for both $TCost(\sigma_{Once})$ and $TCost(\sigma_{Twice})$, which are traces allowing the execution of loops, increase to reach 10.9 and 14.3, respectively. In this case, it is noticeable that the migration costs for the loop traces are more than a trace without the execution of the loop. As a result, these loop traces will be discarded from the calculation.

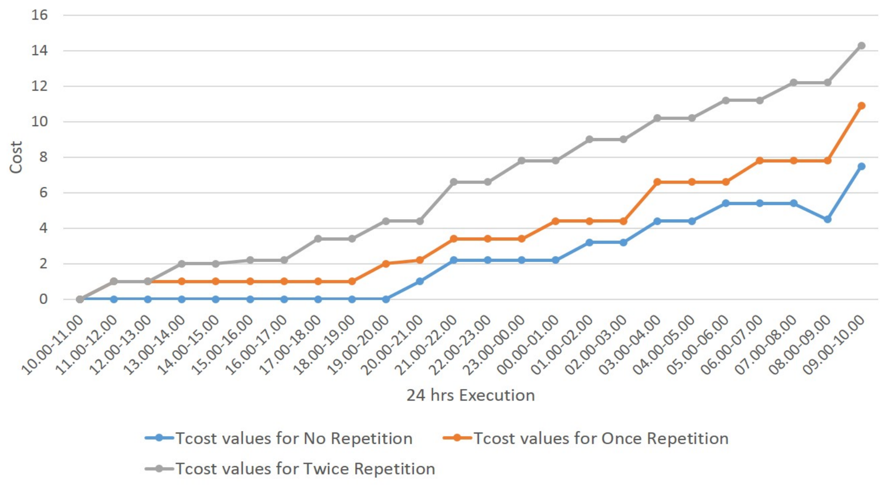
7.3 The Evaluation of the Proposed Method

We evaluated the proposed solution by analysing and comparing the CPN_{Cloud} model for the same case study described in Section 5.1 in Chapter 6 but using a different set of management policies, which are presented in Table 7.1 .

We used the CPN tool [74] to generate CPN models for all tested policies mentioned in Table 7.1. Then, we created a set of sequence of executions for each model. During the process of generating the sequence graphs using the ML function in the CPN tool, we also



(a) Comparing the overall cost values between traces with no execution of loops and with loops execution



(b) The values of $TCost(\sigma_i)$ during the execution of 24 hours with 0 , 1 ,2 repetition(s) for a loop trace

Figure 7.3 – Demonstrating the loop case using a trace of execution from CPN_{Cloud} of Policy A

Time-based Policy_A

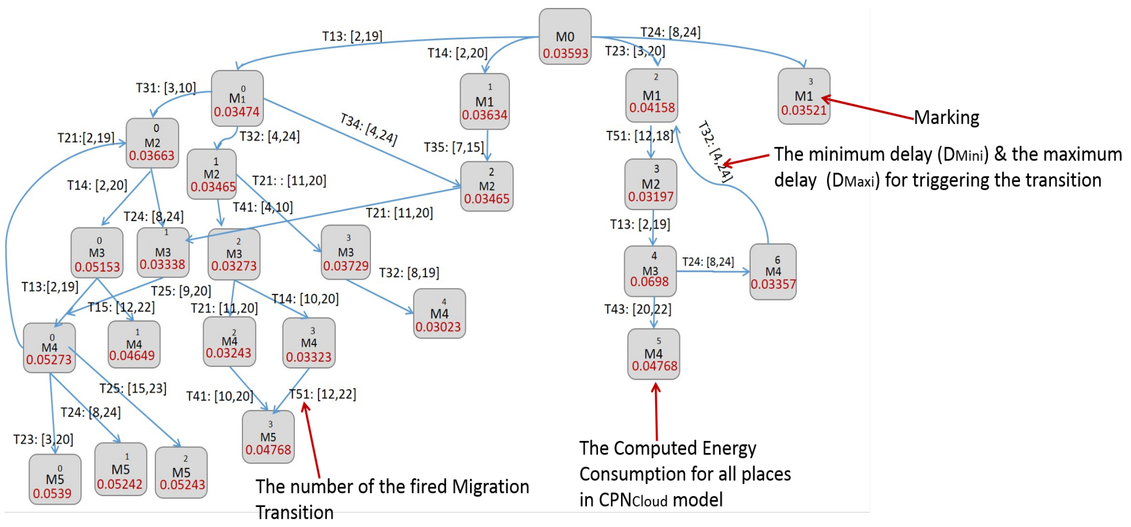
- RuleSet_1: (Applied between Private_Hosts)
  - **When** Private_Host(x) can accept Migrated VM and Time is after 10.00 and Private_Host(y) can migrate VM
    then
    Allow only one VM to be migrated between Private_Host(x) and Private_Host(y) every Δ_{time}.
- RuleSet_2:(Applied between Private_Host and Public _Host)
  - **When** Public_Host(x) can accept Migrated VM and Time is after 10.00 and Private_Host(y) can Migrate
    then
    Allow VM to be Migrated to Public_Host(x) every Δ_{time}.
- RuleSet_3: (Applied at Public _Host)
  - **When** Time is after 10.00 at Public_Host(x)
    then
    Allow VM to be Migrated to Private Hosts from Public_Host(x) every Δ_{time}.

Time-based Policy_B

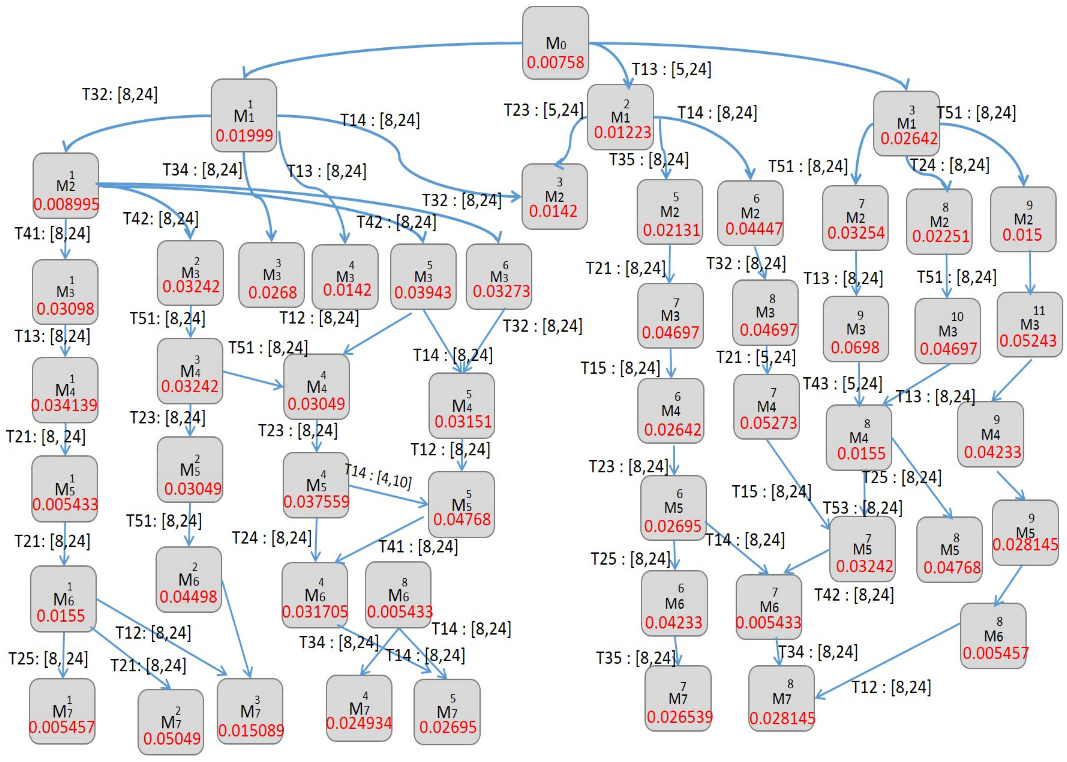
- RuleSet_4: (Applied between Private_Hosts)
  - **When** Private_Host(x) can accept Migrated VM and Time is between 16.00 and 7.00 and Private_Host(y) can migrate VM
    then
    Allow only one VM to be migrated between Private_Host(x) and Private_Host(y) every Δ_{time}.
- RuleSet_5:(Applied between Private_Host and Public _Host)
  - **When** Public_Host(x) can accept Migrated VM and Time is between 16.00 and 23.00 and Private_Host(y) can Migrate
    then
    Allow VM to be Migrated to Public_Host(x) every Δ_{time}.
- RuleSet_6: (Applied at Public _Host)
  - **When** Time is after 23.00 at Public_Host(x)
    then
    Allow VM to be Migrated to Private Hosts from Public_Host(x) every Δ_{time}.

Table 7.1 – A Rules Set Template for Expressing Time-based Rules For Both Policy_A and Policy_B

computed the cost of energy consumption and the migration cost which are associated with the markings. After that, integer programming equations were formulated and solved using the Microsoft Excel Solver [8]. Both the modelling and analysis processes were done on a Samsung laptop which has 2.40GHz Intel(R) Core(TM) processor and 6GB memory.


Marking
The minimum delay (DMini) & the maximum delay (DMaxi) for triggering the transition
The Computed Energy Consumption for all places in CPNCloud model
The number of the fired Migration Transition

(a) Policy_A: random time-intervals



(b) Policy_B: fixed time-intervals

Figure 7.4 – The graphs of sequence of execution for Coloured Petri Net models for Policy A and Policy B

7.3.1 The Sequence of Execution Graphs of the Generated Models

We used the CPN tool to create the models for Policy A and Policy B. The workload for the models was simulated to be generated randomly using the ML function. Figure 7.4 shows the traces of execution graphs for the models. In all graphs, the values inside the markings are the cost of energy consumption for four private hosts. The time intervals which applied at each migration transition are located at the edges between the markings. We notice that the graph in Figure 7.4(a) has two traces containing loop traces. For the loop traces, we applied the theorem explained in Subsection 7.2.3. We noticed that the overall cost values for each of the traces with loops are higher than similar traces without executing the loop sub-traces. This is an example of a case when we discard all the loop traces generated in the graph of Policy A.

7.3.2 Results and Discussion

From the graphs shown in Figure 7.4, there are 18 traces generated from the graphs of Policy A and Policy B. We applied the Cost Calculation Method explained in Section 7.2 to all generated traces. We obtained the minimum cost of Energy Consumption, Total Migration Cost and Optimal Overall Cost at each generated trace for all the graphs. To compare both Energy Consumption and The Overall Cost for all the policies, we selected 18 traces from each generated graph ignoring all the loop traces. The detailed results produced from the analysis method are displayed in Figure 7.5 and Figure 7.6.

Figure 7.5 illustrates the Optimal Energy Consumption Cost for 18 traces generated from each of the traces of execution graphs presented in Figure 7.4. Generally, the figures show that each trace for Policy B has an Optimised Energy Consumption Cost value which is less than the traces in Policy A, since Policy A allows the migration to be triggered at

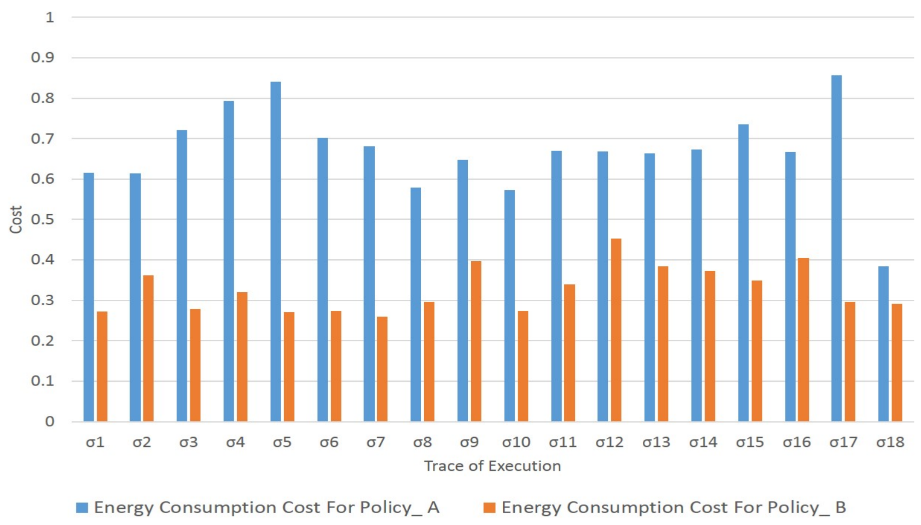


Figure 7.5 – The optimal energy consumption cost for 18 traces from the sequence of execution graphs

any time using random time intervals. In contrast, the migration action in Policy B is restricted to the off-peak time which is from 16:00 until 21:00 mapped as [8-24] in some of the traces of the graph of Policy B (see Figure7.4(b)) .

From Figure 7.5, we can analyse the cost of each trace for each policy individually. For instance, we can see that trace σ_{18} has the lowest Optimised Energy Consumption Cost among traces of Policy A. Whilst trace σ_7 has the least Optimised Energy Consumption Cost among the traces of Policy B which is nearly 0.26. In addition, we can notice that some traces of Policy A have nearly the same cost values which are reasonably high such as σ_{17}, σ_5, σ_4 and σ_3. Conversely, if we look at traces of Policy B, we can also notice some fluctuation in Energy Cost values. Trace σ_{12} has the highest cost among other traces in Policy B.

Figure 7.6 presents the Optimal Overall Cost after accumulating the Migration Cost values of each trace to its Optimised Energy Consumption Cost. We found that there were changes in cost values since some traces required the triggering of migration transitions which means their migration costs were high. As a result, the figures for each trace of

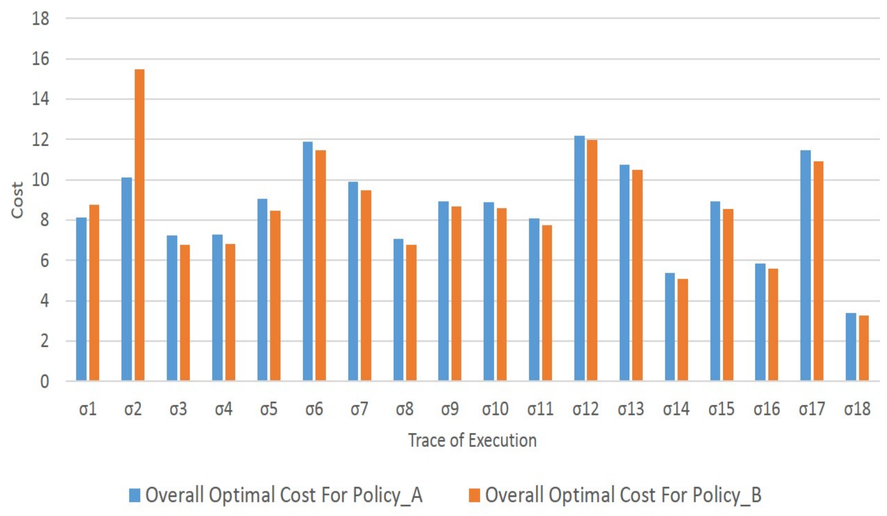


Figure 7.6 – The optimal overall cost for 18 traces from the sequence of execution graphs

execution for both Policy A and Policy B are roughly similar. The average of the Optimal Overall Cost value for both Policy A and Policy B was approximately £9.0.

To summarise, the analysis of the results using the OCCM for calculating the cost traces generated from the CPN models of Policy A and Policy B revealed that Policy B has the Optimised Energy Consumption Cost, but has an Overall Cost value close to the Overall Cost value of Policy A. Thus, restricting the firing of the migration action to be only allowed during off-peak periods, which is applied in Policy B, might save the cost of Energy Consumption, but the migration cost should be considered as an important factor. As a result, in a CPN_{Cloud} model, the migration in Policy B should be applied to places which have the lowest Migration Cost.

Comparing the OCCM Method with the Previous Method

To study the effectiveness of the proposed method on the computed cost values, we compared this method with SCCM. For all traces of both Policy A and Policy B graphs, we applied the previously proposed method explained in Chapter 6. During this process, we ignored the Migration Cost values computed from both methods because these values are

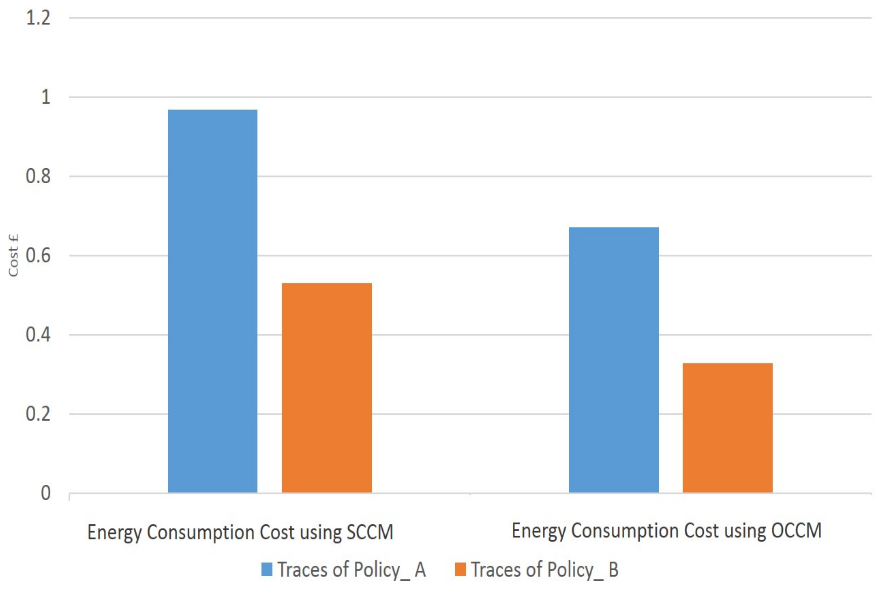


Figure 7.7 – A comparison between the two methods of calculating the cost energy consumption in traces

fixed. Instead, we focused on the computed Energy Consumption Cost values from both methods. We found that the OCCM proposed in this Chapter provides a better estimation of Energy Consumption Cost than the SCCM mentioned in Chapter 6 when time-delays are considered (see Figure 7.7). Figure 7.7 presents the effect on the estimation of Energy Consumption Cost in all traces generated from both the SCCM and the OCCM. Clearly, the OCCM produces averages of Energy Consumption Cost for both Policy A and Policy B which are lower than the values given by the SCCM. The differences in the average rate in cost values for Policy A and Policy B are similar, about 0.3 and 0.2, respectively. Thus, we can summarise that the OCCM is more accurate in estimating the cost of Energy Consumption which can be suitable for analysing some types of management policies. These management policies are the those that include time-intervals for firing migration actions in a CPN_{Cloud} model. However, if the objective is to speed up the process of calculating the cost, or if time delays are not of much concern to cloud domain experts, then the SCCM can be applicable in such types of management policies.

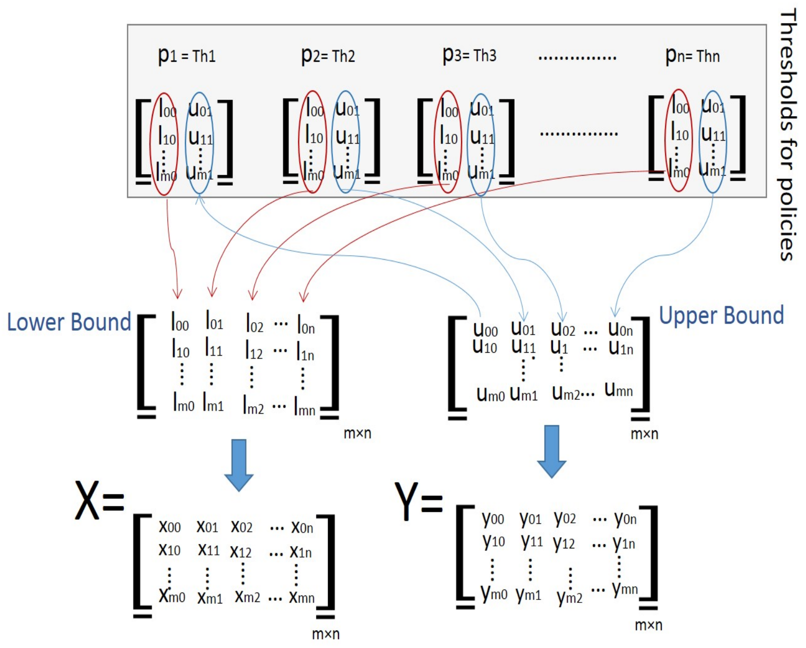
7.4 Toward Finding the Optimal Energy Strategy

As explained in Section 4.2.1, management policies are low-level, or high-level, or combination of both. CPN_{Cloud} models and the Cost Calculation methods covered in this thesis aim at evaluating the expected to be implemented policies. Using the suggested method cloud domain experts and rule developers will estimate the cost of running policies. At the current stage, the proposed analysis methods in both Chapter 6 and Chapter 7 are too far away to be used for finding the optimal policies. However, in this section, we suggested a method which might allow the tool be beneficial for fining the Optimal Management Policies in future. This can be done based on having many previously tested Management Policies which are implemented using the CPN_{Cloud} models. Literally, such policies run using a set of required threshold values. Currently, these threshold values are assigned subjectively. The thresholds used in our examples are Energy Consumption and CPU_Usage. Nevertheless, depending on the management requirements, which are specified by cloud domain experts on earlier design stage, other types of thresholds can be defined. We noticed that obtaining a new optimal Management Policy requires extracting optimal threshold values after modelling and testing various set of Management Policies. This will be explained in the following section.

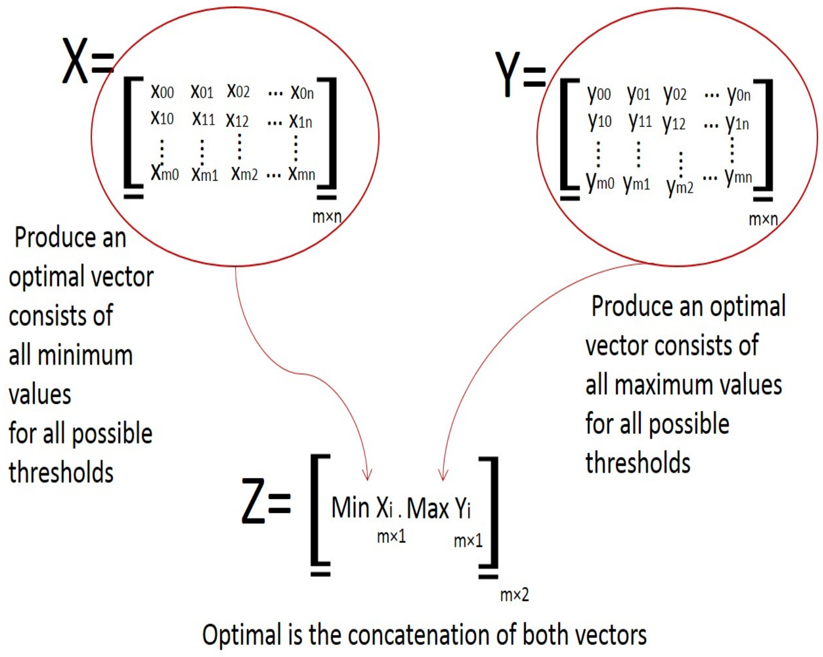
7.4.1 Obtaining the Optimal Thresholds

Let us denote to the Optimal Management Policy as Opt_{Policy}. The previous policies belong to a set of $P = p_1, p_2, \ldots, p_n$ (See Figure 7.8(a)). Each $p_i \in P$ has a matrix Th_i of size $m \times 2$ for a general representation. Here, this matrix represents the lower and the upper bound for both Energy Consumption and CPU_Usage thresholds which is of size 2×2.

To find the Optimal Policy Opt_{Policy} which will be solved using Equation 3, firstly



(a) Constructing Lower-bound & Upper-bound threshods-matrices



X=
Y=
m×n
Produce an optimal vector consists of all minimum values for all possible thresholds
Produce an optimal vector consists of all maximum values for all possible thresholds
Z= [Min Xi . Max Yi]
m×1
m×2
Optimal is the concatenation of both vectors

(b) Getting the optimal thresholds

Figure 7.8 – Extracting the optimal thresholds for the optimal Management Policy

we should find the optimal thresholds Z_{opt} for such policy.

$$Z_{opt} = \quad (Min\ X, Max\ Y) \tag{6}$$

Such that $Min\ X$ and $Max\ Y$ are the vectors of size m. Those vectors result from applying a search optimal method applied to two thresholds matrices which are X and Y. As shown in Figure 7.8(b) X is the matrix that includes the lower-bound values tested in previous modelled Management Policies which is of size $m \times n$. Whilst Y is the matrix containing the values of all the upper bound of the previous tested policies which is also $m \times n$ (See Figure 7.8(a) for constructing both X and Y matrices). Our objective is to get Z_{opt} which is the concatenation matrix that represents the optimal thresholds matrix for the potential Management Policies. Firstly, we optimise the lower-bound matrix X to get all possible minimum values which is the first part in Equation 6 on the right hand side. The minimum part is subject to the following constraints:

- $aj \leqslant x_i \leqslant bj$. aj is the lowest value of lower bound of the tested $threshold_j$ value among the previously tested policies and bj is the highest value of the lower bound of the tested $threshold_j$ for the previously modelled policies.

On the other hand, the second part of Equation 6 is solved by optimising the upper thresholds matrix for getting the maximum values which will be subject to the following constraints:

- $cj \leqslant y_i \leqslant dj$. cj is the lowest value of upper bound of the tested $threshold_j$ value among the previously tested policies and dj is the highest value of upper bound of the tested $threshold_j$ for the previously modelled policies.

Equation 6 has the properties of linear programming which can be solved using Modified Simplex or any multi objective optimisation algorithm. This requires an investigation of a suitable optimisation algorithm which can be a future objective. We want to refer that the optimal policy can be found or not which is based on having adequate previously tested Management Policies. In cases where the optimal policies cannot be found, we can only depend on using the CPN_{Cloud} model for evaluating the currently executed Management Policies.

7.4.2 Using the Optimal Analysis Method

After selecting the possible optimal thresholds as explained in previous section, the optimal Energy strategy can be obtained using our modelling and analysis method suggested in Section 7.2. The optimality is based on the assigned thresholds given at each suggested management policy. After the thresholds which are related to the Energy Consumption and Resources parameters have been defined, the CPN_{Cloud} model for the suggested optimal management policy is created. For the created new model, the analysis method suggested in Section 7.2 is applied. The outcome of the method is the overall cost value for both Energy Cost and Migration Cost which should be compared with other estimated cost values obtained for previously applied policies if the new derived policy is the optimal policy.

7.4.3 The Advantages and Disadvantages of Modelling and Analysis Approach

As explained in Chapter 4 that Management policies are classified with two levels. Management Policies can be used to solve various types of management problems that require triggering a set of management actions. We designed an extensible meta-model which can be defined by cloud domain experts based on their own specified management requirements. For example, the model that can be used to address management objective depends on the business model suggested by cloud domain experts. However, our modelling process is restricted to management actions that require the migration for applications, services, virtual machines and physical nodes. Nevertheless, any management objective such as a reconfiguration for services, which can be done inside the physical host, can be covered and studied using the analysis method suggested in Chapter 6.

One of the advantages of using our modelling approach is related to mathematical formalism. The formalism is simple and enriched with features as well as an adequate information for modelling a cloud-platform. The characteristic of such a platform is composed of a manager that should trigger various sets of management actions. An other advantage of our modelling approach is related to the cost calculation method. Both methods suggested in Chapter 6 and Chapter 7 are for evaluating various set of predefined Management Policies. Our method is an analytical approach which covers all possible options and configurations that can be found. In contrast to other existing approaches discussed in Section 2.7 in Chapter 2, such evaluation methods are based on a simulation which might miss some existing options. Furthermore, it should be noted that our analysis technique is based on mathematical formalism which can be extended to model various types of monitoring parameters. This can only be accomplished by describing the measurement method inside ML-function applied in to CPN_{Cloud} tool. Opposite to existing approaches, cost calculation is based on random selected thresholds. As a result, our approach provides a deep analysis for cost assoicated with triggering management actions.

We want to point out that our proposed modelling and analysis tool requires learning and background about modelling and Petri-nets which can be considered as a drawback. Furthermore, compared to optimisation approaches suggested in Chapter 2 in Section 2.7, our method is limited to a platform that should trigger management actions and should require in corporation of other nodes. Therefore, the modelling concept should be understood in order to apply different types of management objective for using various domain of management policies as introduced at the beginning of Chapter 7. In the following chapter, the limitations of the suggested modelling and the analysis methods are discussed.

7.5 Chapter Summary

Management policies that would be executed in a cloud platform can be assessed in terms of energy cost saving before execution via CPNs. By using the SCCM and the OCCM suggested in this chapter, the estimated energy consumption and the migration costs can be obtained. The OCCM provides a deep analysis for both cost values which are extracted from the traces of the execution graph of CPN_{Cloud} models. The method uses a set of integer programming equations which are solved via the simplex algorithm with Branch and Bound Method provided in Excel Solver. The objective is to find traces which have minimum energy cost values and the best time for firing migration actions during 24 hours. CPNs can be powerful tools for modelling and analysing autonomic cloud platforms and management policies. Both the Simulation-based and the Optimised Cost Calculation methods will allow cloud domain experts to study the effect of executing a policy in a CPN_{Cloud} on cost values before implementing them in a real platform.

CHAPTER 8

Conclusion and Future Work

The efficient management of energy consumption is one of the methods that assists the data centres' owners to reduce the electricity usage of their data centres. The implication of applying such a method might result in the reduction of CO2 emissions [31]. Due to the motivation toward designing energy-efficient data centres, both cloud providers and cloud consumers (organisation or enterprises) have *green policies*, and have their own strategic plan for implementing such policies which can follow the suggested recommendations as in [2, 101, 129]. One type of green policy is an energy management policy, which is related to managing running services in data centres. In our research context, we refer to “energy management policies” as “management policies”. Cloud providers have their own implementations and algorithms for management of their services considering energy consumption. On the other hand, cloud consumers would like to have a method that allows them to describe and implement their “management policies” automatically. Thus, the objective of this PhD thesis is to bridge the gab between the level of describing and implementing for these types of policy.

Throughout this PhD thesis, we studied management policies as a concept for automatically governing energy consumption and energy cost in a cloud environment. Management policies originate from a management objective established by cloud domain experts. The *executable* forms of specified management policies trigger management actions automatically when they are implemented into *MP-Framework*. The architecture *MP-Framework* is based on using a policy-based engine. The framework can be configured with any cloud management system via developing the correct connecting wrappers. To illustrate the applicability of *MP-Framework*, we applied the framework to an Energy Management Case study for a private cloud scenario implemented in Drools [11] and OpenNebula [105].

We classified management policies into low-level and high-level policies. Chapter 4 presented the design specification that can be applied to formulate various types of management policies. Originally, this specification was based on the specification of the UML-Rule Modelling Language (URML). Using the specification discussed in Chapter 4, both the *expressible* and the *executable* forms of management policies can be formulated. The specification that we proposed are used to design the CloudMPL language, which is a domain-specific language consisting of textual expressions. The purpose of CloudMPL is to describe management policies during the early design stages. Furthermore, the proposed specification was utilised to develop a set of mapping rules from CloudMPL to Drools. The purpose of designing such mapping rules was to build the foundation of code-generation from the description level to the implementation level during the development of management policy.

Due to the size and dynamics of the cloud platform, we noticed that the interactions of the *executable* management policies may be very complex during execution in a real cloud platform. Therefore, we saw a need to create a method for assessing the cost of executing various sets of management policies before their implementation in the platform. Hence, in Chapter 5, both cloud platform and management policies are modelled using Coloured Petri-nets (CPNs). The modelling approach that we suggested considering the case of cloud platform allows a live migration action for a virtual machine among a set of available hosts. The novel achievement in Chapter 5 is the ability to create a modelling tool for such a cloud platform. The structure of the cloud platform results in interactions with complex components. Such components have massive amount of information that must be included in the cloud models. Therefore, CPN was a suitable modelling language that can handle the complex structure of the cloud platform and management policies. Each generated CPN_{Cloud} model presents a cloud platform and the potential management policy in a convenient manner as well as in a clear abstract form.

Using CPN_{Cloud} models, we were able either to simulate or to generate a set of traces of executions. From each set of traces produced, we managed to compute the cost values using two proposed methods for cost calculation. Those approaches are the Simulation-based Cost Calculation (SCCM) and the Optimised Cost Calculation (OCCM) methods. Using the overall cost values obtained from SCCM or OCCM methods, cloud domain experts and rule developers are able to estimate energy consumption and migration costs of various sets of management policies before real implementation into a cloud platform. In our thesis, we are concerned about the energy consumption cost and the migration cost. However, any cost model can be encoded in CPN_{Cloud} models and computed using the suggested cost calculation methods.

In conclusion, management policies can be described in either *expressed* or *executable* forms using the specification proposed in Chapter 4. Furthermore, the *executable* form of management policies can be directly executed in a cloud platform using *MP-Framework*. Due to the complex interactions that might arise during the execution of the management policies, Chapters 5, 6 and 7 suggest CPN modelling and analysis methods. Both the cloud platform and management policies can be assessed in terms of energy savings. Although the concept of management policies is covered in various aspects in this research, this study has some limitations which will be explained in the following section.

8.1 Research Limitations

MP-Framework is designed to be run with a cloud management system that has a centralised architecture such as OpenNebula. Referring to the challenges briefly discussed in Section 3.5 in Chapter 3, the *MP-Framework* is limited to the scope of small scale data centres. For medium and large size data centres, the framework can be duplicated as explained in Chapter 3. In addition, in our thesis, we built the foundation for transforming a management policy written at description level to the implementation level using the designed mapping rules from CloudMPL to Drools. Since the full CloudMPL tool should be finished by CloudMPL designers, the transformation is not complete. Furthermore, the CPN_{Cloud} model is related to the proposed *MP-Framework*. Because of the centralised architecture that *MP-Framework* may applied to, CPN_{Cloud} is used to evaluate a cloud platform that has a centralised architecture.

The scalability of the CPN_{Cloud} model is important. Since our modelling and analysis approach is applied to a centralised cloud architecture, the model is limited to configure a certain number of operating nodes. Any increase in the number of nodes in the model, the graphical representation of whole model will be become complex and difficult to manage. This requires an enhancement in the formalism of CPN_{Cloud} model to include various architecture types as will be explained in the following section. Another interesting angle is related to the scalability of the proposed analysis method. As shown in Chapter 6 and Chapter 7, the proposed cost calculation methods are used to estimate the cost values for days, months and years. From our observation that the produced traces which include a large window size, such as months or years, are larger than the ones produced for days. The appearance of such an issue might have an effect on the speed of the process of analysis. Due to the research time, we did not cover the applicability of the cost calculation method to include a larger window size. It requires having more tests to judge the best analysis speed for obtaining the cost values from the models which is considered a limitation. Such a limitation can be addressed in future, as will be explained in the following section.

8.2 Future Research

The objective of designing *MP-Framework* was to create a system that can be easily integrated and configured into a cloud platform (as explained in Chapter 3). *MP-Framework* can be enhanced with some components that provide flexibility in scaling to a semi-decentralised or decentralised environment. Therefore, a recommended path for future research is to explore organising the policy-rule engine in a hierarchical model or employing *MP-Framework* with multi-agent system [52] capabilities.

The existing solutions for cloud management such as HP Helion Eucalyptus [67], and OpenNebula [105] can be deployed into cloud data centres using various deployment models suggested by IT operators. Another possible project would study the adaptation of *MP-Framework* for management energy consumption to various deployment architectures for OpenNebula in data centres.

The transformation process from the CloudMPL language to Drools, which was discussed in Chapter 4, can be fully implemented by creating a tool that allows a cloud domain expert to write a set of management policies in CloudMPL. Then, a suitable Drools code is produced automatically. To do so, a programmer must use the designed CloudMPL mapping rules discussed in Section 4.4.2 in Chapter 4. Thus, designing full packaging tools for authoring CloudMPL and publishing management policies written in Drools are recommended aspects for future study. The existence of such tools will reduce the development and maintenance time for designing management policies.

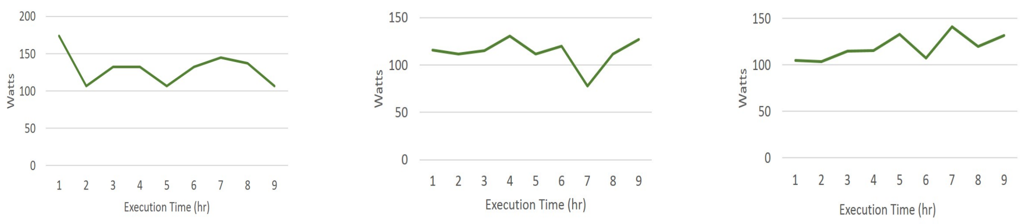
Another interesting direction for future investigation is to empower CPN models for a cloud platform with complex features. Such features will allow the models to match the decentralised architecture of a cloud platform. Future researchers might explore the concept of Hierarchical Coloured Petri-Nets [74]. Furthermore, the cost estimation values, which are obtained from both the proposed cost calculation methods suggested in Chapter 6 and Chapter 7, can be used as input values for the executable *MP-Framework*. Therefore, using values generated from an off-line model can improve *MP-Framework* with a self-management feature, which also has a potential for future work.

To address the scalability of CPN_{Cloud} model, CPN modelling tool is employed with the concept of Hierarchical Coloured Petri-Nets [74] as mentioned previously. The model will become scalable if the places and the inner transitions are hidden inside inner models. Applying the Hierarchical Coloured Petri-Nets [74] would also improve the visual representation of the model. As a future step, it requires extending the main formalism of CPN_{Cloud} to include the Hierarchical Modelling concept. Another dimension related to improving the scalability of our approach is to enhance the Optimised Cost Calculation method explained in Chapter 7. The enhancement objective is to reduce the number of markings that might be found in a single trace generated after executing the model. The reduction in the number of the markings will make the cost calculation method explained in Chapter 7 have a larger window size such as months and years. We are planning to investigate the concept *Marking Encapsulating*. The idea is to combine the calculation for a smaller window size (days) in a single marking. The larger window size will be the main marking in the method. This concept can also be applied to the hierarchical modified models. Such ideas are worthy investigating as in future research designed to tackle the scalability problems for our proposed modelling approach.

APPENDIX A

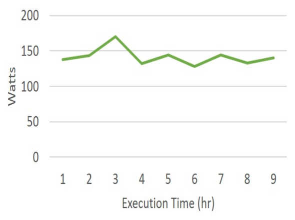
The Detailed Energy Consumption Results

The following is a detailed representation for the amount of Energy consumed by each node in OpenNebula and Drools Platform used during testing *Management Policies* mentioned in Table 3.1 in Chapter 3.

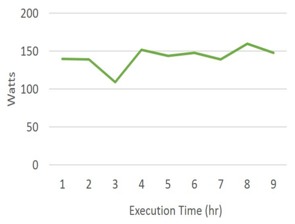


(a) The Energy Consumption at Node 1 (b) The Energy Consumption at Node 2 (c) The Energy Consumtion at Node 3

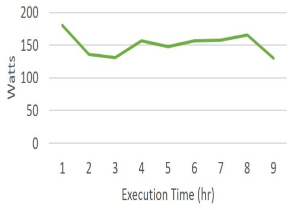
Figure A.1 – The average amount of Energy Consumption at each Node in OpenNebula and Drools Testbed during the execution of Management Policy A for nine hours



(a) The Energy Consumtion at Node 1

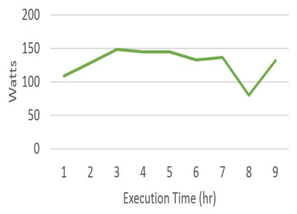


(b) The Energy Consumtion at Node 2

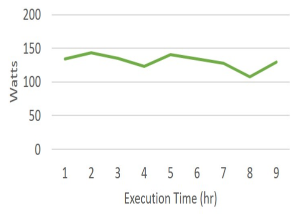


(c) The Energy Consumtion at Node 3

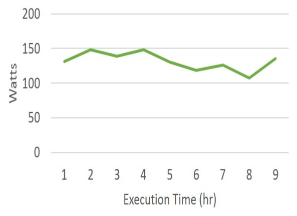
Figure A.2 – The average amount of Energy Consumption at each Node during the executing of Management Policy B in OpenNebula and Drools Testbed for nine hours



(a) The Energy Consumtion at Node 1

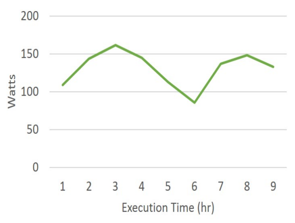


(b) The Energy Consumtion at Node 2

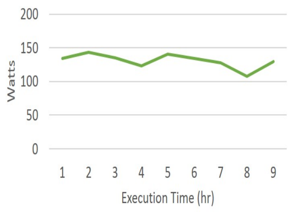


(c) The Energy Consumtion at Node 3

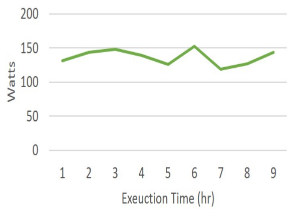
Figure A.3 – The average amount of Energy Consumption at each node during the executing of Management Policy C in OpenNebula and Drools Testbed for nine hours



(a) The Energy Consumtion at Node 1



(b) The Energy Consumtion at Node 2



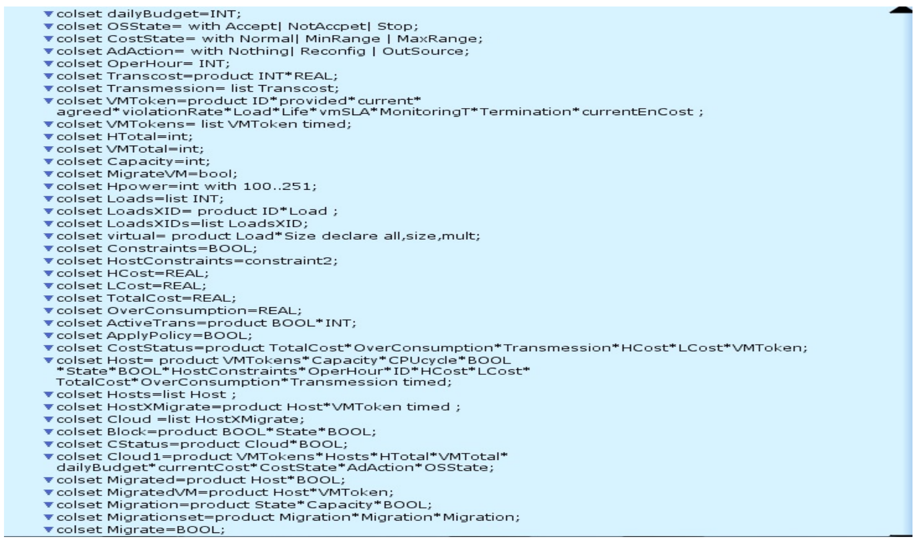
(c) The Energy Consumtion at Node 3

Figure A.4 – The average amount of Energy Consumption at each node during the exeuction of Management Policy D in OpenNebula and Drools Testbed for nine hours

APPENDIX B

A Sample of Implementation of CPN_{Cloud} models in CPN Tool

B.1 The Used Colour Sets



```
▼colset dailyBudget=INT;
▼colset OSState= with Accept| NotAccpet| Stop;
▼colset CostState= with Normal| MinRange | MaxRange;
▼colset AdAction= with Nothing| Reconfig | OutSource;
▼colset OperHour= INT;
▼colset Transcost=product INT*REAL;
▼colset Transmession= list Transcost;
▼colset VMToken=product ID*provided*current*
 agreed*violationRate*Load*Life*vmSLA*MonitoringT*Termination*currentEnCost ;
▼colset VMTokens= list VMToken timed;
▼colset HTotal=int;
▼colset VMTotal=int;
▼colset Capacity=int;
▼colset MigrateVM=bool;
▼colset Hpower=int with 100..251;
▼colset Loads=list INT;
▼colset LoadsXID= product ID*Load ;
▼colset LoadsXIDs=list LoadsXID;
▼colset virtual= product Load*Size declare all,size,mult;
▼colset Constraints=BOOL;
▼colset HostConstraints=constraint2;
▼colset HCost=REAL;
▼colset LCost=REAL;
▼colset TotalCost=REAL;
▼colset OverConsumption=REAL;
▼colset ActiveTrans=product BOOL*INT;
▼colset ApplyPolicy=BOOL;
▼colset CostStatus=product TotalCost*OverConsumption*Transmession*HCost*LCost*VMToken;
▼colset Host= product VMTokens*Capacity*CPUcycle*BOOL
 *State*BOOL*HostConstraints*OperHour*ID*HCost*LCost*
 TotalCost*OverConsumption*Transmession timed;
▼colset Hosts=list Host ;
▼colset HostXMigrate=product Host*VMToken timed ;
▼colset Cloud =list HostXMigrate;
▼colset Block=product BOOL*State*BOOL;
▼colset CStatus=product Cloud*BOOL;
▼colset Cloud1=product VMTokens*Hosts*HTotal*VMTotal*
 dailyBudget*currentCost*CostState*AdAction*OSState;
▼colset Migrated=product Host*BOOL;
▼colset MigratedVM=product Host*VMToken;
▼colset Migration=product State*Capacity*BOOL;
▼colset Migrationset=product Migration*Migration*Migration;
▼colset Migrate=BOOL;
```

Figure B.1 – A sample of the declaration of the colour set of Cloud Platform and Policies used in CPN Tool

B.2 The A Sample of CPN_{Cloud} Model

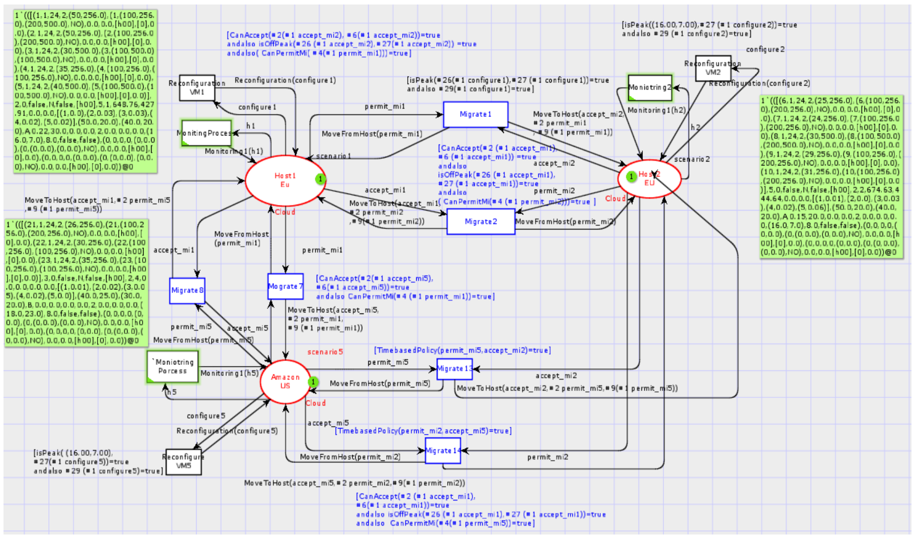


Figure B.2 – A sample of CPN_{Cloud} model used in CPN Tool for three hosting nodes

In Figure B.2, the green squares represent the initial markings. The high-level policies are the guards written above the migration transitions. The values above the arcs or the arrows are the information of the migrated virtual machine as well as the required values to change the state of hosts before and after moving virtual machines.

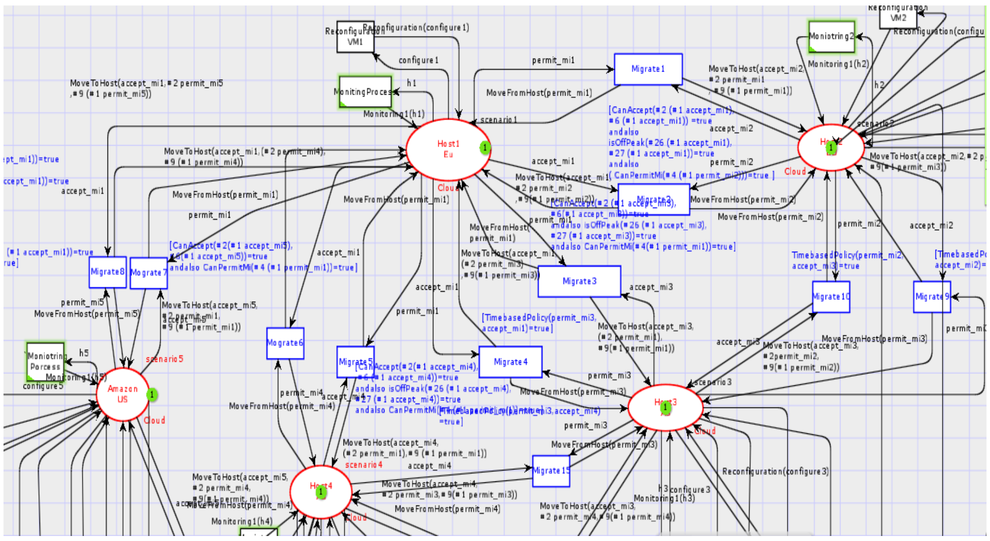
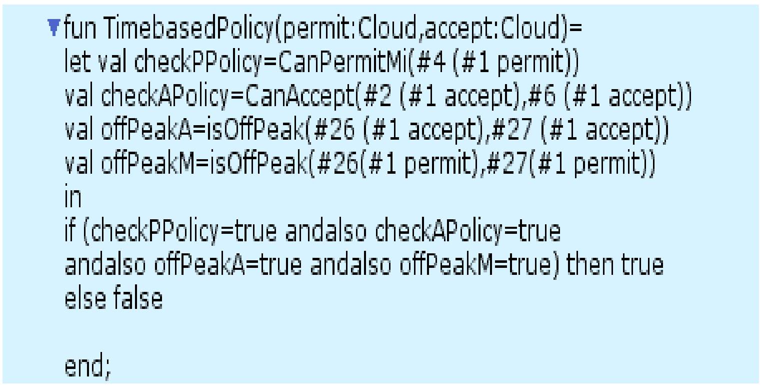
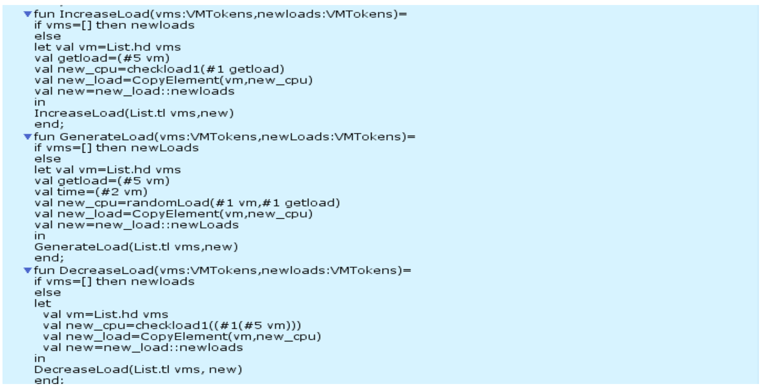


Figure B.3 – A sample of one of CPN_{Cloud} model used in CPN Tool used for the case study explained in Chapter 6



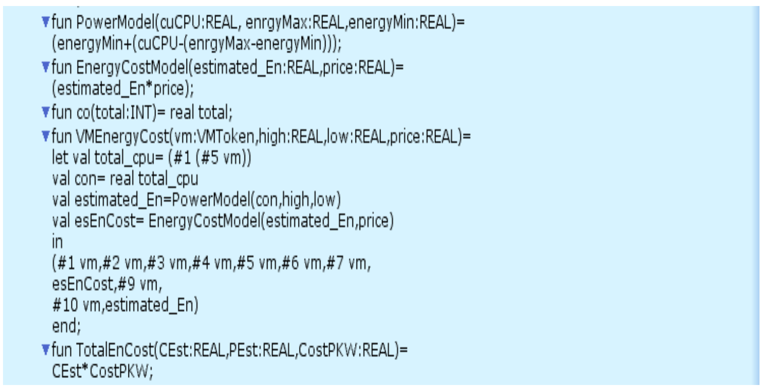
```
fun TimebasedPolicy(permit:Cloud,accept:Cloud)=
let val checkPPolicy=CanPermitMi(#4 (#1 permit))
val checkAPolicy=CanAccept(#2 (#1 accept),#6 (#1 accept))
val offPeakA=isOffPeak(#26 (#1 accept),#27 (#1 accept))
val offPeakM=isOffPeak(#26(#1 permit),#27(#1 permit))
in
if (checkPPolicy=true andalso checkAPolicy=true
andalso offPeakA=true andalso offPeakM=true) then true
else false

end;
```

Figure B.4 – A sample of one of the implemented time-based policy in CPN Tool



```
fun IncreaseLoad(vms:VMTokens,newloads:VMTokens)=
if vms=[] then newloads
else
let val vm=List.hd vms
val getload=(#5 vm)
val new_cpu=checkload1(#1 getload)
val new_load=CopyElement(vm,new_cpu)
val new=new_load::newloads
in
IncreaseLoad(List.tl vms,new)
end;
fun GenerateLoad(vms:VMTokens,newLoads:VMTokens)=
if vms=[] then newLoads
else
let val vm=List.hd vms
val getload=(#5 vm)
val time=(#2 vm)
val new_cpu=randomLoad(#1 vm,#1 getload)
val new_load=CopyElement(vm,new_cpu)
val new=new_load::newLoads
in
GenerateLoad(List.tl vms,new)
end;
fun DecreaseLoad(vms:VMTokens,newloads:VMTokens)=
if vms=[] then newloads
else
let
  val vm=List.hd vms
  val new_cpu=checkload1((#1(#5 vm)))
  val new_load=CopyElement(vm,new_cpu)
  val new=new_load::newloads
in
DecreaseLoad(List.tl vms, new)
end;
```

Figure B.5 – The ML functions for the load generator used in CPN_{Cloud} model



```
▼fun PowerModel(cuCPU:REAL, enrgyMax:REAL,energyMin:REAL)=
  (energyMin+(cuCPU-(enrgyMax-energyMin)));
▼fun EnergyCostModel(estimated_En:REAL,price:REAL)=
  (estimated_En*price);
▼fun co(total:INT)= real total;
▼fun VMEnergyCost(vm:VMToken,high:REAL,low:REAL,price:REAL)=
  let val total_cpu= (#1 (#5 vm))
  val con= real total_cpu
  val estimated_En=PowerModel(con,high,low)
  val esEnCost= EnergyCostModel(estimated_En,price)
  in
  (#1 vm,#2 vm,#3 vm,#4 vm,#5 vm,#6 vm,#7 vm,
  esEnCost,#9 vm,
  #10 vm,estimated_En)
  end;
▼fun TotalEnCost(CEst:REAL,PEst:REAL,CostPKW:REAL)=
  CEst*CostPKW;
```

Figure B.6 – The ML function for the power model used in CPN_{Cloud} model

Bibliography

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2010). 'A view of cloud computing'. volume 53, pages 50–58, New York, NY, USA. ACM. No citations.

[2] Telecommunication Standarization Sector of ITU. (2012). *Methodology for energy consumption and greenhouse gas emissions impact assessment of information and communication technologies in organizations* . [Online] Available from: http://www.itu.int/rec/T-REC-L.1420-201202-I/en.[Accessed: 7th December 2015]. 2 citations in sections 1 and 8.

[3] Object Management Group (OMG) (2008). *Business process model and notation.* [Online] Available from: http://www.omg.org/bpmn/index.htm.[Accessed: 17th September 2015]. One citation in section 2.8.

[4] Object Management Group (OMG) (2008). *Semantics of business vocabulary and business rules specification SBVBR.* [Online] Available from: http://www.omg.org/spec/SBVR/1.0/PDF/.[Accessed: 17th September 2015]. One citation in section 2.8.

[5] CPN Group (2010). *CPN Tools.* [Online] Available from: http://cpntools.org/, [Accessed: 28th September 2015]. 4 citations in sections 2.4.3, 2.4.4, 2.4.5, and 7.1.

[6] RuleML Wiki (2012). *Simple rule markUp language SRML.* [Online] Available from: http://ruleml.org/.[Accessed: 17th September 2015]. One citation in section 2.8.

[7] Stress Project Team (2014). *Stress.* [Online] Available from: http://people.seas.harvard.edu/~apw/stress/.[Accessed: 18th September 2012]. 2 citations in sections 3.4.3 and 3.4.4.

[8] FrontlineSolvers (2015). *EXCEL SOLVER - integer programming.* [Online] Available from: http://www.solver.com/excel-solver-integer-programming, [Accessed: 12th October 2015]. 5 citations in sections 2.5, 7, 7.2.1, 7.2.3, and 7.3.

[9] GUROPI Optimization (2015). *Mixed-Integer Programming (MIP) - a primer on the basics*. [Online] Available from: `http://www.gurobi.com/resources/getting-started/mip-basics`, [Accessed: 25th September 2015]. No citations.

[10] IBM Knowledge Centre (2015). *Introducing ILOG JRules BRMS*. [Online] Available from: `http://www-01.ibm.com/support/knowledgecenter/SS6MTS_7.1.1/com.ibm.websphere.ilog.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global7.html`, [Accessed: 12th October 2015]. 3 citations in sections (document), 4.3, and 4.2.

[11] Jboss Community (2015). *Drools*. [Online] Available from: `http://www.drools.org/`.[Accessed: 17th September 2015]. 5 citations in sections 2.2, 2.2, 2.8, 2.8.1, and 8.

[12] Linux Foundation Collaborative Projects (2015). *Xen Project*. [Online] Available from: `http://www.xenproject.org/`, [Accessed: 17th September 2015]. No citations.

[13] Standard Performance Evaluation Corporation (2015). *SPECpower_ssj2008 Results*. [Online] Available from: `https://www.spec.org/power_ssj2008/results/`.[Accessed: 10th October 2012]. 2 citations in sections 3.3.1 and 7.2.2.

[14] VMWare Community (2015). *VMWare*. [Online] Available from: `http://www.vmware.com/`, [Accessed: 17th September 2015]. One citation in section 2.1.

[15] Abdulla, P. A. and Mayr, R. (2012). 'Petri nets with time and cost'. pages 9–24. 3 citations in sections 6, 6.1, and 6.3.

[16] Abdulla, P.A. and Mayr, R. (2009). 'Minimal cost reachability/coverability in priced timed petri nets'. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 348–363. Springer Berlin Heidelberg. 6 citations in sections 2.10.1, 2.10.1, 2.10.2, 6, 6.1, and 6.3.

[17] Abdulla, P.A and Mayr, R. (2011). 'Computing optimal coverability costs in priced timed petri nets'. In *2011 26th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 399–408, June. 4 citations in sections 2.10.1, 2.10.1, 2.10.2, and 6.

[18] Akshat,V., Kumar,G., Koller,R. and Sen, A. (2011). 'CosMig: modeling the impact of reconfiguration in a cloud'. In *2011 IEEE 19th International Symposium*

on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), pages 3–11, July. No citations.

[19] Almeida A. Bencomo N. Alansari, M. and B. Bordbar. 'CloudMPL: A Domain Specific Language For Describing Management Policies For An Autonomic Cloud Infrastructure'. In *The Fifth International Conference On Cloud Computing and Services Science*, Lisbon, Portugal, 2015. SCITEPRESS. 4 citations in sections (document), 4.4, 4.4.1, and 4.4.

[20] Amador, L. (2012). *Drools Developer's Cookbook*. Packt Publishing Ltd. 2 citations in sections 2.2 and 3.3.

[21] Amazon Company (2015). *Amazon web services*. [Online] Available from: `https://aws.amazon.com/?nc2=h_lg`, [Accessed: 17th September 2015]. 2 citations in sections 1.1 and 2.1.1.

[22] Ayad, A. and Dippel, U. (2010) . 'Agent-based monitoring of virtual machines'. In *2010 International Symposium in Information Technology (ITSim)*, volume 1, pages 1–6, June. One citation in section 1.1.

[23] Behrmann,G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J. and Vaandrager, F. (2001). 'Minimum cost reachability for priced time automata'. In M. Di Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer Berlin Heidelberg. 2 citations in sections 2.10.1 and 2.10.1.

[24] Bo, L., Jianxin,L., Jinpeng,H. Wo,T., Qin,L. and Zhong, L. (2009). 'EnaCloud: an energy-saving application live placement approach for cloud computing environments'. In *IEEE International Conference on Cloud Computing*, pages 17–24. Ieee. 2 citations in sections 1 and 3.4.

[25] Board, J.A. (1990). *Transputer Research and Applications, 2: NATUG-2, Proceedings of the Second Conference of the North American Transputer Users Group, October 18-19, 1989, Durham, NC.* Number no. 3 in Transputer and Occam Engineering Systems. IOS Press. One citation in section 2.7.2.

[26] Borgetto, D., Maurer, M., Da-Costa, G., Pierson, J.M and Brandic, I. (2012). 'Energy-efficient and SLA-aware management of IaaS clouds'. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 25:1–25:10, New York, NY, USA. ACM. 10 citations in sections 1, 1.2, 2.7.3, 3.4.2, 4.2.1, 5, 5.2, 5.2, 2, and 2.

[27] Bousquet, A. and Briffaut, J. and Toinard, C. (2014). ‘An autonomous Cloud management system for in-depth security’. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 368–374, Oct. One citation in section 1.1.

[28] Bo,Z., Yizhi,Q, Tao, M. and Peng,L. (2011). ‘Optimization of consuming resource problem based on reachability graph of petri nets’. In *2011 30th Chinese Control Conference (CCC)*, pages 1745–1748, July. No citations.

[29] Bradley,S., Hax,A. and Magnanti,T. (1977). *Applied Mathematical Programming*. Addison-Wesley. 2 citations in sections 2.5 and 2.5.

[30] Brandtzæg, E. and Parastoo,M. and Mosser, S. (2012). ‘Towards a domain-specific language to deploy applications in the clouds’. In *The Third International Conference on Cloud Computing, GRIDs, and Virtualization. Cloud Computing 2012,*, pages 213–218. One citation in section 2.9.1.

[31] Brown,R. (2008). *Report to Congress on Server and Data Center Energy Efficiency:Public Law 109-431*. [Online] Available from: `https://ses.lbl.gov/sites/all/files/pdf_1.pdf`.[Accessed: 7th November 2015]. 2 citations in sections 1 and 8.

[32] Bunch, C., Chohan, N., Krintz, C. and Shams, K. (2011). ‘Neptune: a domain specific language for deploying HPC software on cloud platforms’. In *Proceedings of the 2Nd International Workshop on Scientific Cloud Computing*, ScienceCloud ’11, pages 59–68, New York, NY, USA. ACM. One citation in section 2.9.1.

[33] Buyya, R., Broberg, J. and Goscinski, A.M. (2010). *Cloud Computing: Principles and Paradigms*, volume 87 of *Wiley Series on Parallel and Distributed Computing*. John Wiley & Sons. No citations.

[34] Buyya, R., Chee S.Y. and Venugopal, S. (2008). ‘Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities’. In *10th IEEE International Conference on High Performance Computing and Communications,2008. HPCC ’08.*, pages 5–13, Sept. One citation in section 1.1.

[35] Buyya,R. and Beloglazov, A. (2010). ‘Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers’. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM. 7 citations in sections 1, 1.2, 3.3.1, 3.3.1, 3.4, 3.4.1, and 3.

[36] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F. and Buyya, R. (2011).

'CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms'. *Software: Practice and Experience*, 41(1):23–50. One citation in section 6.

[37] Callou, G., Maciel, Ermeson C. , Nogueira, B. and Tavares, E. (2008). 'A coloured petri net based approach for estimating execution time and energy consumption in embedded systems'. In *Proceedings of the 21st Annual Symposium on Integrated Circuits and System Design*, SBCCI '08, pages 134–139, New York, NY, USA. ACM. One citation in section 2.4.

[38] Chandrakantha, L. (2008). *Using Excel Solver in Optimization Problems. Mathematics and Computer Science Department, New York.* No citations.

[39] Chhetri, M.B. and Quoc Bao Vo and Kowalczyk, R. (2012). 'Policy-based automation of SLA establishment for cloud computing services'. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 164–171, May. One citation in section 1.1.

[40] Cirstea,H., Kirchner, C., Moossen, M. and Moreau, P. (2004). Production Systems and Rete Algorithm Formalisation. Contrat A04-R-546 —— cirstea04d. Rapport de contrat. [Online] Available from: `https://hal.inria.fr/inria-00099850/file/A04-R-546.pdf`, [Accessed: 29$^{\text{th}}$ September 2015]. No citations.

[41] Cunha, I., Almeida, J., Almeida, V. and Santos, M. (2007). 'Self-adaptive capacity management for multi-tier virtualized environments'. In *2007 10th IFIP/IEEE International Symposium on Integrated Network Management. IM '07.*, pages 129–138. 2 citations in sections 2.7.1 and 2.7.1.

[42] Cunha, M., Mendonca, N. and Sampaio, A. (2013). 'A declarative environment for automatic performance evaluation in IaaS clouds'. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 285–292. One citation in section 2.9.1.

[43] Dantzig,G.B. and Thapa, M.N. (1997). *Linear Programming 1: Introduction.* Springer-Verlag New York, Inc., Secaucus, NJ, USA. No citations.

[44] de Chaves, S.A., Westphall, C.B. and Lamin, F.R. (2010). 'SLA perspective in security management for Cloud Computing'. In *2010 Sixth International Conference on Networking and Services (ICNS)*, pages 212–217, March. One citation in section 1.1.

[45] Der Jeng, M. and Chen, S. C. (1999). 'Heuristic search based on Petri net structures for FMS scheduling'. *IEEE Transactions on Industry Applications*, 35(1):196–202. One citation in section 2.10.2.

[46] Di Bona, D., Lo Re,Aiello, G., Tamburo, A. and Alessi, M. (2011). 'Methodology for graphical modeling of business rules'. In *2011 UKSim 5th European Symposium on Computer Modeling and Simulation*, pages 102–106. Ieee. One citation in section 2.8.

[47] Diaz, M. (2009). *Petri Nets: Fundamental models, Verification and Applications.* ISTE. Wiley. 4 citations in sections 2.3, 2.3.2, 1, and 2.3.2.

[48] Doorenbos, R.B. (1995). *Production Matching for Large Learning Systems.* PhD thesis, Pittsburgh, PA, USA. UMI Order No. GAX95-22942. One citation in section 2.2.

[49] Durkin, J. (1994). *Expert Systems: Design and Development.* Macmillan. One citation in section 3.1.

[50] Emeakaroha, V.C. and Brandic, I. and Maurer, M. and Dustdar, S. (2010). 'Low level metrics to high level SLAs - LoM2HiS framework: bridging the gap between monitored metrics and SLA parameters in cloud environments'. In *2010 International Conference on High Performance Computing and Simulation (HPCS)*, pages 48–54, June. One citation in section 1.1.

[51] Feller, E. , Rilling, L. and Morin, C. (2011). 'Energy-aware ant colony based workload placement in clouds'. In *2011 12th IEEE/ACM International Conference on-Grid Computing (GRID)*, pages 26–33. One citation in section 2.7.2.

[52] Flores,M. and Roberto A. (1999). 'Towards a standardization of multi-agent system framework'. *Crossroads*, 5(4):18–24, June 1999. One citation in section 8.2.

[53] Forgy, C.L. (1982). 'Rete : a fast algorithm for the many pattern/many object pattern match problem'. *Artificial Intelligence*, 19:17–37. 2 citations in sections 2.2 and 2.2.

[54] Foster, I., Yong, Z., Raicu,I. and Shiyong L. (2008). 'Cloud computing and grid computing 360-degree compared'. In *Grid Computing Environments Workshop, 2008. GCE '08*, volume 1, pages 1–10. IEEE. One citation in section 1.1.

[55] Freeman, E. , Robson, E. , Bates, B. and Sierra, K. (2004). *Head First Design*

Patterns. Head First Series. O'Reilly Media, Incorporated. One citation in section 1.2.

[56] Gat, E. (1998). *Artificial Intelligence and Mobile Robots*. chapter Three-layer Architectures, pages 195–210. MIT Press, Cambridge, MA, USA, 1998. No citations.

[57] GITE, V. (2001). *How to stress test cpu and memory (vm) on a linux and unix with stress-ng*. [Online] Available from: `http://www.cyberciti.biz/faq/stress-test-linux-unix-server-with-stress-ng/`.[Accessed: 29th January 2015]. One citation in section 3.4.3.

[58] Gmach, D. , Rolia, J. and Cherkasova, L. (2009). 'Satisfying service level objectices in a self-managing resource pool'. In *2009. SASO '09. Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 243–253. 3 citations in sections 2.7.1, 2.7.1, and 2.7.1.

[59] Goiri, I., JuliaÌĂ, F., Nou, R., Berral, J.L., Guitart, J. and Torres, J. (2010). 'Energy-aware scheduling in virtualized datacenters'. In *2010 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 58–67, Sept. One citation in section 1.

[60] Graham, I. (2007). *Business Rules Management and Service Oriented Architecture: A Pattern Language*. Wiley. One citation in section 4.2.

[61] Gueyoung J., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D. and Pu, C. (2010). 'Mistral: dynamically managing power, performance, and adaptation cost in cloud infrastructures'. In *2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, pages 62–73. 2 citations in sections 2.7.1 and 2.7.1.

[62] Guitart, J., Macias, M., Djemame, K., Kirkham, T., Ming Jiang and Armstrong, D. (2013). 'Risk-driven proactive fault-tolerant operation of IaaS providers'. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, volume 1, pages 427–432, Dec. One citation in section 1.1.

[63] Haibo,M., Huaimin, W., Gang Y., Yangfan Z.,Dianxi S. and Lin,Y. (2010). 'Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers'. In *2010 IEEE International Conference on Services Computing (SCC)*, pages 514–521. 2 citations in sections 2.7.1 and 2.7.1.

[64] Hebborn, J. (2000). *Decision Mathematics*. Number v. 1 in Heinemann modular mathematics for Edexcel AS and A-Level. Pearson Education. 2 citations in sections

3.1.1 and 3.3.

[65] Herbst, H., Knolmayer, G., Myrach, T. and Schlesinger, M. (1994). 'The specification of business rules: a comparison of selected methodologies'. In *Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle*, pages 29–46, New York, NY, USA. Elsevier Science Inc. No citations.

[66] Hien, N. V., Tran, F.D. and Menaud, J.-M. (2010). 'performance and power management for cloud infrastructures'. In *2010 IEEE 3rd International Conference onCloud Computing (CLOUD)*, pages 329–336, July. One citation in section 1.

[67] HP Helion Eucalyptus (2015). *Official documentation for Eucalyptus Cloud*. [Online] Available from: https://www.eucalyptus.com/docs/eucalyptus/4.1.2/index.html, [Accessed: 17th September 2015]. 3 citations in sections 1.1, 2.1, and 8.2.

[68] Hui,C., Chunjie, Z., Yuanqing, Q., Vandenberg, A., Vasilakos, A.V. and Naixue, X. (2010). 'Petri net modeling of the reconfigurable protocol stack for cloud computing control systems'. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 393–400, Nov. One citation in section 2.11.

[69] IBM-ILOG (2007). *ILOG JRules Techinical*. [Online] Available from:http://logic.stanford.edu/poem/externalpapers/iRules/WP-JRules50Strengths.pdf.[Accessed: 8th November 2014]. One citation in section 4.3.

[70] Jboss.org (2015). *Drools tools reference guide*. [Online] Available from: http://docs.jboss.org/drools/release/6.2.0.CR3/drools-docs/html/index.html.[Accessed: 27th July 2015]. 11 citations in sections (document), 2.2, 2.3, 2.2, 2.4, 2.2, 3.1.1, 3.3, 4.1, 4.2, and 4.3.

[71] Jensen, K. (1992). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Number v. 1 in Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg. 6 citations in sections 3, 2.4, 2.4.1, 2.4.2, 2.4.3, and 2.4.4.

[72] Jensen, K. (1995). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Number v. 2 in Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg. 2 citations in sections 2.4.5 and 2.4.5.

[73] Jensen, K. (1997). 'A brief introduction to coloured petri nets'. In *Proceedings*

of the Third International Workshop on Tools and Algorithms for Construction and Analysis of Systems, TACAS '97, pages 203–208, London, UK, UK. Springer-Verlag. 4 citations in sections 2.4, 2.4.3, 5.2.1, and 6.

[74] Jensen, K. and Kristensen, L.M. (2009). *Coloured Petri Nets: Modelling and Validation of Concurrent Systems.* Springer. 19 citations in sections 2.4, 2.4.1, 2, 2.4.1, 1, 2.4.1, 2.4.2, 1, 2.4.3, 2.4.4, 2.4.5, 5, 2.4.5, 1, 2.4.5, 5.2.1, 6, 7.3, and 8.2.

[75] Jensen, K., Kristensen, L. and Wells, L. (2007). 'Coloured petri nets and CPN tools for modelling and validation of concurrent systems'. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254. One citation in section 6.

[76] Jing, X. and Fortes, J.A.B. (2010). 'Multi-objective virtual machine placement in virtualized data center environments'. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 179–188. One citation in section 2.7.2.

[77] Kalman,R. (1959). 'On the general theory of control systems'. *IRE Transactions on Automatic Control*, 4(3):110–110, 1959. One citation in section 2.7.1.

[78] Kecskemeti, G., Maurer, M., Brandic, I., Kertesz, A., Nemeth, Z. and Dustdar, S. (2012). 'Facilitating self-adaptable inter-cloud management'. In *2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 575–582. IEEE. No citations.

[79] Kipp,A. and Jiang, T., Fugini, M. and Salomie, I. (2012). 'Layered Green Performance Indicators'. *Future Gener. Comput. Syst.*, 28(2):478–489. 6 citations in sections 1, 1.2, 4.2.1, 4.2.1, 4.2.1, and 4.6.

[80] Kliazovich, D., Bouvry, P., Audzevich, Y. and Khan, S.U. (2010). 'GreenCloud: a packet-level simulator of energy-Aware cloud computing data centers'. In *2010 IEEE Global Telecommunications Conference (GLOBECOM 2010)*, pages 1–5. One citation in section 6.

[81] Kounev, S., Brosig, F., Huber, N. and Reussner, R. (2010). 'Towards Self-Aware Performance and Resource Management in Modern Service-Oriented Systems'. In *2010 IEEE International Conference on Services Computing (SCC)*, pages 621–624. 3 citations in sections 1.2, 2.7.3, and 5.

[82] Kramer, J. and Magee, J. (2007). 'Self-managed systems: an architectural chal-

lenge'. In *2007 Future of Software Engineering*, FOSE '07, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society. No citations.

[83] Kusic, D., Kephart, J.O., Hanson, J.E., Nagarajan, K., Guofei, J. (2008). 'Power and performance management of virtualized computing environments via lookahead control'. In *2008 International Conference on Autonomic Computing. ICAC '08.*, pages 3–12. One citation in section 2.7.1.

[84] KVM (2015). *Kernel Virtual Machine.* [Online] Available from: `http://www.linux-kvm.org`, [Accessed: 17th September 2015]. One citation in section 2.1.

[85] Li, L. and Hadjicostis, C.N. (2011). 'Least-cost planning sequence estimation in labelled Petri nets'. *Transactions of the Institute of Measurement and Control*, 33(3-4):317–331. One citation in section 2.10.2.

[86] Li, M., Ye, F., Kim, M., Chen, H. and Lei, H. (2011). 'A scalable and elastic publish/subscribe service'. In *2011 IEEE International Parallel Distributed Processing Symposium (IPDPS)*, pages 1254–1265, May. No citations.

[87] Lirui, B., Tong, L., Xinjun, W. and Zhongwen, X. (2011). 'Charging model research of infrastructure layer in cloud computing based on cost-profit petri net'. In *2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 435–441, Oct. One citation in section 2.11.

[88] Liu, L., Wang, H., Liu, X., Jin, X., He, W.B., Wang, Q. B. and Chen, Y. (2009). 'GreenCloud: a new architecture for green data center'. In *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, ICAC-INDST '09, pages 29–38, New York, NY, USA. ACM. One citation in section 2.6.

[89] Love,C. (2011). *Metering and Monitoring Energy Use in Data Centres.* [Online] Available from: `http://www.goodcampus.org/uploads/DOCS/148-Briefing_Paper_12\_(RECSO)_\discretionary{-}{}{}_Data_Centre_Metering_final.pdf`.[Accessed: 7th November 2015]. No citations.

[90] Lukichev, S., Giurca, A., Wagner, G., Gasevic, D. and Ribaric, M. (2007). 'Using uml-based rules for web services modeling'. In *2007 IEEE 23rd International Conference on Data Engineering Workshop*, pages 290 –297, april. One citation in section 2.8.

[91] Maurer, M. and Brandic, I. and Emeakaroha, V.C. and Dustdar, S. (2010). 'Towards

knowledge management in self-adaptable clouds'. In *6th World Congress on Services (SERVICES-1),2010*, pages 527–534, July. 5 citations in sections 1.1, 1.2, 2.7.3, 4.2, and 5.2.

[92] Maurer, M., Brandic, I. and Sakellariou, R. (2013). 'Adaptive resource configuration for cloud infrastructure management'. *Future Generation Computer Systems*, 29(2):472–487. 9 citations in sections 1.1, 1.2, 2.7.3, 3.4.2, 4.2, 4.2.1, 5, 5.2, and 5.2.

[93] Mehrotra, R., Dubey, A., Abdelwahed, S. and Monceaux, W. (2011). 'Large scale monitoring and online analysis in a distributed virtualized environment'. In *2011 8th IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe)*, pages 1–9, April. One citation in section 1.1.

[94] Mernik,M., Heering,J. and Sloane, A.M. (2005). 'When and how to develop domain-specific languages'. *ACM Comput. Surv.*, 37(4):316–344. One citation in section 2.9.

[95] Mi,H., Wang,H., Yin, G., Zhou, Y., Shi, D. and Yuan, L. (2010). 'Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers'. In *IEEE International Conference on Services Computing*, pages 514–521. Ieee. 2 citations in sections 1 and 3.4.

[96] Milner, R. (1997). *The Definition of Standard ML: Revised.* MIT Press. One citation in section 2.4.

[97] Minarolli, D. and Freisleben, B. (2011). 'Utility-driven allocation of multiple types of resources to virtual machines in clouds'. In *2011 IEEE 13th Conference on Commerce and Enterprise Computing (CEC)*, pages 137–144, Sept. 3 citations in sections 1.1, 2.7.1, and 2.7.1.

[98] Ming,M., Jie,L. and Humphrey, M. (2010). 'Cloud auto-scaling with deadline and budget constraints'. In *11th IEEE/ACM International Conference on Grid Computing (GRID), 2010*, pages 41–48. No citations.

[99] Morťn, D., Vaquero, L.M. and Galťn, F. (2011). 'Elastically ruling the cloud: specifying application's behavior in federated clouds'. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 89–96. Ieee. No citations.

[100] Murata, T. (1989). 'Petri nets: properties, analysis and applications'. *Proceedings of the IEEE*, 77(4):541–580, Apr. 3 citations in sections (document), 2.3, and 2.5.

[101] Murugesan, S. and Gangadharan, G. (2012). *'Green Cloud Computing and Envi-*

ronmental Sustainability', page 432. Wiley-IEEE Press. 3 citations in sections 1, 1.1, and 8.

[102] Murwantara, I. M., Bordbar, B. and Minku, L. (2014). 'Measuring energy consumption for web service product configuration'. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications &#38; Services*, iiWAS '14, pages 224–228, New York, NY, USA, 2014. ACM. One citation in section 3.3.1.

[103] Nathuji,R., Kansal, A. and Ghaffarkhah, A. (2010). 'Q-clouds: managing performance interference effects for QoS-aware clouds'. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 237–250, New York, NY, USA. ACM. One citation in section 1.1.

[104] Nunez, A., Castane, G.G., Vazquez-Poletti, J.L., Caminero, A.C., Carretero, J. and Llorente, I.M. (2011). 'Design of a flexible and scalable hypervisor module for simulating cloud computing environments'. In *International Symposium on Performance Evaluation of Computer Telecommunication Systems (SPECTS), 2011*, pages 265–270. One citation in section 6.

[105] OpenNebula Community (2015). *Opennebula: The open source toolkit for cloud computing.* [Online] Available from: `http://www.opennebula.org`.[Accessed: 15th December 2012]. 17 citations in sections (document), 1.1, 1.2, 1.2, 2, 2.1, 2.1.1, 2.1.1, 2.2, 2.7.1, 3.1, 3.3, 3.5, 4.2, 5.2, 8, and 8.2.

[106] ovirt Community (2015). *ovirt.* [Online] Available from: `http://www.ovirt.org/Documentation`, [Accessed: 17th September 2015]. One citation in section 2.1.

[107] Padala, P. , Shin, K.G. , Zhu, X. , Uysal, M., Wang, Z., Singhal, S. , Merchant, A. and Salem, K. (2007). 'Adaptive control of virtualized resources in utility computing environments'. In *2007 Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys '07, pages 289–302, New York, NY, USA. ACM. No citations.

[108] Paulson, L.C. (1996). *ML for the Working Programmer.* Cambridge University Press. One citation in section 2.4.

[109] Perez-Palacin, D., Mirandola, R. and Merseguer, J. (2011). 'Enhancing a qos-based self-adaptive framework with energy management capabilities'. In *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical*

Systems – ISARCS, QoSA-ISARCS '11, pages 165–170, New York, NY, USA. ACM. No citations.

[110] Pokharel,M. and Park, J.S. (2009). 'Cloud computing: future solution for e-Governance'. In *Proceedings of the 3rd International Conference on Theory and Practice of Electronic Governance*, ICEGOV '09, pages 409–410, New York, NY, USA. ACM. No citations.

[111] Qian, Z. and Agrawal, G. (2012). 'Resource provisioning with budget constraints for adaptive applications in cloud environments'. *IEEE Transactions on Services Computing*, 5(4):497–511. 3 citations in sections 1.1, 2.7.1, and 2.7.1.

[112] REWERSE (2008). *UML-based Rule Modeling Language*. [Online] Available from:`http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=URM`.[Accessed: 11th October 2014]. 9 citations in sections 2.8, 2.8.1, 4.1, 4.1.1, 4.1.2, 4.3, 4.3.1, 4.3.1, and 4.3.2.

[113] REWERSE (2008). *URML Meta Model*. [Online] Available from:`https://oxygen.informatik.tu-cottbus.de/strelka/URML-Metamodel.htm`.[Accessed: 11th October 2015]. 6 citations in sections (document), 2.8, 4.1, 4.1.1, 4.1.2, and 4.3.

[114] Rosenberg, F. and Dustdar, S. (2005). 'Design and implementation of a service-oriented business rules broker'. In *Seventh IEEE International Conference on E-Commerce Technology Workshops*, pages 55–63. Ieee. One citation in section 2.7.3.

[115] Satoh, F. and Itakura, M. (2011). 'Cloud-based infrastructure for managing and analyzing environmental resources'. In *2011 Annual SRII Global Conference (SRII)*, pages 325–334. No citations.

[116] Shalloway,A. and Trott,J. (2003). *Design Patterns: Elements of Reusable Object-Oriented Software with Applying Uml and Patterns:An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Addison Wesley. One citation in section 2.7.3.

[117] Spivey, W. A. (1963). 'Linear Programming. an introduction '. *The Macmillan Company*, 7. One citation in section 2.5.

[118] Taha, H.A. and Schmidt, J.W. (2014). *Integer Programming: Theory, Applications, and Computations*. Operations Research and Industrial Engineering. Elsevier Science, 2014. 4 citations in sections 2.5, 1, 2, and 2.5.

[119] The Apache Software Foundation (2015). *Apache JMeter*. [Online] Available from: http://jmeter.apache.org/, [Accessed: 29th September 2015]. One citation in section 6.2.1.

[120] Toraldo, G. (2012). *OpenNebula 3 Cloud Computing*. PACKT Publishing Ltd. 10 citations in sections (document), 1.2, 1.2, 2, 2.1, 2.1.1, 2.1, 2.1.1, 2.1.1, and 2.6.

[121] Van,H.N., Tran, F.D. and Menaud, J.M. (2009). 'SLA-aware virtual resource management for cloud infrastructures'. In *2009 Ninth IEEE International Conference on Computer and Information Technology. CIT '09*, volume 1, pages 357–362. 2 citations in sections 2.7.1 and 2.7.1.

[122] Vaquero, L.M., Morán, D., Galán, F. and Alcaraz-Calero, J.M. (2012). 'Towards runtime reconfiguration of application control policies in the cloud'. *Journal of Network and Systems Management*, 20(4):489–512, Aug. One citation in section 2.7.3.

[123] Verma, A., Ahuja, P. and Neogi, A. (2008). 'pMapper: Power and migration cost aware application placement in virtualized systems'. In *Proceedings of the 9th ACM International Conference on Middleware*, Middleware '08, pages 243–264, New York, NY, USA. Springer-Verlag New York, Inc. No citations.

[124] Walsh, W.E., Tesauro, G., Kephart, J.O. and Das, R. (2004). 'Utility functions in autonomic systems'. In *2004 Proceedings. International Conference on Autonomic Computing*, pages 70–77. 2 citations in sections 2.7.1 and 2.7.1.

[125] Wampler,J.F. and Newman, S.E. (1996). 'Integer Programming'. *The College Mathematics Journal*, 27(2):95–100, 1996. No citations.

[126] Wang, J., Zhou, R., Li, J. and Wang, G. (2014). 'A distributed uule engine based on message-passing model to deal with big data. *Lecture Notes on Software Engineering*, 2(3):275–281, 2014. One citation in section 3.5.

[127] Wei,C., Xiaoqiang,Q., Jun, W. and Tao, H. (2012). 'A two-level virtual machine self-reconfiguration mechanism for the cloud computing platforms'. In *2012 9th International Conference onUbiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC)*, pages 563–570, Sept. No citations.

[128] Wesolowski, K. (2009). *Introduction to Digital Communication Systems*. Wiley. One citation in section 2.10.2.

[129] Whitney,J. and Pierre Delforge,P. (2014). *Scaling Up Energy Efficiency Across the Data Center Industry: Evaluating Key Drivers and Barriers*, August. [Online] Available from: https://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf.[Accessed: 7th November 2015]. 3 citations in sections 1, 4.2.1, and 8.

[130] Whittle,J., Sawyer, P., Bencomo, N., Cheng, B.H. and Bruel, J.M. (2010). 'RELAX: a language to address uncertainty in self-adaptive systems requirement'. *Requirements Engineering*, 15(2):177–196, 2010. One citation in section 4.4.1.

[131] Wood, T., Shenoy, P., Venkataramani, A. and Yousif, M.(2009). 'Sandpiper: black-box and gray-box resource management for virtual machines'. *Computer Networks*, 53(17):2923 – 2938. Virtualized Data Centers. 2 citations in sections 2.7.1 and 2.7.1.

[132] Xtext (2014). *Xtext Textual Domain-specific Language (DSL).* [Online] Available from: http://www.eclipse.org/Xtext/ .[Accessed: 29th March 2014]. One citation in section 4.5.

[133] Xu, J. and Fortes, J. (2010). 'Multi-objective virtual machine placement in virtualized data center environments'. In *IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 179–188. Ieee. 3 citations in sections 1, 1.1, and 3.4.

[134] Ye, K., Huang, D., Jiang, X., Chen, H. and Wu, S. (2010). 'Virtual machine based energy-efficient data center architecture for cloud computing: a Performance perspective'. In *Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 171–178, Washington, DC, USA. IEEE Computer Society. One citation in section 1.1.

[135] Yixin, D., Hellerstein, J.L., Parekh, S., Griffith, R., Kaiser, G. and Phung, D. (2005). 'Self-managing systems: a control theory foundation'. In *2005 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems. ECBS '05.*, pages 441–448. 2 citations in sections 2.7.1 and 3.1.

[136] Zhang, B., Qu, Y., Ma, T. and Li P. (2011). 'Optimization of consuming resource problem based on reachability graph of petri nets'. In *2011 30th Chinese Control Conference (CCC)*, pages 1745–1748, July. One citation in section 2.10.2.

[137] Zhang, Q., Cheng, L. and Boutaba, R. (2010). 'Cloud computing: state-of-the-art and research challenges'. volume 1, pages 7–18. Springer-Verlag. No citations.

[138] Zurawski, R. and MengChu Z. (1994). 'Petri nets and industrial applications: a tutorial'. *IEEE Transactions on Industrial Electronics*, 41(6):567–583. One citation in section 2.3.1.