A Reinforcement Learning Based Management Solution for Context-Aware Environments Applied on

Green Datacenters

State Reward Computation, Nash Equilibrium Policy Negotiation

and Action Enforcement

Georgiana Copil

June, 16, 2010

# Abstract

This report presents a framework for building self-\* systems in context-aware environments. The solution gathers information from the context and keeps it in an acceptable state by enforcing its associated policies. The fundamental part of this framework is the reinforcement learning approach for finding the best sequence of actions which will be used for bringing the context back to an acceptable state. This approach is used both for enforcing a self-healing behavior for the environment in which a datacenter resides, and for lowering the amount of energy consumed through a self-adapting algorithm. The policies associated with the resources of the context give rules for the way the context should look. Through the reinforcement learning approach actions are found to keep the context in a state defined as acceptable by the policies. The presented approach is general, it giving the possibility to enforce any of the self-\* behaviors, on any context by simply representing the context under the RAP model and writing adequate policies for the respective context. The proposed solution is tested on simulated contexts and on real context validating both its functionality and scalability.

# Table of Contents

[Abstract I](#_Toc264727068)

[Table of Contents i](#_Toc264727069)

[List of Figures iv](#_Toc264727070)

[List of Tables v](#_Toc264727071)

[List of Listings vi](#_Toc264727072)

[Glossary vii](#_Toc264727073)

[1. Introduction 1](#_Toc264727074)

[1.1 Motivation 1](#_Toc264727075)

[1.2 Objectives and contributions 2](#_Toc264727076)

[1.3 Publications 3](#_Toc264727077)

[1.3.1 A Reinforcement Learning based Self-healing Algorithm for Managing Context Adaptation [3] 3](#_Toc264727078)

[1.3.2 An Autonomic Algorithm for Energy Efficiency in Service Centers [4] 3](#_Toc264727079)

[1.4 Overview of the report 3](#_Toc264727080)

[2. Theoretical Background 4](#_Toc264727081)

[2.1 Context-Aware Computing 4](#_Toc264727082)

[2.2 Energy-Aware Computing 4](#_Toc264727083)

[2.2.1 Virtualization 5](#_Toc264727084)

[2.2.1.1 Hypervisors 5](#_Toc264727085)

[2.2.1.2 Virtual appliances 6](#_Toc264727086)

[2.2.2 Hyper-V WMI Provider 7](#_Toc264727087)

[2.3 Autonomic Features 7](#_Toc264727088)

[2.4 Context Model 8](#_Toc264727089)

[2.5 Mobile Agents – JADE 9](#_Toc264727090)

[3. Related Work 10](#_Toc264727091)

[3.1 Context Aware Systems 10](#_Toc264727092)

[3.2 Energy Aware Systems 11](#_Toc264727093)

[3.3 Self-\* Systems 12](#_Toc264727094)

[3.4 SLA Negotiation 13](#_Toc264727095)

[4. Analysis and design 15](#_Toc264727096)

[4.1 Conceptual architecture 16](#_Toc264727097)

[4.2 The Self-Healing Behavior 17](#_Toc264727098)

[4.2.1 Algorithm Description 17](#_Toc264727099)

[4.2.2 Recheck Time 18](#_Toc264727100)

[4.2.3 Rollback 18](#_Toc264727101)

[4.2.4 Inter-Independent Resources Group 19](#_Toc264727102)

[4.2.4.1 Dependency between Resources 19](#_Toc264727103)

[4.2.4.2 Inter-Independent Resources Group 19](#_Toc264727104)

[4.2.5 Mathematical Description of the Context 19](#_Toc264727105)

[4.2.5.1 Entropy 20](#_Toc264727106)

[4.2.5.2 Policy Contribution to the Entropy 21](#_Toc264727107)

[4.2.5.3 Entropy Threshold 21](#_Toc264727108)

[4.2.6 Algorithm Complexity 22](#_Toc264727109)

[4.3 Datacenter Self-Adapting Behavior 22](#_Toc264727110)

[4.3.1 Datacenter Context Mapping 24](#_Toc264727111)

[4.3.2 Entropy 25](#_Toc264727112)

[4.3.3 Reward in Reinforcement Learning 25](#_Toc264727113)

[4.3.4 Task History 25](#_Toc264727114)

[4.3.5 Tail Server 26](#_Toc264727115)

[4.3.6 Algorithm Complexity 26](#_Toc264727116)

[4.4 SLA Negotiation between Client and Datacenter 27](#_Toc264727117)

[4.4.1 Fuzzy Logic Negotiation 29](#_Toc264727118)

[4.4.2 Nash Negotiation 29](#_Toc264727119)

[4.4.2.1 Straight Forward Nash Negotiation 29](#_Toc264727120)

[4.4.2.2 Nash Bargaining Solutions 30](#_Toc264727121)

[4.4.3 Dynamic Policy Concept 30](#_Toc264727122)

[4.5 Isomorphism of Self-\* 30](#_Toc264727123)

[5. Implementation 31](#_Toc264727124)

[5.1 Context Management Infrastructure Implementation 33](#_Toc264727125)

[5.1.1 JADE Overview 33](#_Toc264727126)

[5.1.2 Hyper-V WMI Provider 34](#_Toc264727127)

[5.2 Agents’ Implementation 36](#_Toc264727128)

[5.2.1 Context Model Administering Agent 37](#_Toc264727129)

[5.2.1.1 Environment Ontologies 37](#_Toc264727130)

[5.2.1.2 Creating the Agents 39](#_Toc264727131)

[5.2.2 Reinforcement Learning Agent 39](#_Toc264727132)

[5.2.2.1 Reinforcement Learning Datacenter Behavior 40](#_Toc264727133)

[5.2.3 Task Management Agent 42](#_Toc264727134)

[5.3 Utility Components 43](#_Toc264727135)

[5.3.1 Nash Negotiation Component 43](#_Toc264727136)

[5.3.2 Hyper-V Management WMI Endpoint 45](#_Toc264727137)

[5.3.3 Enforcement and Simulation Unit 47](#_Toc264727138)

[6. Results and Testing 49](#_Toc264727139)

[6.1 Environment Self-Healing 49](#_Toc264727140)

[6.1.1 Tracing 50](#_Toc264727141)

[6.1.1.1 Simple Situation 50](#_Toc264727142)

[6.1.1.2 Repeating pattern 51](#_Toc264727143)

[6.4.1.3 Random Sensor Values 52](#_Toc264727144)

[6.2 Datacenter Self-Adaptation 52](#_Toc264727145)

[6.2.1 Simulating A Datacenter in X3D 52](#_Toc264727146)

[6.2.1.1 Hardware infrastructure 53](#_Toc264727147)

[6.2.1.2 Tracing 55](#_Toc264727148)

[6.2.2 Real-World Testing 57](#_Toc264727149)

[6.2.2.1 Tracing 57](#_Toc264727150)

[7. Conclusions 61](#_Toc264727151)

[7.1 Results 61](#_Toc264727152)

[7.2 Encountered Impediments 62](#_Toc264727153)

[7.3 Further Development 63](#_Toc264727154)

[References 64](#_Toc264727155)

[Appendix A: Java Source Code 68](#_Toc264727156)

[NashNegotiator.java 68](#_Toc264727157)

[ReinforcementLearningDataCenterManagementBehavior.java 71](#_Toc264727158)

[TaskManagementAgent.java 82](#_Toc264727159)

[DeployTaskCommand.java 83](#_Toc264727160)

[TaskDto.java 85](#_Toc264727161)

[ReceiveMessageRLBehaviour.java 88](#_Toc264727162)

[DefaultQoSPolicy.java 91](#_Toc264727163)

[Appendix B: C# Source Code 93](#_Toc264727164)

[VMHandling.cs 93](#_Toc264727165)

[Service1.cs 104](#_Toc264727166)

# List of Figures

[Figure 4.1: GAMES datacenter infrastructure 14](#_Toc264403563)

[Figure 4.2: Self-healing algorithm 16](#_Toc264403564)

[Figure 4.3: Self-adapting Algorithm 21](#_Toc264403565)

[Figure 4.4: Reinforcement Learning Flow 21](#_Toc264403566)

[Figure 4.5: Reinforcement Learning Algorithm- Contexts’ Expansion 22](#_Toc264403567)

[Figure 4.6: Tail Server 24](#_Toc264403568)

[Figure 4.7: Distance between two servers 26](#_Toc264403569)

[Figure 5.1: View on the System 29](#_Toc264403570)

[Figure 5.2: Conceptual architecture of the system 30](#_Toc264403571)

[Figure 5.3: Relations between agents 31](#_Toc264403572)

[Figure 5.4: FIPA Standard: Services Provided by a Platform 33](#_Toc264403573)

[Figure 5.5 : Hyper-V Node 34](#_Toc264403574)

[Figure 5.6: Hyper-V Services Structure 34](#_Toc264403575)

[Figure 5.7: WMI and Relational Database Comparison 35](#_Toc264403576)

[Figure 5.8: Environment Ontology 36](#_Toc264403577)

[Figure 5.9: Actuator instance example: Heater 36](#_Toc264403578)

[Figure 5.10: SWRL Rule 37](#_Toc264403579)

[Figure 5.11: Java Implementation for Ontologies 37](#_Toc264403580)

[Figure 5.12: Add Task GUI 42](#_Toc264403581)

[Figure 5.13: Virtual Machine Handling 45](#_Toc264403582)

[Figure 5.14: Command Package 46](#_Toc264403583)

[Figure 5.15: Command Package Class Diagram 47](#_Toc264403584)

[Figure 6.1: X3D Context Representation 48](#_Toc264403585)

[Figure 6.2: Ontology for Smart Laboratory 49](#_Toc264403586)

[Figure 6.3: Face Recognition Policy broken 50](#_Toc264403587)

[Figure 6.4: Context disturbed using a pattern 50](#_Toc264403588)

[Figure 6.5: Context randomly disturbed 51](#_Toc264403589)

[Figure 6.6: X3D View on the Datacenter Context 52](#_Toc264403590)

[Figure 6.7: SWRL Rule for Server1 GPI Policy 53](#_Toc264403591)

[Figure 6.8: SWRL Rule for Task 1 KPI Policy 53](#_Toc264403592)

[Figure 6.9 : Datacenter Context Ontology 54](#_Toc264403593)

[Figure 6.10: Deploy Task 1 on Server 1 Action 55](#_Toc264403594)

[Figure 6.11: Wake up Server 3 Action 55](#_Toc264403595)

[Figure 6.12: Deploy Task 2 on Server 3 Action 55](#_Toc264403596)

[Figure 6.13: Deploy Task 3 on Server 3 Action 56](#_Toc264403597)

[Figure 6.14: Before the Self-Adapting Algorithm 57](#_Toc264403598)

[Figure 6.15: Resources Usage Before 57](#_Toc264403599)

[Figure 6.16: All the tasks are deployed 58](#_Toc264403600)

[Figure 6.17: Task 2 runs a small XP 58](#_Toc264403601)

[Figure 6.18: Resources Usage After 59](#_Toc264403602)

# List of Tables

[Table 4.1: Mapping between <R,A,P> and datacenter context 22](#_Toc264409199)

[Table 6.1: Datacenter servers’ characteristics 52](#_Toc264409200)

[Table 6.2: Received tasks SLA requirements 53](#_Toc264409201)

# List of Listings

[Listing 5.1: Method Call Example 36](#_Toc264807565)

[Listing 5.2: Check if a Policy is respected 39](#_Toc264807566)

[Listing 5.3: CMAA creates the other agents 39](#_Toc264807567)

[Listing 5.4: Reinforcement Learning Agent Memory Handling 40](#_Toc264807568)

[Listing 5.5: RLA learning algorithm 41](#_Toc264807569)

[Listing 5.6: RLA learning algorithm 41](#_Toc264807570)

[Listing 5.7: RLA reward function 42](#_Toc264807571)

[Listing 5.8: RLA self-adapting algorithm’s effect on the environment 42](#_Toc264807572)

[Listing 5.9: Send Message to RLA 43](#_Toc264807573)

[Listing 5.10: Nash Utilities 44](#_Toc264807574)

[Listing 5.11: Finding Nash Equilibrium 45](#_Toc264807575)

[Listing 5.12: Import Virtual Machine 46](#_Toc264807576)

[Listing 5.13: Deploy Virtual Machine 47](#_Toc264807577)

# Glossary

ACL – Agent Communication Language

API – Application Programming Interface

CDB – Context Disturbing Behavior

CIA – Context Interpreting Agent

CSV – Cluster Shared Volume

CMAA – Context Model and Administrating Agent

COBRA – Context BRoker Architecture

CPU – Central Processing Unit

DTO – Data Transfer Object

ETTK – IBM Emerging Technologies Toolkit

FIPA –Foundation for Intelligent Physical Agents

GAMES – Green Active Management of Energy in Service centers

GPI – Green Performance Indicators

GUIA – Graphical User Interface Agent

HTTP – HyperText Transfer Protocol

IIRG – Inter-Independent Resources Group

ISCSI – Internet Small Computer System Interface

JDK – Java Development Kit

JeOS –Just enough Operating System

JRE – Java Runtime Environment

JVM – Java Virtual Machine

KPI – Key Performance Indicators

JADE – Java Agent Development Framework

OS – Operating System

OWL – Ontology Web Language

QoS – Quality of Service

RLA – Reinforcement Learning Agent

SLA – Service Level Agreement

SWRL – Semantic Web Rule Language

TMA – Task Management Agent

UbiComp – Ubiquitous Computing

VMM – Virtual Machine Monitor

XML – eXtensible Markup Language

X3DA – X3D Agent

WMI – Windows Management Instrumentation

Chapter 1

# Introduction

## Motivation

Reducing energy consumption and using biodegradable materials has become the main purpose in all the existent industry fields. In the last few decades, researchers have been focusing on improving performance in all possible fields, neglecting overall efficiency considering energy consumption. With the growth of impact of industry and pollution on the environment, the direction of evolution has changed from a performance directed one to an efficiency based approach.

Green computing is a new trend in computer science, encouraging the development of environmentally sustainable computing. Basically, it regards the study and practice of using computing resources efficiently. Studies have been focused on designing computing resources which maximize the energy efficiency during product’s lifetime, are biodegradable and don’t use hazardous materials. Extensive research is being undertaken in design of algorithms and systems for environmentally-friendly computer technologies and efficient use of computers.

On 2007, the report to congress on server and data energy efficiency [1] gave a definite warning in what regards the energy consumption of servers and data centers. From 2000 to 2006 the energy consumption of datacenters has doubled in United States, rising to about 61 billion kilowatt-hours (kWh) and being similar to the amount of electricity consumed by approximately 5.8 million average U.S. households. This is why a great part of the effort of painting the computing industry in green has to be directed towards decreasing datacenters’ and servers’ overall energy consumption, including the cooling infrastructure that supports IT equipment in datacenters.

Reducing the energy consumption in datacenters can be approached by improving the efficiency of computer resources or by software solutions of intelligently using those resources. Several solutions have been approached for decreasing datacenters’ overall energy consumption. These can be categorized in two main directions: hardware solutions and software solutions. The hardware solutions intend to decrease energy consumption by designing low power components thereby reducing the entire’s machine energy consumption. The software solutions are at their beginnings but there have been researches focused on decreasing power consumption depending on the existing load or on distributing efficiently tasks in the datacenter in order to have optimal power consumption.

## Objectives and contributions

The objectives of this thesis project are the following:

* A **description of the datacenter context** composed from servers which have certain resources as CPU, Storage, Memory and this **context’s mapping on the <R,A,P>**(Resources, Actors, Policies) model presented in [2].

The datacenter is composed of a number of servers with different characteristics for different resources like memory, storage and CPU. These are being mapped on a context model composed of Resources, Actors and Policies. The role of Resource is being played by any server in the datacenter and the task plays the Actor role. Considering the previously mentioned elements, policies are being defined using SWRL describing the situations in which the GPI ( Green Performance Indicators) for the datacenter, and the KPI (Key Performance Indicators) for the task are being respected.

* **Developing a self-\* algorithm** with the purpose of optimally distributing the workload, while using as less energy as possible.

Using the above context, a self-adapting algorithm is being designed, for the getting the described model to the lowest possible energy consumption. This algorithm involves a reinforcement learning approach by adapting the self-healing algorithm presented in [3], and adding few improvements. The self-healing algorithm is also being used for controlling the temperature and humidity in the datacenter room.

* **Negotiating the Service Level Agreement** of the task for the case in which the task doesn’t fit in any server on the datacenter, to modify existing policies for the datacenter and make it possible for the tasks to be properly distributed.

Because the user is giving us a requested range for the resources of a server, there might be a situation in which there is no place for that task due to GPI Policies. In this case, a negotiation should be made in order to have a higher maximum accepted value and the task to fit. There is also the case where the task could get more than the minimum requested value, case in which a redistribution of available resources among existing tasks on that server will be made.

## Publications

### A Reinforcement Learning based Self-healing Algorithm for Managing Context Adaptation [3]

This paper presents a reinforcement learning approach for finding the optimum sequence of actions for healing a broken context. For this algorithm to be able to function weights are added to resources and policies and concepts like entropy and inter-independent resources group are being introduced.

### An Autonomic Algorithm for Energy Efficiency in Service Centers [4]

This paper presents an improvement of the previous self-healing algorithm, used for the self-adaption of the energy-efficient datacenter. The self-adapting approach features a closed feedback loop with four MAPE phases: Monitoring, Analysis, Planning and Execution.

## Overview of the report

In the next parts of this report, I will present a theoretical background needed in order to present this project, with available software and technologies needed for implementing it on a real datacenter. The next chapter, chapter 3 gives a view on the existing work in the domain. The fourth chapter describes the algorithms and models and from here on the architecture, the design and implementation is presented ending with testing and conclusion.

Chapter 2

# Theoretical Background

## Context-Aware Computing

Awareness is one of the main problems arising in nowadays computing systems. Building systems which are aware about what happens and about their awareness becomes crucial in hospitals, modern buildings and even personal houses. To handle contexts in all of these environments the system must have sensors for each monitored element and a way to control that element, through different actuators. In order for the system to know when the context is not in the state desired by the user, policies are being described for all elements composing the context. For example, in a smart laboratory, we need to know that if no one is inside the light should be off. Generally the system takes the sensor information and enforces actions on actuators through web services, this way having a low coupled architecture.

## Energy-Aware Computing

Energy-awareness is a subset of context awareness, improving the systems towards green use of the computing infrastructure. The green use of a datacenter implies reducing the energy consumption of computers and other information systems as well as using them in an environmentally sound manner. For the real servers and datacenters to have this behavior there are several options like programmatically assign loads to resources or programmatically assigning loads to servers. The load assignment for datacenters is easily accomplished through virtualization. By this, a system administrator could combine several physical systems into virtual machines on one single, powerful system, thereby unplugging the original hardware and reducing power and cooling consumption.

### Virtualization

According to Wikipedia, the term "virtualization" was coined in the 1960s, to refer to a virtual machine (sometimes called pseudo machine), a term which itself dates from the experimental IBM M44/44X system [5].Virtualization is a new technique reproducing computer hardware through software. In a typical server environment there exist different servers each hosting only one task, for example a web server and a file server. By using server virtualization, both the previously mentioned servers will be running on the same machine, one independently of the other, therefore reducing the costs and energy consumption of a second machine. The center of the entire virtualization process is the virtual machine, it being defined as a software implementation of a machine that executes programs like a physical machine. There are two types of virtual machines: system virtual machines and process virtual machines. From the process virtual machines, the JVM (Java Virtual Machine) and the .NET Framework, which runs on a VM called the Common Language Runtime, are the most known one. Process virtual machines run as a single application inside the operating system, and support one single process. They are created when that process is started, and destroyed when it exits. We are interested with the system virtual machines which allow the sharing of the underlying physical machine resources between different virtual machines, each running its own operating system [5]. There are several advantages coming with the use of system virtual machines, like the fact that multiple operating system environments can run on the same computer, in strong isolation from each other or that the virtual machine can provide an instruction set architecture different from that of the real virtual machine. The software layer providing virtualization for system virtual machines is called a virtual machine monitor or hypervisor. Due to the fact that it is an important part of datacenter administration having virtual machines in the role of tasks, hypervisor description will be detailed in the followings.

#### Hypervisors

The hypervisor, also called Virtual Machine Monitor (VMM) provides the guest operating system a virtual platform and monitors their execution. Despite the fact that the virtual machines can commonly used resources, the failure of one virtual machine won’t produce the failure of all the other virtual servers running on that machine. The isolation ensured by the hypervisor is one of the main features of virtualization, which brings it to the top of technologies to be used in modern datacenters. The hypervisors are split into two categories: software and bare-metal. Software hypervisors need a host operating system to run on and have lower I/O performance due to the overhead resulting from the hypervisor-host OS communication. Bare-metal hypervisors received their name from the fact that they run on “bare metal”, needing no host operating system. Actually the server must be formatted before this hypervisors are installed. This close connection to the underlying hardware brings better I/O performance and is also faster due to the removal of the layer introduced by the operating system.

Microsoft Hyper-V is the hypervisor which is the most present one in the data centers all over the world. It can be run on an x64 version of Windows Server 2008, the R2 version having the live migration feature enabled. In order for Hyper-V role to be enabled for a windows server, the processor needs to have hardware assisted virtualization. This is available for processors that include a virtualization option (Intel VT or AMD Virtualization). Live migration is supported with the use of cluster shared volumes (CSV). This feature is extremely important for enabling the movement of a virtual machine from a server to another, from efficiency reasons. Live migration of virtual machines from a server to another is done automatically for situations in which the node (the Hyper-V server) fails. In this situation, each virtual machine running on the failed node may migrate to other live nodes independently of other virtual machines. Due to the fact that this diploma project is undertaken within a larger group with different missions, we have chosen Hyper-V as a hypervisor, it providing both a high-level and a low level API. In there are presented on short two other hypervisors, one which offers full virtualization and one offering paravirtualization, together with a description of paravirtualization and comparison to full virtualization.

VMWare ESXi is a “bare metal” hypervisor, meaning that it doesn’t need to run on top of other operating systems. This implies a lower overhead and a better control and granularity for allocating resources (CPU time, disk bandwidth, etc.) and a considerable increase of security. VMWare ESX and VMWare ESXi offers advanced resource management capabilities to improve performance and increases consolidation ratios. Both Hyper-V and VMWare ESXi offer a full virtualization approach which allows datacenters to run an unmodified guest operating system, thus maintaining the existing investments in operating systems and applications and providing a nondisruptive migration to virtualized environments. On the other hand, the paravirtualization approach modifies the guest operating system to eliminate the need for binary translation. Therefore it offers potential performance advantages for certain workloads but requires using specially modified operating system kernels [6]. The Xen open source project was designed initially to support paravirtualized operating systems. While it is possible to modify open source operating systems, such as Linux and OpenBSD, it is not possible to modify “closed” source operating systems such as Microsoft Windows. It is also not practical to modify older versions of open source operating systems that are already in use. As it turns out, Microsoft Windows is the most widely deployed operating system in enterprise datacenters. For such unmodified guest operating systems, a virtualization hypervisor must either adopt the full virtualization approach or rely on hardware virtualization in the processor architecture [7].

#### Virtual appliances

A software appliance is a full application stack containing the operating system, the application software and any required dependencies, and the configuration and data files required to operate. Everything is preinstalled, preintegrated and ready to run. Software appliances come in the form of a file which can be a virtual machine image, an ISO, a USB key image, or an Amazon EC2 AMI. They run a JeOS(Just Enough Operating System) is a customized operating system that precisely fits the needs of a particular application (Ubuntu JeOS, OEL JeOS, SUSE JeOS, OpenSolaris JeOS, OpenSolaris JeOS, Orange JeOS, and Windows Server Core).The virtual appliances, a sub-class of software appliances, add to the advantages of software appliances the benefits of virtualization.

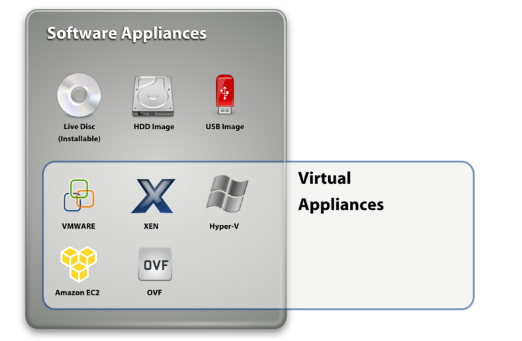


Figure 2.1: Software Appliances Taxonomy- © Novell Inc

The biggest advantage of virtual appliances is that there is a considerable energy economy because we don’t have any more tasks waiting one for another and redirection of conflicting tasks, being an alternative to massive changes and increased complexity on the software stack. Virtual appliances allow for rapid movement of virtual appliances between physical execution environments, provide an improved isolation between several appliances sunning on the same server and improve fault tolerance .For tasks that need to be sent to a data center and run on a server, a virtual appliance can be created with all the needed applications and only the needed part of the operating system and sent to be run secure and isolated from other tasks.

In terms of solutions to energy efficient datacenters, software appliances are a feasible alternative to virtual machines, considering that by having all the needed software and no more than that on top of a lightweight JeOS the time for programmatically creating virtual machines with different characteristics is reduced to 0 and the dimension of the virtual machine is reduced. Not all the hypervisors are able yet to host virtual appliances, therefore this is a subject for implementation after virtual appliances leave the research state and people get acquainted with them.

### Hyper-V WMI Provider

For monitoring virtual machines in a network of servers running Hyper-V, the Hyper-V WMI Provider is being used. It is a high level API, giving information about virtual machines and their status. It enables developers and scripters to build custom tools, utilities and enhancements for the virtualization platform, managing all aspects of the Hyper-V Services [8]. Most functions in this API are available in Basic, PowerShell, C# and C++, therefore the limitations from the programming language point are not too high.

## Autonomic Features

In a manifesto in 2001, IBM invites the world, their customers, competitors and colleagues to accept the Grand Challenge of building and deploying computing systems that regulate themselves and remove complexity from the lives of administrators and users. On short, they consider that the new Grand Challenge in computing industry is the overgrowing software complexity both in terms of management and in terms of maintenance. They believe that the growing complexity of the I/T infrastructure threatens to undetermine the very benefits that information technology aims to provide. Human intervention and administration to manage software complexity is starting to be overwhelmed. It is estimated that at current rates of expansion, there will not be enough skilled IT people to keep the world’s computing systems running.

Considering the fact that “in the evolution of humans and human society, automation has always been the foundation for progress” [9], IBM states that it’s time to design and build computing systems capable of running themselves, adjusting to varying circumstances and preparing their resources to handle most efficiently the workloads we put upon them. These autonomic systems must anticipate the needs and allow users to concentrate on what they want to accomplish rather than figuring how to rig the computer systems to get them there [10].

They give four directions of approach in terms of self-management, described bellow: self-configuration, self-healing, self-optimization and self-protection. Systems having all these capabilities are also called CHOP systems or self-\* systems. This thesis presents a self-healing approach for the datacenter room, and a self-optimizing approach for having a datacenter which consumes the optimum amount of energy.

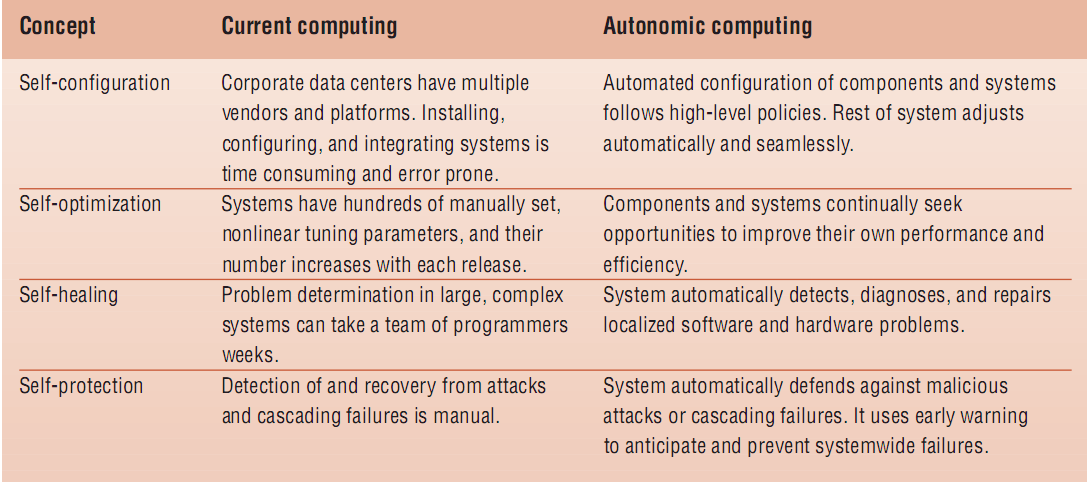


Figure 2.2: Four aspects of self-management © IBM

## Context Model

The most relevant context modeling approaches are presented in [11]. They are classified by the scheme of the data structures which are used to extract information: key-value models, markup scheme models, graphical models, object oriented models, logic based models and ontology based models. Out of these, the logic based model and ontology based model are the ones that store information on which it can easily be reasoned upon. The logic based model is a very formal model, defining conditions on which concluding expression or facts may be derived from a set of other expression or facts. Ontologies are of the most promising instruments to specify concepts and interrelations [11]. They are suitable both for projecting parts of the information used in our everyday life onto a data structure utilizable by computers, and for reasoning in that information with the help of a rule language layered on top of RDF. This is why the chosen context model for our context-aware system is an ontology model. The ancient Greek philosopher Parmenides of Elea was among the first to propose an ontological characterization of the fundamental nature of reality. An ontology together with a set of individual instances of classes constitutes a knowledge base. In reality, there is a fine line where the ontology ends and the knowledge base begins. For creating and modifying the ontology, the Protégé ontology editor and knowledge-base framework is being used.

## Mobile Agents – JADE

Java Agent DEvelopment(JADE) Framework is a mobile agents framework which simplifies the implementation of multi-agent systems. A mobile agent is a software agent with features like autonomy, social ability, learning and mobility. Mobile agents are extremely important in context-aware applications, each agent taking a responsibility and acting independently of the other agents. They can communicate and can be distributed on different machines, or can move even at run-time from one machine to another and resume the execution on the new machine.

Chapter 3

# Related Work

Self-\* and context-aware systems have become leading domains in terms of research. The self-\* initiative has been started by IBM, in the manifesto from 2001, stating that developing autonomic systems is the new challenge in IT [9]. For a system to be self-\* it has to be aware of the context in which it has to function. Therefore, for creating a self-adapting datacenter, it has to be aware of the surrounding context (waiting tasks, temperature, and humidity) and of the energy context (servers’ power consumption, resources used by each deployed task). It will also need to have policies for it to know how to adapt in new situations. On the other hand, considering that the user will tell the datacenter how many resources he would need, and that the user tends to ask too much therefore the datacenter can be in a situation where it can’t fit the task on none of the servers, a negotiation process is needed for SLA. In the following subchapters, related work is presented for all the above mentioned problems.

## Context Aware Systems

Ubiquitous Computing is a term coined by Mark Weiser around 1988, and refers to the seamless integration of devices into the users’ everyday life. The appliances should be hidden into the background, to make the user aware of his tasks and priorities rather than computing devices and technical issues. One field of pervasive of ubiquitous computing is the context-aware (sentient) computing. Context aware systems operate without necessary user intervention, by taking environmental context into account. Smart rooms, which can be seen as intense ubiquitous computing environments, are a step toward Weiser’s [12] vision. Currently, there are already some smart rooms functioning at different organizations, like MIT’s Intelligent Room[13] or Stanford’s iRoom Project [14].

What seems to be a problem in context-aware computing is the privacy of people whose activity is monitored by pervasive computing systems and on whose behalf actions are being undertaken. This is why, in [15] Hong and Landay describe an architecture for privacy sensitive ubiquitous computing which addresses the issue generated by people’s concerns about the strong potential for abuse over a potential lack of control. Confab is a toolkit for facilitating the development of privacy-sensitive ubiquitous computing applications. Confab provides a framework for ubiquitous computing applications, where information is collected and processed as much is possible on the end-user’s computer. In this way, the user can specify how much of the information on his computer is available, and for what purposes. Confab therefore gives “an extendable design that provides a software architecture support for building privacy-sensitive ubicomp applications that are optimistic, pessimistic, and mixed-initiative” [15].

Currently most attempts to use context-awareness within ubiquitous computing have been focusing on the physical elements of the environment, or the user’s device. In contrast with that, a new context-awareness direction towards capturing the cognitive elements of a user’s context is explored by Prekop and Burnett with a conceptual model of Activity Centric Context [16]. The focus of the activity-centric view of the context is on the information surrounding the performance of an activity undertaken by an agent. The activity-centric view has, as its name states, the agents and activities in the centre of the entire perspective on the context. By monitoring and storing agent’s activities, each of the activity belonging to a higher lever activity together with its context, Prekop and Burnett’s paper provides a model for supporting complex context-aware applications after capturing agent’s behavior.

For supporting context-aware systems different ontologies and languages have been described, with the purpose of easing the pervasive frameworks.

The COBRA-ONT ontology for COntext BRoker Architecture (COBRA) is “a collection of ontologies for describing places, agents, events and their associated properties in an intelligent meeting room domain” [17]. The design of COBRA addresses important issues like supporting resource-limited mobile computing and addressing concerns for user privacy, and contains four essential components: a context knowledge base, a context reasoning engine, a context acquisition module and a policy management module.

In [18] Kagal et al. present a policy language specialized for pervasive systems. Rei is a policy language based on deontic concepts, which can be used to describe several kinds of policies. With its help, security policies can be described, restricting access to resources of organizations. It can be used to create actions on resources and describe restrictions and availabilities for users in an organization or defining conversation policies very important in autonomous environments. All of the above possibilities makes Rei a versatile and expressive policy language especially for context-aware computing systems, it being a branch of pervasive computing.

## Energy Aware Systems

For decreasing datacenters’ energy consumption, many solutions have been approached. Brogetto and Stolf describe in [19] an autonomous system which through virtualization and consolidation, manages to decrease energy consumption. They use TUNe as an autonomic job scheduler, and find the best consolidation solutions for reducing energy consumption. What differs from our approach is that we employ a learning algorithm for finding the solution, and that we also use as consolidation action the move actions, for moving virtual machines from one server to another.

In [20] it is used a machine learning approach with dynamic backfilling to optimally distribute tasks in a datacenter. Their approach applies some scheduling policies that reduce the number of unused machines according to the workload needs in each moment, and decide task placing and reallocation in order to compact jobs in the lowest number of machines with-out degrading their service level agreements (SLA). Tasks are regarded as normal applications, and no virtualization is undertaken. Also, the service level agreements aren’t being negotiated. The way in which one can break the SLA agreement is seen as a policy constraint.

HP scientists present in [21] an energy aware grid that is intended to provide global utility infrastructure, managing both energy efficiency matters and thermal issues in datacenters. Workload placement decisions are being taken considering energy coefficients and depending on them the resource co-allocator chooses Globus Resource Allocation Manager (GRAM) as destination for the workload. In addition to that, HP scientists have also designed a physical infrastructure for supporting a low energy consumption of the entire grid.

Srinivasan et al. present in [22] an approach which uses a Swarm Intelligence based approach (SITA) for Task Allocation and scheduling in a dynamically reconfigurable environment such as the computational Grid. An ant colony optimization technique is being used for optimal resource discovery in the Grid, with great adaptation on extraordinary situations like node failure, link failure, and congestion. With the role of ants some distributed agents are being used, which are working in parallel and independently of each other. They take into account both cost and time minimization, for obtaining a balance between these two conflicting items, using a constraint satisfaction based approach to task allocation.

In [23] an energy aware design of service-based applications evaluates KPI (key performance indicators) and GPI (Green Performance Indicators), trying to obtain the best tradeoff between the two of them. The energy efficiency of a service is defined depending on the service’s normal and idle mode, price and energy consumption. Their goal is to maximize the aggregated quality and energy values by considering all possible execution paths and find the best solution at least for the most frequent ones. Considering that both this paper and the thesis which is being presented now are part of the GAMES (Green Active Management of Energy in Service centers) project, there are several similarities between what Ferreira et al. present and the project being currently presented. They both use KPI and GPI for estimating energy efficiency and virtualization for a better encapsulation of tasks, but with different approaches. In [29], Ferreira et al. emphasize that the energy consumption is also a service quality problem. Through this approach, they tackle the energy efficiency problem at the service level as a nonlinear Service Concretization (SC) problem, considering both infrastructure characteristics and quality of service requirements. For a better description of the context, a the energy efficiency GPI metric is defined, its computation being based on attributes like execution time and energy measures like energy consumption.

## Self-\* Systems

Computing systems have reached a level of complexity where the human efforts necessary for maintenance and development are getting out of hand. Just as in 1920, when human operators weren’t enough to work with the switchboards, the solution is automation [24]. Autonomic computing tries to simulate the behavior of the human body, in which the autonomic nervous system takes care unconsciously of reflexes, the digestion, rate and depth of the respiration and other such processes. The term autonomic computing was first used by IBM in 2001 to describe computing systems that are self-managing, just as the human body auto-manages itself without us even thinking about it. Huebscher et al. present of review of the autonomic computing domain, and approaches that have been taken in this domain. Self-management is described as being implemented with the help of utilities for estimation of system’s behavior, through reinforcement learning or Bayesian techniques.

Carzaniga et al. propose in [25] the idea of automatic workarounds as a self-healing method for software automation. Giving a failure event, the system can automatically execute one or more alternative sequences that are known to have an equivalent behavior. A method of self-healing is described by giving formalization to the workaround problem, and finding solutions for it. A general architecture for automatic workarounds is proposed, and a mean of workarounds representation and run-time usage is described.

In [26] a solution for an integrated datacenter using power management is presented, employing server management tools, appropriate sensors and monitors and an agent based approach for achieving power and performance objectives. By intelligently turning off servers when the workload is small, there can be achieved over 25% power savings over the unmanaged case without incurring SLA penalties. By using a testbed containing IBM BladeCenters and employing a realistic HTTP traffic from a NASA web site they prove the efficiency of the solution.

So far, proposals have been made for self-healing computing systems, but without an actual proof-of-concept tool for creating self-\* software. To overcome this problem, IBM has developed a demo toolkit which addresses the problem of self-healing and self-optimizing software [27]. IBM Emerging Technologies Toolkit (ETTK) explores these two key aspects of autonomic computing both in a visual and in a practical way. It demonstrates that we already have almost all the necessary knowledge for creating such tools and that this technology could be a reality today.

## SLA Negotiation

Katsuhide Fujita et al. propose in [30] a Distributed Mediator Protocol for securely finding Pareto optimal solutions. They employ “approximated fairness”, using deviation for measuring the difference of utilities achieved by agents and a Nash bargaining solution maximizing the product of each agent’s utilities in our model. By defining a non-linear utility function in a complex utility space for multi-issue negotiation the estimation of agents’ satisfaction is quantified.

In [31], Chen et al. describe an automated negotiation mechanism which combines a game theory approach and a co-evolutionary approach for finding a Pareto optimal solution. In spite of the fact that the agents don’t know other agents’ strategies and payoffs, solutions that complies Nash equilibrium and Pareto efficiency concepts are discovered. The proposed method adopts genetic algorithms to optimize agent’s negotiation strategies generating payoff matrices, and after that finding the optimized point through a game-theory approach.

Brandic et al. describe in [32] a way of establishing adaptable, versatile and dynamic services by undertaking a service mediation process, considering negotiation bootstrapping. This work is being achieved in the context of Foundation of Self-Governing ICT Infrastructures (FoSII) project. They approach the gap between QoS methods and Grid/Cloud services through describing an architecture for service management, with components for meta-negotiations and SLA-mappings. The negotiation is being described in a document, having pre-requisites with negotiation terms and authentication data, and an agreement which concludes the negotiation, therefore having the term meta-negotiation.

The trust negotiation Strategy based on Negotiation Petri Net (SNPN) is described in [33], modeling the policies participating a trust negotiation. In automated trust negotiation, the access to a resource is given based on attributes rather than identification, policies participating negotiation being modeled as Negotiation Petri Nets in this paper. The SNPN is a complete negotiation strategy, having constructed both a Negotiation Petri Net and a Reverse Negotiation Petri Net and thereby being a good strategy for automated trust negotiation. Credential exchanges in this solution are being considered until both parties know that there exists a successful negotiation, and at the end only credentials in the safe disclosure sequence are disclosed, this way avoiding the disclosure of any credentials that are not needed for successful negotiation.

Multi-issue negotiation is an important problem in everyday life, existing numerous situations in which, for example, not only the price is to be negotiated, but also quantity, delivery time or other issues. For multi-issue negotiation the solution space is n-dimensional instead of the normal one-dimensional space which single-issue negotiation relies on. Most of the existing work has been focused on independent issues on multi-issue negotiation, and on sequential negotiation for multi-issue problems. In the followings, there are presented several solutions to these problems.

Guoming Lai et al. present in [34] a decentralized model for multi-attribute negotiation, being able to be applied to the situations where agents are self-interested. Agents can negotiate multiple attributes simultaneously, and always consider the Pareto optimal outcome. The model allows a proposer to make several offers in each round. The responder can choose an offer, or can reject all of them, exchange roles with the negotiator and proceed to make offers to the former negotiator. This model can be applied to incomplete information scenarios, where the agents have information only about themselves and not about other agents. Despite the fact that they don’t know the others’ agents utility functions or negotiation strategies, the generalization property is essential in this model, it being applied successfully in systems with complex utility functions in continuous negotiation domains.

In [35] an analysis of feasible solutions for multi-issue negotiation is presented. Fatima et al. present a comparison between a Package Deal Procedure (PDP) and a Simultaneous Procedure (SP). They define -Nash equilibrium strategies for both the PDP and SP, adding rules for reaching an outcome that maximizes social welfare for the simultaneous procedure since it results in multiple equilibria. From comparing the PDP and SP with the formulated strategies, it results that package deal procedure always generates optimal outcomes, while the simultaneous procedure may be better for one of the two agents, and also increases social welfare. In [36], they consider the case where there are time constraints, uncertainty about the opponent’s negotiation parameters and interdependence between issues, and compare several approaches for the considered negotiation strategies, in terms of time of agreement, time to compute equilibrium, Pareto optimality and uniqueness of the equilibrium. This paper is thoroughly documented giving also a review on possible negotiation types, and presents several theorems and conclusions regarding package deal procedure, simultaneous procedure and sequential procedure.

Chapter 4

# Analysis and design

This part describes the algorithms used for handling SLA negotiation and optimal task distribution in a datacenter having as components servers, a storage server and a global loop which coordinates the entire task flow.



Figure 4.1: GAMES datacenter infrastructure

The server cluster contains several servers, on which we can operate actions like move virtual machine, deploy virtual machine or remove a virtual machine from specific server in the cluster to another one. The environment is also controlled through an algorithm providing self-healing capabilities, which enforces some temperature and humidity policies on the datacenter environment. I will describe the concepts involved in the design of the self-adapting datacenter in the followings.

## Conceptual architecture

The conceptual architecture below represented in figure 5.2 gives an overall view of the design of this system. There are two main parts, one handling the self-healing capabilities of the environment, and one handling the self-adapting capabilities of the energy-efficient datacenter.

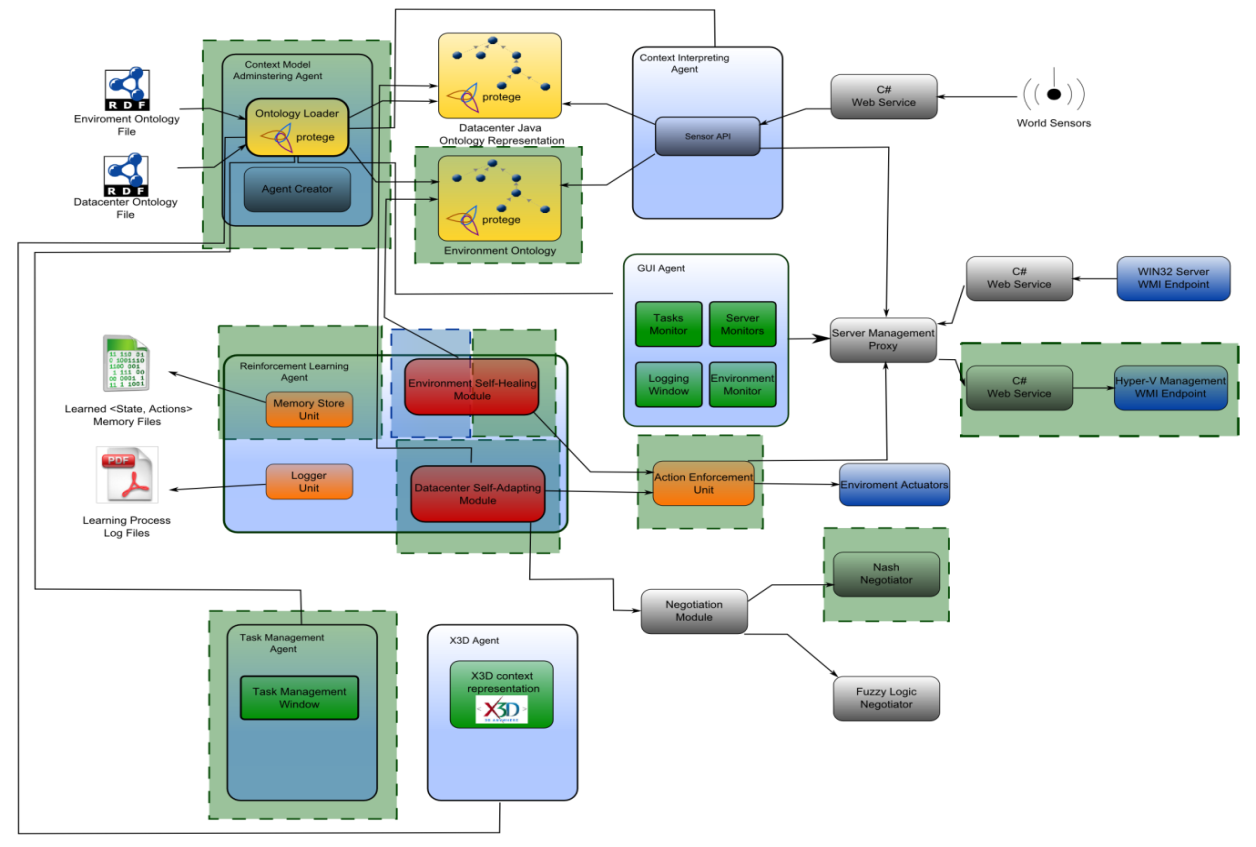


Figure 5.2: Conceptual architecture of the system

The part emphasized in green is developed by me and will be further detailed, while the remaining components of the architecture are developed by the author of [39]. They will be mentioned and properly referenced when needed. The component containing the Environment Self-Healing module is the result of the work of both me and Daniel Moldovan [39], for the CONSENS research project [3] and therefore it is a joint development effort, also described in [39].

The environment ontology file describes the environment in which the datacenter resides. It describes resources and actuators associated to them, together with available actions for each resource. The datacenter ontology, describes the servers and tasks and admissible values for all of these. It follows the same model as the environment ontology, the <R, A, P> model. The two ontologies are loaded and forwarded to other agents by the Context Model and Administrating Agent (CMAA). The CMAA is also responsible for creating agents and sending them the appropriate information. It creates the Reinforcement Learning Agent (RLA), which implements the self-\* behavior for the service center as well as for its room, the Task Management Agent (TMA) which is responsible for adding and removing tasks from the datacenter, the X3D and GUI Agents responsible for giving a visual representation of the simulation, and the Context Interpreting Agent, responsible for getting and interpreting information from the sensors. The actions found by the RLA are enforced in the datacenter through Hyper-V Management WMI endpoint component, while the ones found by the environment behavior of the RLA are enforced through environment actuators.

## The Self-Healing Behavior

The self-healing behavior described in [2] finds an action plan for bringing the context to a healthy state. Enforcing these series of actions on the broken context brings it to an optimal state. There are several concepts defined in this section which ease the algorithm description.

The context is described under a <R, A, P> (Resources, Actors and Policies) form, the entire description being kept in an ontology. The Resources refer to both sensors and actuators associated to those sensors, for applying actions which will modify resource’s value. The actors are in this case the actions associated to each resource, while Policies are conjunctions of sensor’s states, which describe the optimal state for that sensors’ association.

For evaluating the closeness of the context to the optimal context described through the policies, the entropy concept is being defined. When the entropy is above the defined threshold, the self-healing algorithm will start.

### Algorithm Description

Our approach for the self-healing process consists of the following steps (Figure 4.2):

1. **Check if the current context has been encountered so far.**

If in the memory file we have a context equal to the current context, we take add the associated sequence of actions to the current sequence of action (at the beginning, NULL). If after the sum of ***recheck times*** for the actions, the system is not healed we ***rollback*** the sequence of actions and go to step 2.

If we don’t find any sequence learned, we go to step 2.

1. **Select the broken policy with the largest entropy contribution**

Search through the policies having entropy contribution larger than zero, and select the ***policy with the largest contribution to the entropy*** (this means that it is either the most important, or the most broken, or a combination between them). Go to step 3.

1. **Take the next Inter-Independent Resources Group (IIRG)**

Select the next IIRG for which we find a best sequence of actions through a reinforcement learning process, taking as reward a function inversely proportional to the entropy. Go to step 4.

1. **Add the found sequence of actions to the actions sequence list**

We take the found sequence of actions and add it to the sequence of actions list. If there are no more resources to be brought to their optimal states for this policy, go to step 5, otherwise go to step 3.

1. **Add the found actions to the current sequence of actions**

Take the list of sequences of actions found for the current policy and add it to the current action sequence associated to the entire context. If the entropy is still larger than the threshold, go to step 1, otherwise go to step 6.

1. **Memorize the (sequence of actions, Step1 context) association**

Associate the current list of sequences of actions to the initial broken context.

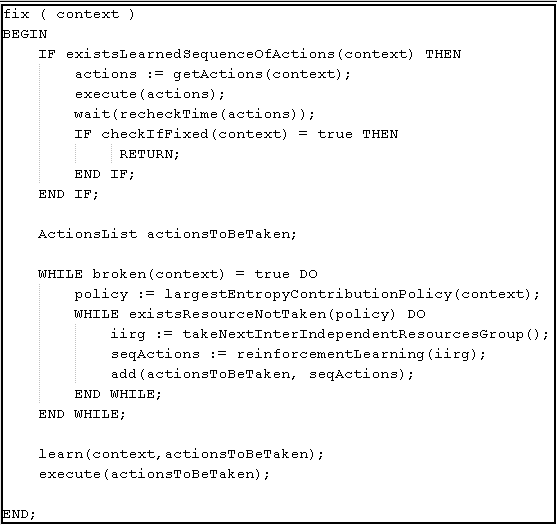


Figure 4.2: Self-healing algorithm

If while searching for the sequence of actions which would bring the context to an acceptable one, there appears a new broken policy, with a contribution to the entropy much larger than the one that we are fixing, it goes first in the queue with broken policies, the current process of healing the policy which is supposed to have had the greatest contribution to the entropy is suspended and the process of finding the sequence of actions is being started for the new policy.

The self-healing process is described as a sequence of steps, emphasizing on few concepts which are the baseline of this algorithm.

### Recheck Time

The recheck time is a physical property which defines each resource. It describes the needed time for the consequences of an action to be spotted on the current action. For example, we need some time for the temperature to be lowered after we have started the air conditioning. In this context, we have the temperature sensor with the role of sensor, the air conditioning unit with the role of actuator and the time that we need to wait with the role of recheck time.

Recheck time is a constant associated with each resource, referring to the time needed for an action to have the expected effect. If a resource will have a number of actions to be completed, the recheck moment will be after (nrActions+1)\* recheckTime. The default for the recheck time is zero, being valid for resources on which actions have an immediate effect.

### Rollback

After we enforce the found sequence of actions list on the current context, we check they produced the expected effect. If this is not the case, each action will be rewinded in the reverse order with regard to the order in which they were taken. Then the system continues by searching for a sequence of actions that will have the desired effect.

### Inter-Independent Resources Group

#### Dependency between Resources

The dependency of a resource to another resource is visible when the actions taken upon a resource affect the other resource to. The dependency relation can be seen as a dependency function:

0, if no path exists between i and j

, if k is the length of the shortest path from resource i to resource j

The dependency is a **transitive** relation:

(4.1)

Therefore if resource i is dependent on resource j, and resource j is dependent of resource k, resource i will be dependent of resource k. For example, in a smart laboratory context, if the air conditioning affects the temperature, and the temperature affects the humidity, the temperature will indirectly affect the humidity, so the resource humidity is dependent on air conditioning with a degree two of dependency.

There are two dependency types: direct dependency and indirect dependency. Direct dependency means that a change of the first resource would lead to a change in the other resource. On the other hand, indirect dependency describes a sequence of resources in which each resource depends on the next resource, which means that the first resource indirectly depends on the last resource.

(4.2)

The *dependency graph* is a direct weighted graph, having as nodes the resources and as arcs dependency relations and as weights the dependency degrees. The *dependency degree* between the resources is the inverse of the length of the shortest path between the nodes.

#### Inter-Independent Resources Group

Two resources are *inter-independent* with each other iff D(i,)0D(j,i)0. That means that doesn’t depend on and j doesn’t depend on i.

The *Inter-Independent Resources Group* is a set S of resources for which the following properties are true:

* S,
* S,–resource in current policy such that weight( weight(
* The set S is maximal.

In other words, any resources belonging to a current IIRG are inter-independent, have the highest weights and form a maximal set.

### Mathematical Description of the Context

Each resource and each policy in our context have an associated *weight* describing the importance of the resource or policy in the current context, in relation with the other existing resources or policies. The policy weights are represented as a (1, n) matrix, which has on each i-th row the weight of the policy i.

The weight of a policy reflects how much we need for it to be respected:

W = . W- Weight Vector

. – weight of the policy i

On the other hand, resources importance in a policy depends both on the resource and on the policy. This is why, for resources weights we have a two-dimensional matrix:

…

…

…

R = … … … … … R-the resource importance matrix

… … … … … R [i,j] = the importance of resource i in policy j

… … … … …

…

In the context of a policy, each resource will have a value which indicates how much the resource respects that policy. The value of a resource is formalized as follows:

…

…

V= … … … … … V-the matrix of values

… … … … … V [i,j]=the value of resource j in policy i

… … … … …

…

One element of this matrix, V [ij], will be a function dependent on resource j and policy i, evaluating how much does the resource j respect the policy i. This function will be defined as:

v:



0, rj has discrete states, state(rj)=desiredstate(rij)

v (i,j) k ,rj has discrete states,state(rj)!= desiredstate(rij)

|state(rj)-desiredstate(rij)| , otherwise

Having these matrices, the state of the context at a given time is defined by P\*R\*V.

#### Entropy

According to Wikipedia, “the entropy is a measure of how disorganized a system is” [5]. Extrapolating on the second laws of physics, the entropy has become an important concept in cosmology and even everyday life. As the universe is supposed to be a closed system, by the Second Law of Thermodynamics, its total entropy is constantly increasing. This has lead to speculations that the universe is fated to a heat death. In the study “Natural selection for least action” [37] Ville Kaila and Arto Anilla describe how the second law of thermodynamics can be written as an equation of motion to describe evolution, showing how natural selection and the principle of least action can be connected, evolution exploring possible paths to level differences in energy density and increasing entropy most rapidly.

Extrapolating on the problem of thermodynamics and entropy, the same concept is defined for our environment. The entropy is the measure for how disorganized our system is. It describes the way in which the environment policies are fulfilled. For our context described above, the entropy will be:

E= ∑ \*. (4.3)

Where:

* **-** the weight of i-th policy
* - the importance of resource j in policy i
* **-** the value of resource j in policy i.

The entropy can also be expressed as: E=∑P\*R\*V (4.4). In this relation, the product P\*V describes the respectability degree of the policy, while the product P\*R gives the total weight of a resource within the environment, considering all the described policies.

#### Policy Contribution to the Entropy

The entropy contribution of a policy to the entire system entropy describes the way in which the incomplete fulfillment of a policy affects the entire context.

The policy contribution to the entropy will be defined as:

=\*(4.5)

Where represents the contribution to the entropy of the ith policy, is the weight of the policy i, and\* is the respectability degree of the policy i in the current context.

#### Entropy Threshold

For describing an acceptance limit for the context, we introduce the concept of *entropy threshold.* This threshold is a numeric value describing a context which isn’t acceptable anymore, and therefore we need to start the self-healing algorithm. The self-healing algorithm will start each time the entropy will pass the entropy threshold.

Depending on how strict we are about the way in which the system should behave, we have a few possibilities for deciding when the system should consider itself broken or “disturbed”. There are several variants:

* At its stricter version, the system is only healthy when all the policies are perfectly respected. This is the case when T=0.
* For a more complex system, we can attach to each policy a value which denotes how much of it should be respected, and compute the threshold as :
  + - (4.6)
* The threshold can be dependent also on a weighted average of the policy contribution to the entropy, and add it to the minimal entropy. This solution would also improve computational time and assure that the system can go out of its way as long as an important policy is not broken
  + - (4.7)

### Algorithm Complexity

The algorithm complexity is a breath first search complexity, therefore having a worst case complexity of, where b is the branching factor and d is the depth factor. The branching factor is described by the following relation:

(4.8)

Where defines the number of resources existent in the context, defines the maximum number of actions associated to each resource while is the branching level at a moment of time. From each branch level, we are recursively defining more and more branching points, and therefore the branching factor grows with the reached.

The depth of our search tree will be a variable of the type of actions and resources of our context. For example, the difference between a “set” action and an increase by one action is essential for the algorithm running time when we have to get from a resource value of 0 to 50000. The set action takes a level in the tree of the algorithm, because with one action the expected result is reached, while the increment action takes 50000 levels in the tree of our algorithm, meaning an exponential increase of 50000 in the complexity of our algorithm.

Therefore, the depth of our tree depends on the possible values of the resources (range of resource values) and the way in which the respective actions are defined.

The depth factor is:

(4.9)

Where defines the number of actions associated to each resources. The depth factor is described by the total number of actions associated to each resource. The context is guaranteed to reach a solution before taking all the possible actions, each action being taken a maximum number of times defined by.

The above branching and depth factors describe the worse case situation, which will rarely appear due to the fact that policies and resources are weighted, and there exists an entropy variable describing the degree of disturbance in the system, and the way in which the system progresses throughout the breath first search path. Therefore we have that the complexity is:

(4.10)

## Datacenter Self-Adapting Behavior

An autonomic datacenter which can lower the amount of energy consumed without disregarding the service level agreement needs to adapt itself to all kinds of new situations, to find solutions to new problems and keep them in mind for the future.

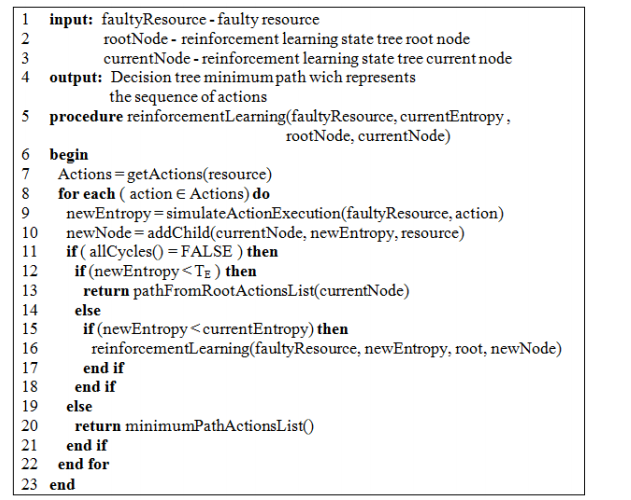


Figure 4.3: Self-adapting Algorithm

The self-adapting behavior is implemented through a reinforcement learning approach. The algorithm used for enforcing the environment self-healing behavior is adapted for creating the self-adapting the self-healing behavior of the datacenter.

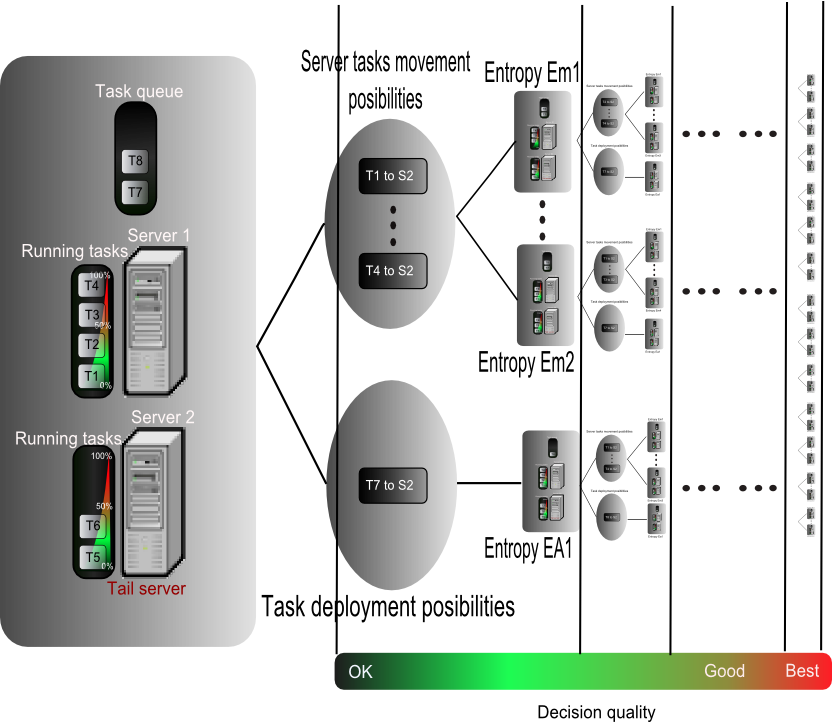


Figure 4.4: Reinforcement Learning Flow

At each step, the algorithm simulates the execution of each of the possible actions, and evaluates the expected reward and entropy for the resulting states. From all the resulting states, and the states that haven’t previously been expanded, the system chooses the one with the largest reward. From this new current state, the system is retrying all the possible actions and expanding them.

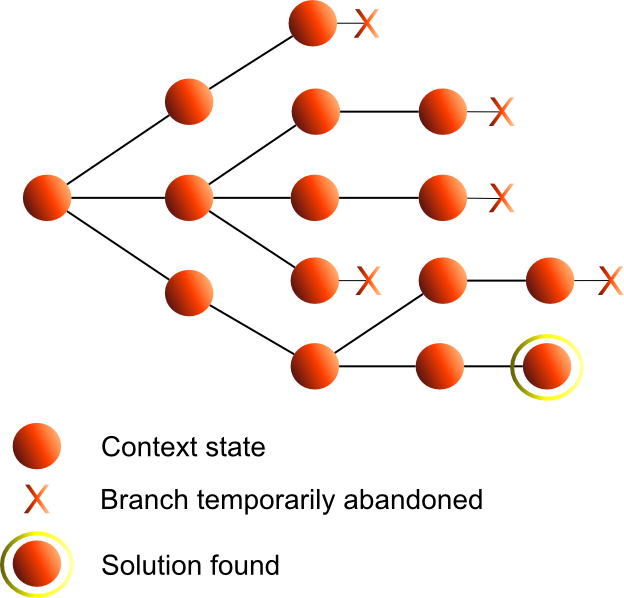


Figure 4.5: Reinforcement Learning Algorithm- Contexts’ Expansion

By always expanding the leaf with the largest reward (figure 4.3), the system is guaranteed to find the best sequence of actions which would bring the system as close as possible to the 0-entropy state.

### Datacenter Context Mapping

Any autonomy can happen only when the system is aware of what is happening around it. This is why, before being a self-adapting system it needs to have the form of a context-aware model, and be able to understand and reason through this information. The model presented in [4] is being used for creating the context-aware part of the system. In order for us to be able to use this model, we need to map the datacenter elements on the context-aware model.

The <R, A, P> model describes a context-aware system, having resources, actors and policies which interact with each-other. Resources are described as physical characteristics of the environment or physical objects with which we can interact. The actors interact with resources and can change the entire context, while policies describe the best state for the context.

Table 4.1: Mapping between <R,A,P> and datacenter context

|  |  |  |
| --- | --- | --- |
| <R,A,P> entity | Self-Adapting context entity | Details |
| Resources | Servers | * All datacenter servers |
| Actors | Actions | * Deploy task * Move task * Send server to hibernate * Wake up server |
| Policies | Energy policies | * Green performance indicators |
| QoS policies | * Key performance indicators |

Therefore, as shown in the Table 4.1, in our datacenter the servers will play the role of resources, the actions applied on those servers will play the role of actors and the QoS and energy policies will be the policies of the context-aware system.

### Entropy

The entropy describes the degree of disturbance in the environment. For the green datacenter context, the entropy reflects both the happiness of the customer, and the happiness of the environment.

The entropy needs to be held as its lowest value, and therefore the definition of the function describing the entropy is extremely important. Since we have defined policies both for Key Performance Indicators (KPI) and Green Performance Indicators (GPI), the entropy will evaluate the respectability degree of each policy.

(4.11)

Where:

* + - represents the weight of the policy (QoS or energy policy)
    - is the weight of the system resource i in the policy j. The resource weight reflects the system resource importance for the policy. If a policy imposes no restrictions to a system resource then the resource weight of the resource for that policy is zero.
    - is the deviation between the current value of the resource, and the accepted range for that resource

For the deviation above between the desired resource value and the existent one, if the optimum values have the form of a range, there exists a deviation when current value is either smaller than the lower limit of the range, or larger than the upper limit of the range.

### Reward in Reinforcement Learning

In any reinforcement learning algorithm, the context’s state is defined by the reward the systems gets for arriving or for being in that state. In this approach, the following function is defined for describing the expected reward:

(4.12)

In the relation 4.9, defines the expected reward for the next state of the context. It is an expression of the reward of the system for getting in the current state, , and a gamma factor multiplied with the transition reward for getting from state i to state i+1. The gamma factor is called a discount factor, enabling the system to try to take the best actions in the early phase of the algorithm. and are numerical values for current entropy and respectively the future entropy, for the case in which the system takes an action with the cost . The variable describes the number of actions taken so far, and is included in the function describing the expected reward because we need to take as few actions as possible, considering that they are time consuming.

### Task History

George Santayana, a Spanish philosopher affirmed in his book [37] that “Those who cannot remember the past are condemned to repeat it.” This is why, besides remembering what actions we have found for bringing the context to an acceptable state, we also need to store the type of tasks that we have received in the datacenter so far, and try to reason on this information.

The obvious reasoning on this information would be a load prediction, for us to know approximately how many servers and what type of servers we need for a time period in the future. The task history component would ease a lot the time wasted on waiting for servers to start, because knowing what will come next we can anticipate the workload and take the right attitude when considering it.

### Tail Server

Using the history component, we are able to predict the task flow and therefore we will be able to wake up servers when anticipating many tasks arriving to the datacenter, and there are not enough tasks available.

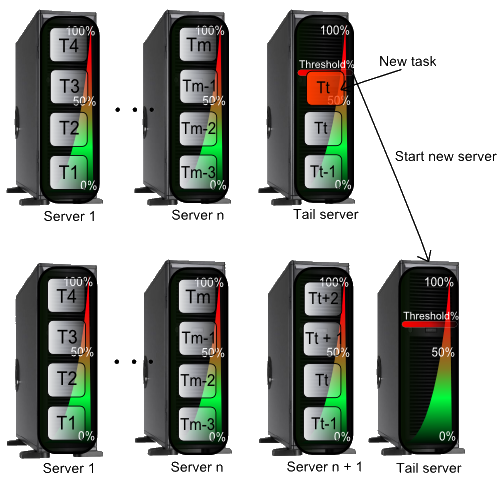


Figure 4.6: Tail Server

Also, when the workload of the tail server grows above its defined threshold, we need to wake up a new server (figure 4.4). The decision on the type of the server to be woken up will be made depending on the task flow predicted by the history component.

### Algorithm Complexity

For estimating the complexity of the algorithm, we need to find the branching factor and the depth factor, knowing that the complexity of a depth first search algorithm is , and that the reinforcement learning algorithm is based on a depth first generation of nodes.

The branching factor is equal to the number of actions that could bring the context into a new state. The number of wake up actions is equal to the number of sleeping servers, the number of sleep actions is equal to the number of online servers, the number of deploy actions is equal to the number of undeployed tasks multiplied with the number of available servers, while the number of move actions is equal to the number of tasks multiplied with the number of available servers.

(4.13)

The depth factor will also be equal to the number of actions, since by doing all the actions available it is impossible not to find the solution.

(4.14)

The worst case complexity of the algorithm will be:

(4.14)

Even though the complexity seems considerable, the worst case situation will rarely be reached due to the reward function which enables the algorithm to rapidly converge to a solution, intelligently, without searching blindly through the entire space.

## SLA Negotiation between Client and Datacenter

Negotiation is a dialogue intended to resolve disputes, to produce an agreement upon courses of action, to bargain for individual or collective advantage, or to craft outcomes to satisfy various interests [5].

In service centers, the negotiation is an essential part since a client will always ask for the most resources possible for the money that he pays. Therefore the system will always get to a situation where the new task won’t fit in any of the existent servers. In these situations, a negotiation process needs to be undertaken in order to lower the requirements of the client in an admissible range, and be able to deploy the task on a server.

Negotiation in this system needs to have under consideration multiple issues, because we have multiple resources which define a server or some task requirements. It also needs to be taken between the entire datacenter and the client represented by the task. There are several possibilities for doing that:

* **“Choose first” solution**

Consider the server which is the closest to the imaginary one described by task requirements, and negotiate between the task and the found server. This solution doesn’t guarantee that there exists a solution to that negotiation and we don’t need to take the next closest server.

* **“Choose last” solution**

Create a possible server from existent servers in the datacenter by taking extremities of the admissible resource ranges, and negotiate between this server and the task. After a solution is found to the negotiation, the returned server will be the one closest to the negotiation solution.

* **Sequential elimination solution**

For a better accuracy, we can sequentially reduce the available servers, after having done negotiations on each resource. This ensures a solution closer to the optimal. The disadvantage of this solution is that we need to make evaluations after each negotiation in order to get the remained servers.

For the first two solutions, we need a mean of evaluating the distance between two servers. The distance between the two servers is visually described bellow, in figure 4.5.

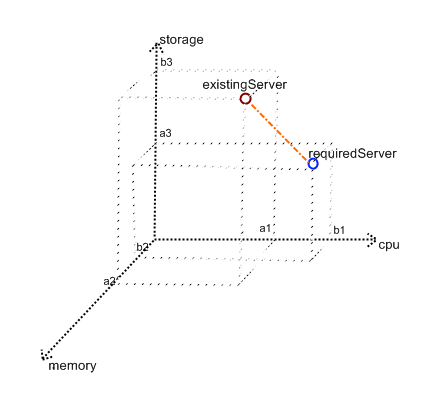


Figure 4.7: Distance between two servers

The server is defined by CPU, memory, storage and other resources. We will denote them a1, a2, a3, and we will have for the existing server the defining list of resources (a1,a2,a3,…) and for the required server the list (b1 ,b2 ,b3,…). For finding the server which differs the least from the required server, we need the server having a minimum distance to the required server (figure 4.5). For this, we can use Gaussian (4.16) Manhattan (4.17) or Chebyshev (4.18) distances:

(4.16)

(4.17)

(4.18)

In this project, there are two approaches for negotiating: with the help of fuzzy logic or through finding the Nash equilibrium.

There are several possibilities for negotiating multiple issues at once:

* **Considering the issues independent one of another**

This solution is applicable to situations where the issues are not interdependent with each other. Each issue is negotiated in the same time, and after that the result is formed of all the results of the negotiations. This is a fast process, but the negotiation result is guaranteed to function only for independent issues.

* **Sequential Negotiation**

This solution is applicable to any situations, since negotiation is made for each issue. The result of each negotiation will be the input for the negotiation for the next issue. After negotiating all issues, the result will be the output of the last negotiation. The issues need to be weighted, because we need to know which the priority for each resource, considering that we have to negotiate for a single issue at a time.

* **Package Deal Procedure**

The package deal procedure is the one giving the optimal negotiation solution. Issues are negotiated as a package, and offers are made in terms of collection of interrelated issues. The offer is being made as a proposal for each issue under negotiation. Agents are allowed to either accept an offer, or reject all of it. Despite the fact that the Package Deal Procedure (PDP) gives always the optimal solution, the time spent in finding the solution is the highest from the three described.

### Fuzzy Logic Negotiation

Fuzzy logic is a form of multi-valued logic derived from fuzzy set theory, to deal with reasoning that is approximate rather than precise [5]. For negotiating, we define two ranges: one is the range requested by the client, another is the range needed by the server to have optimum power consumption together with their satisfaction value represented by the Y axis. Through defuzzification by center of gravity we get a value for the current resource which is acceptable both for the client and the datacenter.

### Nash Negotiation

John Forbes Nash put the basis of the domain called game theory. He proposed a solution concept of a game involving two or more players, in which each player is assumed to know the equilibrium strategies of the other players and no player has anything to gain by changing their options unilaterally. If each player has chosen a strategy and no player can benefit by changing his or her strategy while other players keep theirs unchanged, then the current set of strategy choices constitutes a Nash equilibrium [5].

Nash Theorem revolutionized and basically put the foundation of Game Theory. If we see the current negotiation like a cooperation game between the energy and the QoS, we can also see the Nash Equilibrium as the point in which both the client and the data center are satisfied. There are two kinds of approaches for Nash negotiation: the straight forward technique and the bargaining technique. The two of them will be described as follows.

#### Straight Forward Nash Negotiation

For a straight forward Nash negotiation, we build a matrix having as elements pairs of utility values for the task and respectively for the datacenter, for one resource. The utility functions are:

(4.19)

(4.20)

(4.21)

(4.22)

The negotiated point will be the point with both the maximum utility of the task with report to the value of the server’s attribute and the maximum utility of the server with report to the value of the task’s attribute.

For each resource (CPU, memory, storage) we find the equilibrium value, while we have as ranges the extremity points of what we have in the entire data center (ex. If we have two servers with a CPU desired range of (400, 2000) and one of (200, 1600), the range for the datacenter will be (200, 2000). After finding the equilibrium points for all resources, we take the server which is the closest to the found values.

#### Nash Bargaining Solutions

Another approach to finding Nash equilibrium involves two agents, and offers made from one agent to another. This is also called a **bargaining** process. In Nash bargaining game each of the two players demand a part of some good, and the two players get their demand if the two proposals sum to no more than the total good, otherwise both players getting nothing.

The Nash bargaining solution is guaranteed to reach a Parento optimal solution. In Nash bargaining solution each player offers a deal based on backward reasoning (what he expects from the other to offer at the next move). At each step if the first player refuses the offer made by the second player, a new offer will be made from the second player towards the first player. In this state, their roles will interchange. By this technique and by defining acceptable offers at each step, the negotiation is expected to rapidly converge to a Pareto Optimal Solution.

### Dynamic Policy Concept

After negotiating, the policies for datacenter need to be modified in order to fit the negotiated values. This involves the existence of dynamic policies which will reset to the old values when a new task appears in the task queue.

## Isomorphism of Self-\*

By defining a context-aware model described by <R, A, P> (Resources, Actors, Policies). This model generalizes any environments, from rooms to datacenters and therefore we can use on it the same algorithm searching for actions to enable for self-\* behavior. Therefore, the reinforcement learning algorithm with or without improvements can be used for self-healing and self-adapting capabilities implementation. From this we can generalize and say that by applying reinforcement learning on the <R, A, P> model we can implement any kind of self-\* behavior.

Chapter 5

# Implementation

This chapter handles, as its name says, the implementation issues of the project. In the figure 5.1 we have an overview of the system as a whole. There are emphasized several modules each having its own purpose, and two agents who are focused on certain activities.

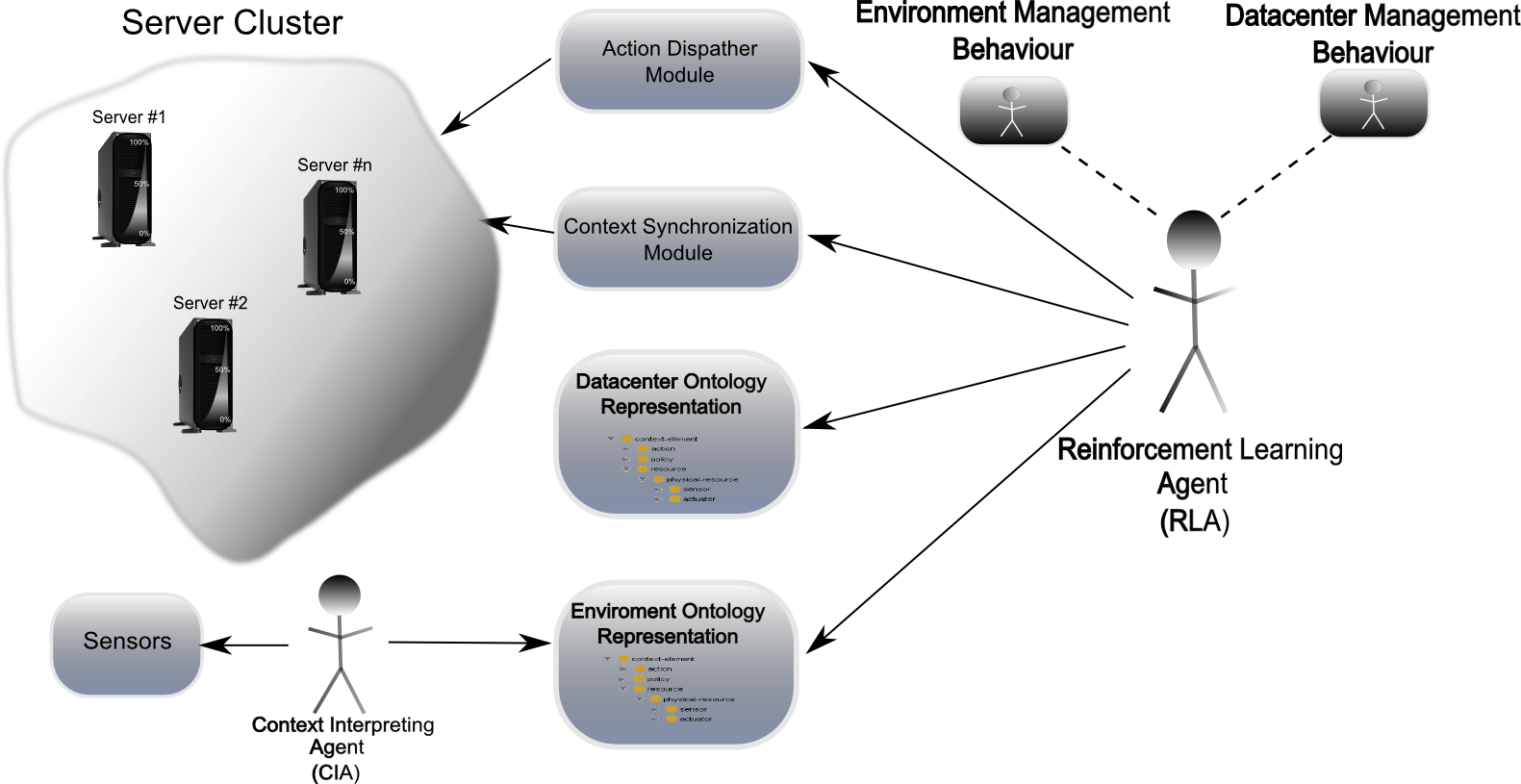


Figure 5.1: View on the System

The server cluster is a typical cluster, as obtained with Windows Server 2008 R2, sharing a common storage, having enabled the failover clustering feature. Sensors, needed for the environment management behavior, are handled by the Context Interpreting Agent (CIA), and information about them is stored in the environment ontology. Information about servers and tasks are held in the datacenter ontology, and handled mostly by the reinforcement learning agent. Context Synchronization module is used for having similarity between what it exists in the real world, and what the Reinforcement Learning Agent (RLA) knows about the world.

The RLA has two different behaviors: one handling the environment, with its characteristics and one handling the datacenter with servers and virtual machines. The Environment Management Behavior is a self-healing module, which implements the self-healing algorithm described in 4.1. It ensures that the temperature and the humidity are in normal limits, for the normal functioning of the datacenter. The Datacenter Management Behavior is a self-adapting module, implementing the reinforcement learning algorithm on the context described in 4.2. It ensures finding the deployment actions for each oncoming task, and assuring consolidation actions like putting the servers on hibernate, or moving virtual machines from one server to another.

The action dispatcher module enforces actions upon the datacenter, therefore realizing the self-adapting mechanism in real datacenters.

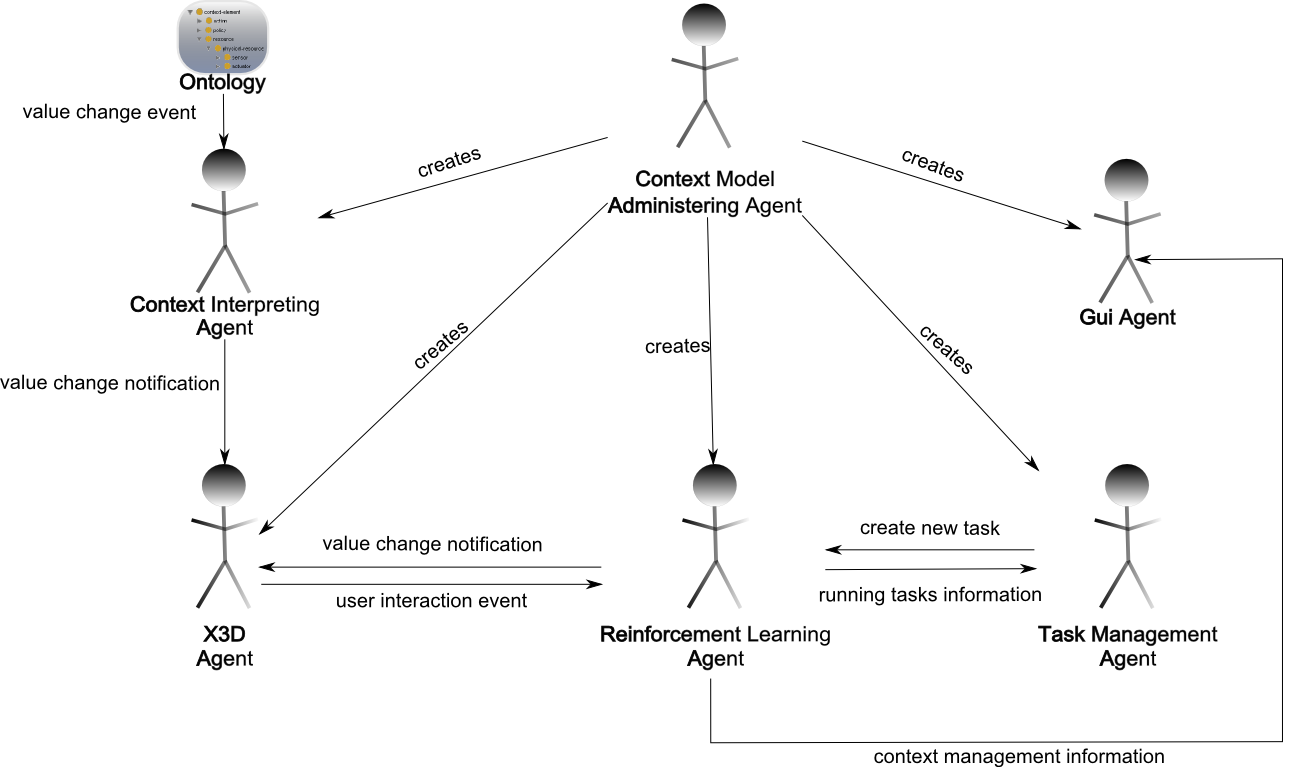


Figure 5.3: Relations between agents

The figure 5.3 describes the relations between agents. All agents are created by the Context Model Administering Agent(CMAA), each with its own purpose. The Context Interpreting Agent (CIA) checks if any value of the ontology was changed, and if so, it sends notifications to the X3D agent. The Reinforcement Learning Agent (RLA) also sends him notification in case the environment was changed by RLA’s agent. The ontology can be changed by an user interaction event, case in which the X3D Agent who caught the respective event, sends a notification to the RLA. The RLA also sends information about the running tasks to the Task Management Agent, who notifies the RLA when a new task was created. Context management information is also sent from Reinforcement Learning Action to the Graphical User Interface Agent (GUIA).

## Context Management Infrastructure Implementation

This system is based on agents, each with its own purpose, which communicate to realize the needed system’s behavior. For this, the Java Agent DEvelopment Framework (JADE) has been used.

The system also needs a way to capture information about servers, and to modify servers’ properties. Since the tasks need to be deployed and moved around the datacenter for optimal task distribution, it also needs an API which allows it to do such actions. We have used Hyper-V WMI Provider, which enables easy virtual machine handling in a datacenter.

Both the JADE framework and the Hyper-V WMI Provider are described bellow, emphasizing the most important issues.

### JADE Overview

The Java Agent Development framework is an enabling technology, a middleware for the development and run-time execution of peer-to-peer applications which are based on the agents’ paradigm and which can seamless work and interoperate both in wired and wireless environment [40].

The agent platform can be distributed across multiple machines, each agent being able to send and receive remotely messages. They don’t need to share the same operating system, the control of the framework being assured via a remote GUI with visual overview over the agents that can migrate from one container to another when required.

Agent-based systems are intrinsically peer-to-peer systems. Each agent is considered a peer, which might need to initiate communication with another agent. Agents are loosely coupled entities, which can decide to accept or neglect an oncoming message. This method of communication allows the agent to decide which message it wants to serve first and which at a later time. It also allows the sender to control its thread of execution and not to remain blocked until the response is received and removes the temporal dependency between sender and receiver. The communication is considered just a type of action, and it carries with it semantic meaning. The agent will therefore know which type of message it has received, and will be able to properly understand the meaning of its content.

The agents’ paradigm projects several artificial intelligence concepts to the distributed technology. It describes the agent as an autonomous entity,who have control on their own threads and even take decisions under some circumstances. The agents are proactive: they are not only described as entities which react to external stimulus, usually having a goal-directed behavior. Agents are also social entities, which need to interact with other agents for accomplishing their goals [40].

Foundation for Intelligent Physical Agents (FIPA) is IEEE Computer Society standards organization for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems [5]. FIPA model fully embraces the agent paradigm, further detailing a model of an agent platform and a set off services that should be provided. The Agent Communication Language (ACL) is one of the most important parts of the FIPA standard, considering the fact that agents are social and autonomous entities. FIPA standardized a library of 22 acts of communication, like requesting, proposing, informing, querying, calling a proposal or refusing. The FIPA society also defined message structure that allow sender and receiver to fully understand the meaning of a message, and described a library of common interaction protocols like delegating an action or calling for a proposal. The greatest advantage of FIPA model is that of having the standard status, defined and accepted by the agent community.

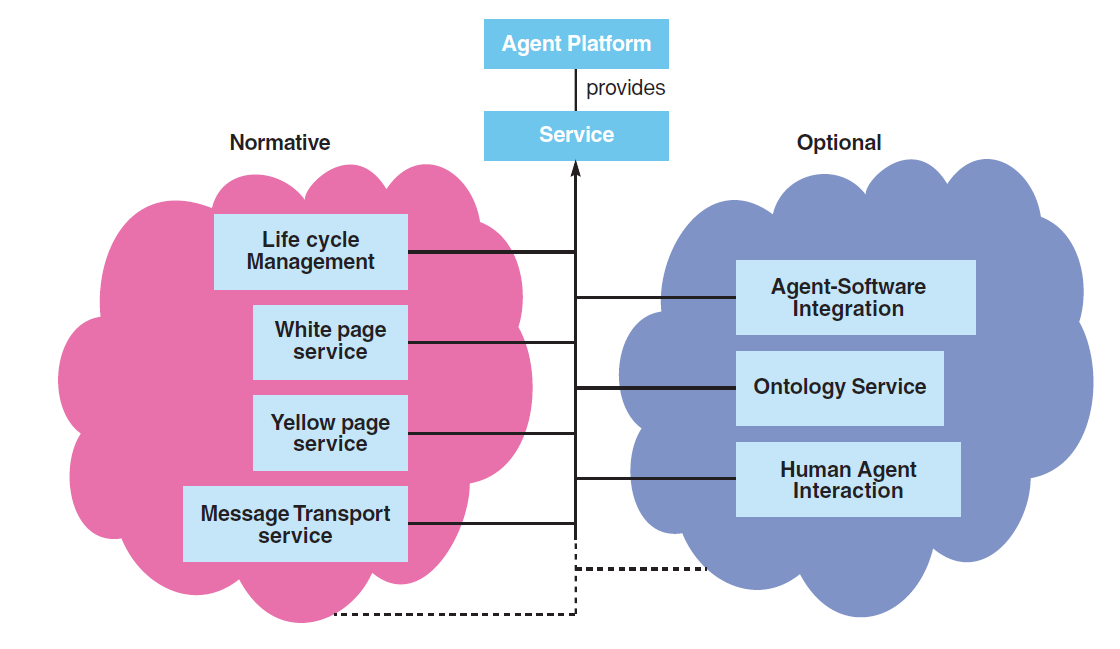


Figure 5.4: FIPA Standard: Services Provided by a Platform

JADE fully complies with the FIPA standards [40], therefore having the advantage of interoperability, agents being able to operate with other agents, provided that they comply with the same standard.

### Hyper-V WMI Provider

Microsoft Hyper-V, codenamed Viridian and formerly known as Windows Server Virtualization, is a hypervisor-based virtualization system for x86-64 systems [5]. Starting with the release of Windows Server 2008 R2, Live Migration is supported with the used of Cluster Shared Volumes (CSVs). This allows for failover of individual virtual machines, instead of entire host having to failover. That means that when a node fails, not all of the machines running on that server will fail. It also means that virtual machines can be moved from a node to other without needed to be turned of and stopped.

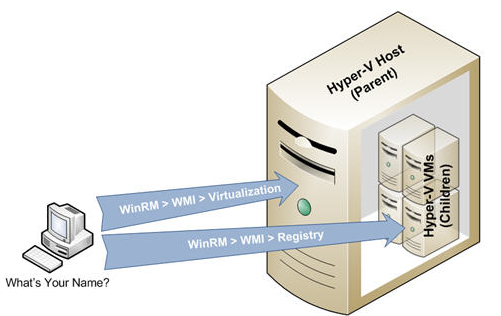


Figure 5.5 : Hyper-V Node

The API which enables the programmatic use of Hyper-V is the Hyper-V WMI Provider. The WMI provider for Hyper-V enables developers, and scripters, to quickly build custom tools, utilities, and enhancements for the virtualization platform. The WMI interfaces can manage all aspects of the Hyper-V services. It provides a virtual system management service, tools for network management and resource management.

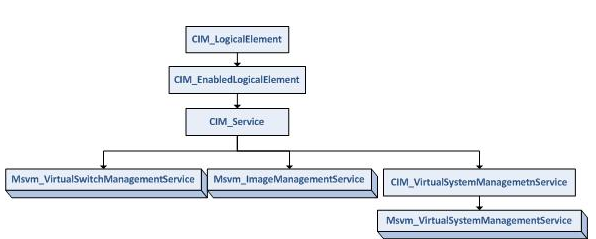


Figure 5.6: Hyper-V Services Structure

We can describe the WMI Classes by analogy with the SQL relational database. They are both defined with schema, but the WMI introduced the concept of inheritance in its class definition. Children class can inherit property and methods defined in their parents.

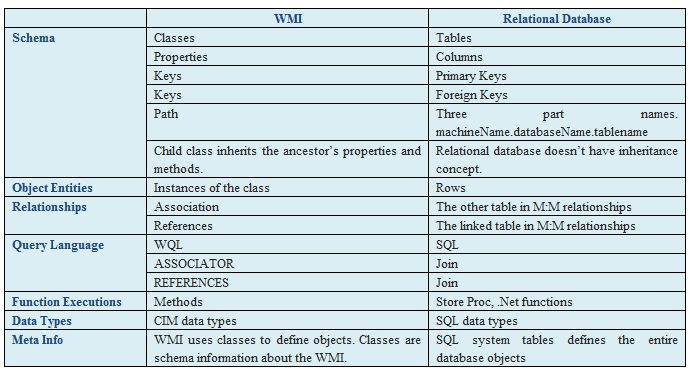


Figure 5.7: WMI and Relational Database Comparison

Properties of WMI objects describe characteristics of WMI objects. A property is a named value pair, and has a CIM type (one from CIM\_ILLEGAL, CIM\_EMPTY, CIM\_SINT8, CIM\_UINT8.. etc).

The methods existent in WMI are invoked through the InvokeMethod method called on the virtual system service, by giving as parameter the name of the method which needs to be called, and its parameters.

|  |
| --- |
| Listing 5.1: Method Call Example |
|  |

The inParams array of properties needs to be given the parameters with which the function is being called. The structure of the inParams object depends on the function which we want to call.

## Agents’ Implementation

The system is implemented through different JADE agents, each with its own purpose, which communicate with each other for realizing their goals. In the next sections, each agent is described, by giving its purpose and emphasizing implementation details.

### Context Model Administering Agent

The CMAA puts the basis of this system. First of all, it loads the ontologies needed for our context-aware system. The two ontologies describe the environment and respectively the datacenter model, being mapped on the <R, A, P> context-aware model.

#### Environment Ontologies

Figure 5.7 gives a perspective of the environment ontology. The ContextElement is the main ontology element. Actions, policies and resources inherit the ContextElement, they all being participants in the described environment.



Figure 5.8: Environment Ontology

The resource can be an actuator or a sensor. The sensor gives information on a property of the environment, while the actuator allows for actions to be exerted on the environment. Each actuator has some associated actions, and each action has associated a number of properties that it affects from the environment under the form of sensors. Therefore, by transitivity, the actuator will know which properties it affects from the environment, or in other words, which sensor will change its value due to an action exerted on the actuator.

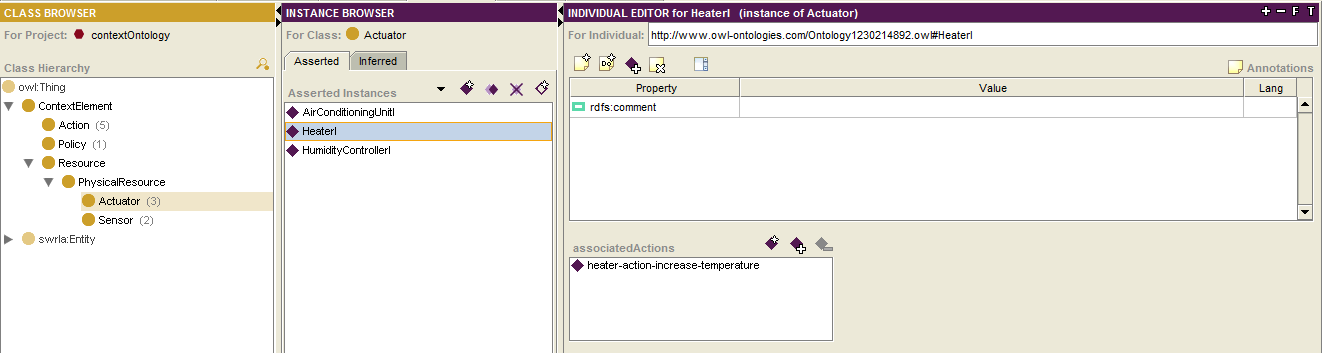


Figure 5.9: Actuator instance example: Heater

In the figure 5.8 we can see an example of an actuator instance, HeaterI, which has associated the action heater-action-increase-temperature. The instance has a specific URL, to enable an access point for instances of the ontology

Figure 5.9 gives a perspective of the environment ontology. The ContextElement is the main ontology element. Actors, policies and resources inherit the ContextElement, they all being participants in the described environment.

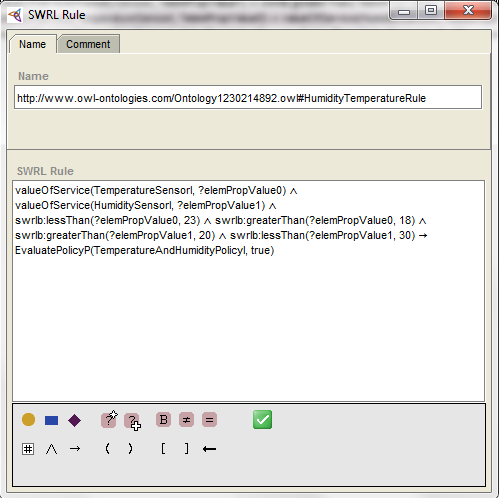


Figure 5.10: SWRL Rule

The antecedent of the SWRL rule describes the context in which the TemperatureAndHumidityPolicyI is true. It describes a situation in which the temperature is between the values 18 and 23 degrees, and the humidity between 20 and 30. When the antecedent is valid, the consequent should be valid too. The consequent consists of an assertion of the property respected of TemperatureAndHumidityPolicyI to true.

Due to the fact that we are trying to lower the energy consumption of a datacenter, we need to be careful with the efficiency of our algorithm. This is why we have chosen generating java code, with the help of Protégé ontology editor, having a considerable improvement in time efficiency instead of inferencing directly on SWRL rules.

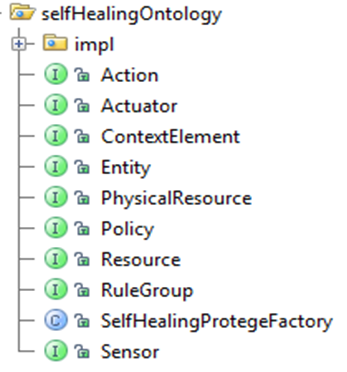


Figure 5.11: Java Implementation for Ontologies

In the figure 5.11 we have the two ontologies, one for environment and one for datacenter, which reside in the package selfHealingOntology and respectively greenContextOntology. In each of these packages, the SelfHealingProtegeFactory and respectively the ProtegeFactory classes provide methods for handling the ontologies.

|  |
| --- |
| Listing 5.2: Check if a Policy is respected |
|  |

On top of these, methods have been created for telling whether or not the QoS and Energy policies are respected (see listing 2). Instead of doing an inference on whether or not the policy is respected, with this behavior the system checks if the slot associated to respected property of the policy is true or false, and returns the value. This approach is much faster than the one using reasoners.

#### Creating the Agents

The listing 2 shows the instantiation of all the agents, achieved by the Context Model Administering Agent. First of all, the container on which the current agent resides is taken. On this container, the CIA, RLA, GUIA, X3DA and TMA are created.

|  |
| --- |
| Listing 5.3: CMAA creates the other agents |
|  |

Each of the agents receive the needed parameters. The Context Interpreting Agent receives as parameters the JenaOWLModels both for self-healing model and for self-adapting model. The agent which handles the graphical user interface for the datacenter receives only the owlModel for the datacenter. The reinforcement learning agent needs all owlModels and policy conversion models. On the other hand, the x3d agent and the task management agent don’t need any parameters.

### Reinforcement Learning Agent

The reinforcement learning agent has the following main behaviors: the self-healing behavior, the self-adapting behavior and the receive message behavior.

The ReceiveMessageRLBehavior class has a Cyclic Behavior. It enables the reinforcement learning agent to receive messages from different agents. It has an action method, which gets the received message and interprets it, taking measures depending on the content of the message.

The existence of the two behaviors is also a consequence that the reinforcement learning environment behavior has been defined for the CONSENS project [3], with the purpose of implementing a self-healing behavior, while the datacenter behavior is an improvement of the previous one.

I will detail the datacenter behavior of the reinforcement learning algorithm, while the environment behavior is detailed in [39].

Each time the agent is created, it loads the memory with what it has learned both for the datacenter and for the environment. In listing 4, both the memory for the self-healing algorithm and the memory for the self-adapting algorithm are restored in two variables and used for learning from them.

|  |
| --- |
| Listing 5.4: Reinforcement Learning Agent Memory Handling |
|  |

When the memory file isn’t found, there is instantiated a new variable for each of the behaviors.

#### Reinforcement Learning Datacenter Behavior

The reinforcement learning algorithm is a recursive algorithm. It keeps a priority queue with already visited contexts, ordered by the reward of being in that state.

Each time the algorithm starts, it polls a context from the queue. If it is empty, that means that all of the possible options have been exploited and have nothing else to try, and we should return the best context found so far. If this is not the case, the datacenter memory needs to be refreshed in order for it to know the protégé factory of the current time.

|  |
| --- |
| Listing 5.5: RLA learning algorithm |
|  |

If there exists a set of commands for the current context in the datacenter memory, the commands are added to the already found set of commands which brought us to this context, and the result is returned.

In listing 6 it is presented the code for finding the possible deploy actions in a datacenter. Each server is taken once at a time, with the condition that the task which has the broken policy fits on it, and the server is running. The server should also not contain once more the same task, and the task shouldn’t be running before deployment.

|  |
| --- |
| Listing 5.6: RLA learning algorithm |
|  |

A new SelfOptimizingCommand is instantiated, for the server at which we arrived and the task associated with the most important broken policy. If the context doesn’t contain exactly the same command once more in the commands queue, the system tries to simulate it. This solution is used because we don’t need cyclic behaviors. A new cotext snapshot is created, and the actions already found are executed on it. The entropy and reward function are set on that context and, after the action is rewinded the new context is added to the queue.

Considering the fact that the algorithm always considers the context with the highest rewards, it rapidly finds the actions. The reward is computed depending on the previous context’s entropy, and the current context’s entropy. It also depends on the command’s cost and the number of actions taken so far. Since we want to encourage the difference between the previous entropy and the entropy of the current context, the cost of the command and the number of actions to be as low as possible.

|  |
| --- |
| Listing 5.7: RLA reward function |
|  |

The entropy is defined by the sum of the products between the policy respectance degree and their priorities. The task respectance degree is the product between the weight of each resource and the difference between the requested value of the resource and the received value of the resource. The energy respectance degree is given by the sum of the products between the weights of the associated components, and the deviation from the optimal range for each of them. In fact, when the task respectance degree and energy respectance degree tend to zero, the policies are respected.

The listing 8 shows the code for simulating the effect of actions taken by the self-adapting algorithm on the context.

|  |
| --- |
| Listing 5.8: RLA self-adapting algorithm’s effect on the environment |
|  |

The environment is supposed to increase its temperature with a number of degrees equal to the number of actions found by the self-adapting algorithm.

### Task Management Agent

The task management agent handles adding and deleting a task, which can reside on a remote computer. At its creation, it instances the task management graphical user interface. For transmitting information from one agent to another, a data transfer object is being created, for describing the task with all of its properties.

The task data transfer object has attributes like the name of the task, the requested number of cores, the ranges of CPU, memory and storage, and the received number of cores, CPU, memory and storage.

|  |
| --- |
| Listing 5.9: Send Message to RLA |
|  |

The task management GUI will call the send message to RLA whenever an action occurs. The message can be of different types. If the message is of type INDIVIDUAL\_CREATED, the new message to be sent should contain as object content the message sent by the Task Management GUI. If the message type is INDIVIDUAL\_DELETED the content of the message will be as a String, and will contain the name of the task.

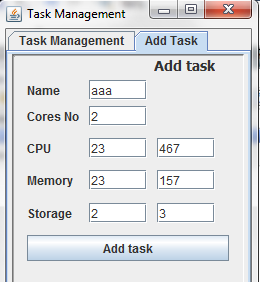


Figure 5.12: Add Task GUI

The Receive Message Behavior of the Task Management Agent receives information like all the tasks available in the datacenter, under the form of DTO Tasks, and sends it forward to the task management agent which will set the appropriate information in the GUI.

## Utility Components

The system has some components which are used for implementing some behaviors needed for the agents. Out of these, the negotiation and action enforcement components will be detailed in the followings.

### Nash Negotiation Component

The Negotiator class is a factory for two kinds of negotiation classes: the Nash negotiation and the Fuzzy logic negotiation. The Nash negotiation will be further detailed. The utility, or payoff function is a function which gives the degree of happiness for the first player having made the decision x under the assumption that the second player will make the decision y. For the Nash direct negotiation to be successful, we need to assume that each player knows the other’s player strategy.

For negotiating with the help of Nash theory, we have to consider our negotiation as a game. The two players are task and server, or QoS and Energy, playing a co-operative game.

In the listing 7 there is described an approach for building the payoff functions. The utilities1 function describes the payoff for task knowing that the server will choose y for that resource, while the utilities2 function describes the payoff for the server, knowing that the task chooses the x value for the resource under negotiation.

|  |
| --- |
| Listing 5.10: Nash Utilities |
|  |

Each function is composed of a reward function multiplied with a penalty function. The function for the task is an ascending function, which has the highest reward when it is at its highest value. The penalty for the payoff function of the task is multiplied by the penalty which is inversely proportional with the difference between the two values chosen by the players. The reward function for the server is a descending function, which has a lower value with the growth of the resource value. The penalty function for the server is the module of the difference between the value chosen by the server and the value chosen by the task.

There is an easy way to identify Nash equilibria on a payoff matrix: if the first payoff number, in the duplet of the cell, is the maximum of the column of the cell and if the second number is the maximum of the row of the cell - then the cell represents a Nash equilibrium [5].

The listing bellow emphasizes the way in which the Pareto optimal states are found, and out of them, finds the one which has the maximum values for the utilities functions.

|  |
| --- |
| Listing 5.11: Finding Nash Equilibrium |
|  |

The Pareto optimal states are the ones in which the utilities1 is maximum on the row, while utilities2 function is maximum on the column. The Nash equilibrium state is the one in which the average of utilities is maximum.

The negotiation process occurs for each resource and it is realized with a server chosen by the first method.

### Hyper-V Management WMI Endpoint

The servers have installed on them the Windows Server 2008 R2 operating system, with the Hyper-V role and the Failover Cluster feature installed. The Hyper-V is a hypervisor which can host or create virtual machines, and with whose help there can be created clusters with virtual machines that are able to live migrate from one node to another (See chapter ).

The Hyper-V Management endpoint component uses the WMI API for handling the virtual machines in the datacenter. The methods are written in C#, and used in web services which are published on a web site, and accessed from the global loop for enforcing the found actions.

Each of the commands is projected on a combination of web service method calls. The deploy command involves firstly a deploy, and then a start virtual machine. The move command is made in four main steps: firstly, the virtual machine is turned off; secondly, a snapshot of the virtual machine is taken, and the virtual machine is exported; the third step is to import the virtual machine; the fourth step is to start the imported virtual machine.

It is very important to pay attention to the computers on which the web services are called, and to careful send the paths from which and to which the virtual machines need to be exported or imported.

The VMHandling class has methods which are handling virtual machines, by using the Hyper-V WMI Provider. The WebService class has methods which use the methods of VMHandling class, residing in a web site. It will be accessed by the Global Loop Controller for reinforcing the actions that were found through the self-adapting algorithm.

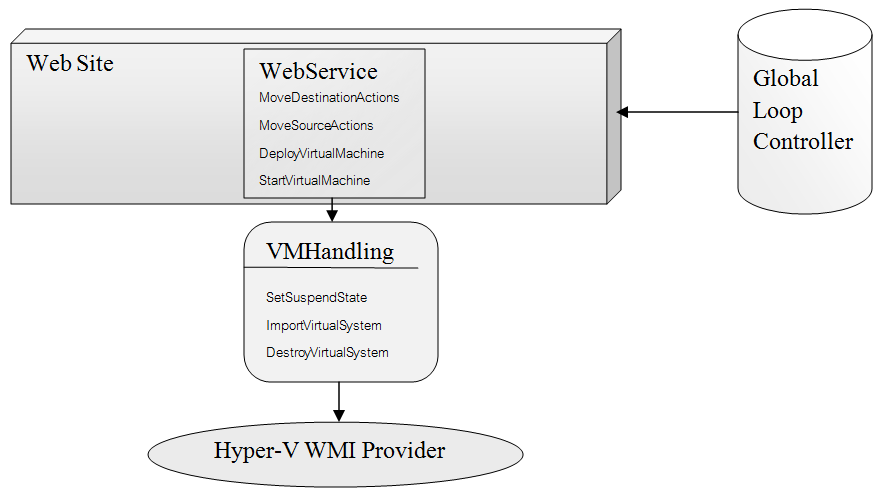


Figure 5.13: Virtual Machine Handling

For handling virtual machines, an object holding the Virtual System Management Service needs to be instantiated (see ). On this object we can call GetMethodParameters method, which gives us an array of parameters, indexed by their name. After calling the method for getting the parameters and filling the array with the proper parameters, we need to call the InvokeMethod method, having as parameters the name of the method to be called, the parameters with which the method needs to be called and an optional reference that is returned if the operation is executed asynchronously.

|  |
| --- |
| Listing 5.12: Import Virtual Machine |
|  |

The InvokeMethod method will return some output parameters, in which the return value can be started or completed. If it is started we need to wait for the operation to complete, and check from time to time to see its status. This is what the function Utility.JobCompleted (outParams, scope) does.

|  |
| --- |
| Listing 5.13: Deploy Virtual Machine |
|  |

The web method above uses the ImportVirtualSystem method from VMHandling class for importing the virtual machine. Firstly it copies the virtual machine together with its snapshots, configuration files and hard disks in a new folder. This is necessary because if we import a virtual machine from one folder, we can’t import it anymore somewhere else and we can’t delete it as long as it is not deleted from the hyper-v manager. Because of this tight connection, we created on the shared storage a folder for each server and at deployment or movement, the machine is copied into the folder of the destination server.

The interaction with virtual machines through the Hyper-V WMI follows the same pattern as for importing virtual machine. We need to get the scope, the virtual system management service and invoke the appropriate methods for our needs. After this, that function can be used to create a web service to be called by the global loop controller whenever an action needs to be enforced.

#### Modifying Virtual Machine’s Properties through WMI

In Listing 5.14 it is shown the way in which the setting data is taken for each of the needed resources, in our case memory, processor and storage.

|  |
| --- |
| Listing 5.14: Getting resources data from the virtual system |
|  |

Listing 5.15 shows the code through which values are given to the memory and processor properties. The named array gives ways to set limit, reservation and virtual quantity for the memory and the processor, therefore modifying the values for the current virtual machine.

|  |
| --- |
| Listing 5.15: Setting resources’ values |
|  |

### The two ManagementObjects above, namely memorySetting and processorSetting are stored in a resourceSettingData array and the method ModifyVirtualSystemResources is invoked with input parameters holding resourceSettingData array.

### Enforcement and Simulation Unit

This component contains all the commands, both for self-healing purposes and self-adapting purposes.

In the figure 5.14 the command package is visible, with its sub-packages: self-healing command and self-optimizing command.

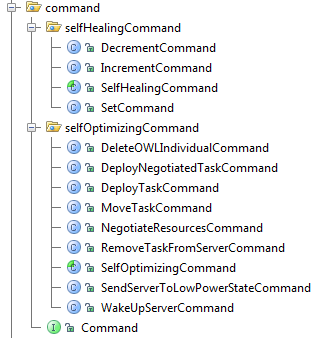


Figure 5.14: Command Package

The self-healing command package holds the abstract class SelfHealingCommand, which is extended by Decrement, Increment and Set Commands , which are basic commands to be enforced by the self-healing algorithm.

The self-optimizing command package holds the abstract class SelfOptimizingCommand, which is extended by Delete OWL Individual, Deploy Negotiated Task, Deploy Task, Move Task, Negotiate Resources, Remove Task From Server, Send Server To Low Power State and Wake Up Commands , which are basic commands to be enforced by the self-adapting algorithm.

Both the SelfOptimizingCommand and the SelfHealingCommand implement the interface Command, having the methods of execute and rewind on ontology, execute on web service and execute and rewind on X3D. The action can be therefore enforced at several levels, from ontology and X3D visualization to real world through web services. Each command has associated to it a cost, important for evaluating the entropy and for the learning process.

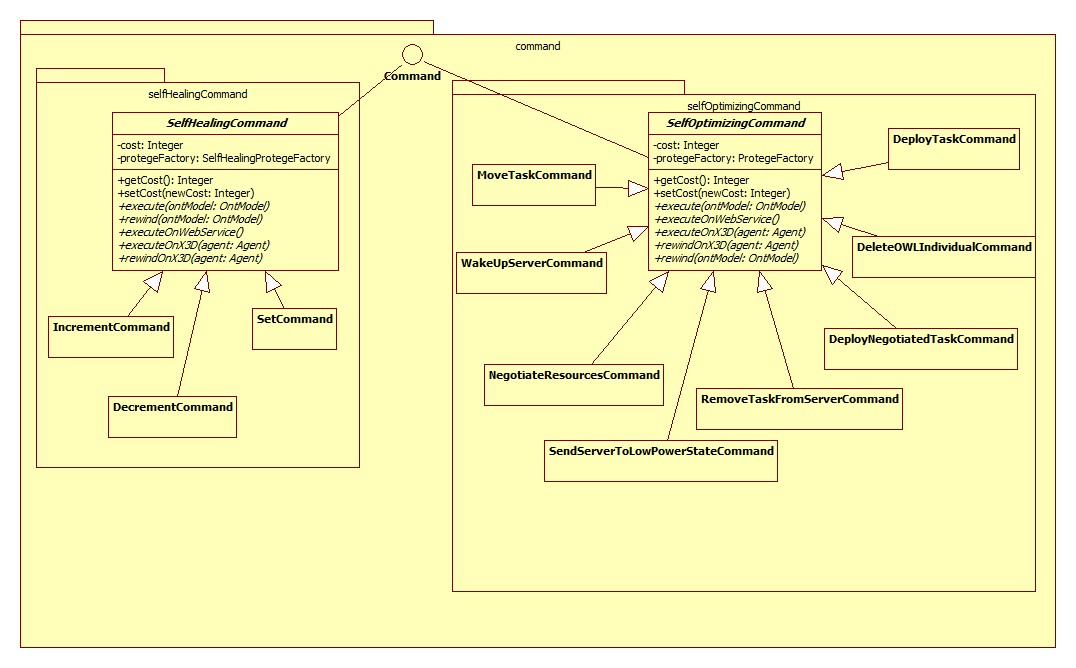


Figure 5.15: Command Package Class Diagram

The figure 5.15 shows the class diagram of the commands package, whose classes provide functionality both in terms of simulation and in terms of enforcement of commands. All the commands existent in the system implement the same interface, but have different protégé factory.

Chapter 6

# Evaluation

## Environment Self-Healing

For a thorough testing of the environment self-healing, in the next phase it is presented a more complex environment, with a larger number of resources. The environment is one of a smart laboratory, with a light, a temperature sensor, a computer and an alarm. It also has a sensor which tells whether or not the room is empty.



Figure 6.1: X3D Context Representation

For evaluating the efficiency in terms of time for the self-healing approach, there are provided two manipulation mechanisms.

The first one is through a Context Disturbing Behavior (CDB) attached to the RLA. It assigns values to sensors using a predefined pattern or random values in order to properly test the running time on different situations.

The second solution uses a direct manipulation method using X3D in which a click event performed on an X3D object executes a specific action on the object.

For this proof of concept implementation, the resources were considered as having equal weights.

### Tracing

#### Simple Situation

The validity of the returned actions needs to be tested, and this is why in the followings it is presented a tracing of the algorithm on a specific case, created using X3D direct manipulation.

The starting case is represented by the following states of the environment: the professor enters the room, the computer is off and the alarm is on. This situation breaks the Face Recognition Policy. For each resource which doesn’t have a value in the accepted range according to the current policy, the algorithm simulates executing each of the actions attached to the actuators associated to the current resource, generating each time a new context. For this context it goes recursively and has the same actions, for the policy which is the most broken for the current context.

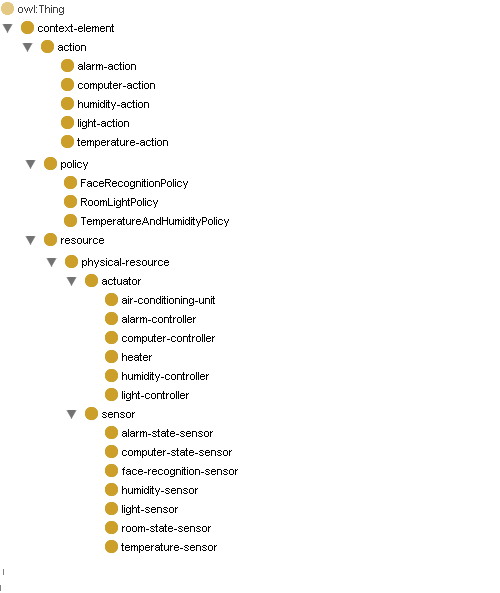


Figure 6.2: Ontology for Smart Laboratory

In this particular case, the first broken resource is the computer, on which each possible action is simulated, generating new contexts. In this case the only possible action is to “Set” the computer to ON, because setting it to OFF will not change its state. For each resulting state the simulation is continued with the next sensor that breaks the policy, namely the Alarm sensor. In a similar way, the Alarm sensor can only be set to OFF. Through these actions the Face Recognition Policy is fixed, as presented in figure 6.3.

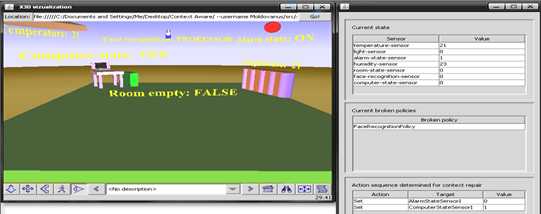


Figure 6.3: Face Recognition Policy broken

#### 6.1.1.2 Repeating pattern

In order to test the algorithm’s performance the first sensor manipulation mechanism was used, employing the Context Disturbing Behavior (CDB), for a repeating pattern of situations.

The pattern consists of four broken contexts:

* the professor is in the room while the Computer is OFF and Light is OFF
* the student is in the room while the alarm is ON
* the Temperature and Humidity are out of their admissible ranges
* an unknown person is in the room and the Alarm is OFF- it is supposed that at most one person is in the room

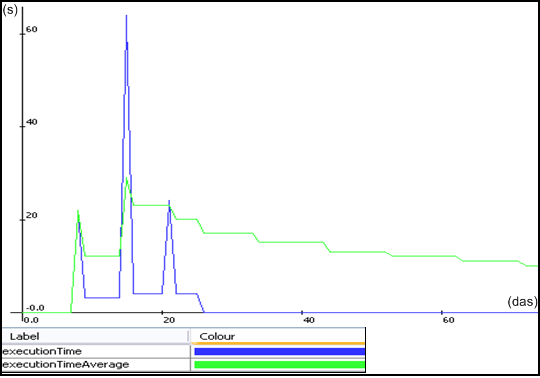


Figure 6.4: Context disturbed using a pattern

In the figure 6.4, the importance of the remembering process is visibly essential. First times that the algorithm runs for finding actions for the four situations, the running times are 20, 4, 61 and respectively 22 seconds. By repeating the patter, the execution time becomes zero, and the execution time average decreases considerably, going towards zero.

#### 6.4.1.3 Random Sensor Values

In order to test the algorithm’s performance the first sensor manipulation mechanism was used, employing the Context Disturbing Behavior (CDB), for random situations, with random sensors’ values situations.

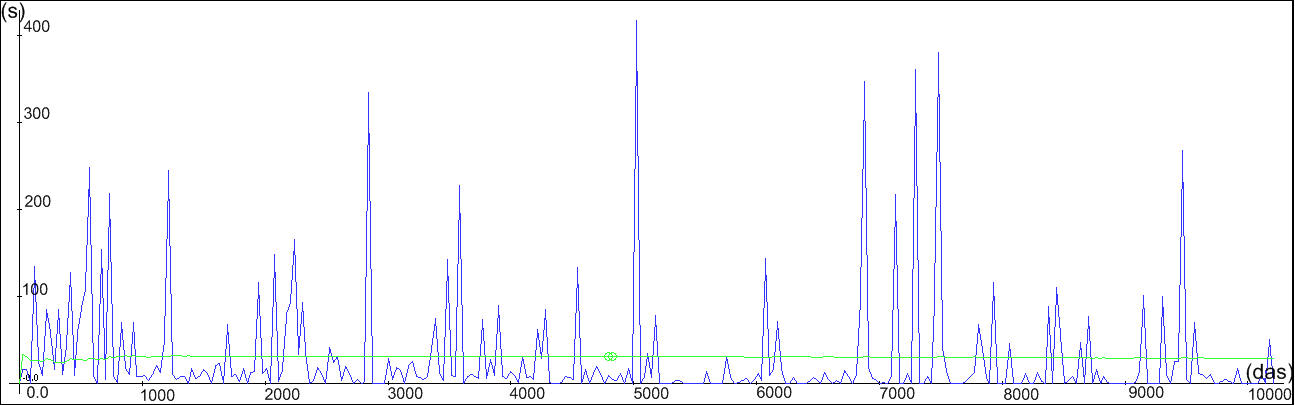


Figure 6.5: Context randomly disturbed

The figure 6.5 shows a 24 hours run for the random test. During this time, the CDB gave random values to all sensors from the following ranges:

* Temperature between 15 and 25
* Humidity between 15 and 35
* Light, Room State, Computer, Alarm with values 0 for ON and 1 for OFF
* Face Recognition Sensor with values 0 for Professor, 1 for Student and 2 for Unknown

Each of the spikes in the graph represents encountering a new situation, for which actions have not yet been stored to memory, and the self-healing algorithm was performed. In the first 10000 seconds, there are many spikes over 10 seconds. After that, with the increase in number of states stored in the memory, achieves the performance of having only three running times greater than 50 seconds in the time interval [50000, 70000]. Also an overall reduction in the number and height of the peaks is visible because at each step the algorithm checks if it doesn’t already know the best sequence of actions for the context that it arrived in. If this is not the case, results will be stored in association with the current context. Considering that the number of possible sensor combinations for the chosen ranges is 22.481.940, the self-healing mechanism behaves quite well in rapidly finding and taking the needed actions for fixing the broken context.

## Datacenter Self-Adaptation

The testing part has two main sides: firstly, the datacenter is simulated and X3D is used for creating navigable visualization environment and after that the algorithm is run on real servers.

### Simulating A Datacenter in X3D

In the figure 6.1 a datacenter is represented in X3D. The wireframe boxes represent play the role of servers, with grey or blue at the bottom of the box signaling that the server is alive or hibernating. Near each center we have a power meter, measuring the energy consumption of each server. In the center of the datacenter we have the server hosting the Global Loop Controller, which coordinates the entire activity of the datacenter. There are also three sensors above the datacenter, measuring the temperature and humidity in the room.

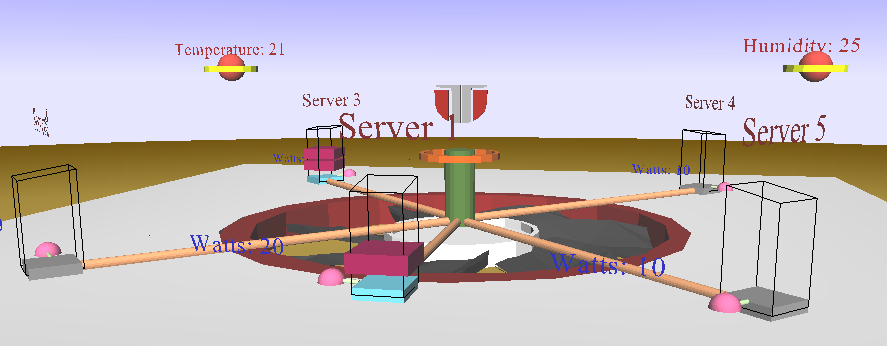


Figure 6.6: X3D View on the Datacenter Context

#### 6.2.1.1 Hardware infrastructure

The hardware infrastructure for the simulated datacenter is described in the table below. It contains five servers each with its own characteristics. This information is also recorded in the ontology which holds the model for the datacenter, with instances for each server.

Table 6.1: Datacenter servers’ characteristics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr | CPU | Memory | Storage | Power source |
| 1 | 1 x 3000 MHz | 2048 DDR2-1066 | 250 GB @ 7200 rpm | 300 W |
| 2 | 2 x 3000 MHz | 2048 DDR2-800 | 250 GB @ 7200 rpm | 365 W |
| 3 | 4 x 2000 MHz | 4096 DDR3-800 | 146 GB @ 10000 rpm | 480 W |
| 4 | 4 x 2260 MHz | 6144 DDR3-1333 | 146 GB @ 15000 rpm | 675 W |
| 5 | 8 x 2000 MHz | 8192 DDR3-1600 | 300 GB @ 15000 rpm | 2 x 1100 W |

To each server corresponds a policy which gives the optimal GPI (Green Performance Indicator). The policies are set to true through reasoning on SWRL rules. The antecedent of the rule consists of conjunctions of states in which each of the resources should be. When all of these conditions are true, the property respected of EnergyPolicy\_1 is asserted to true.

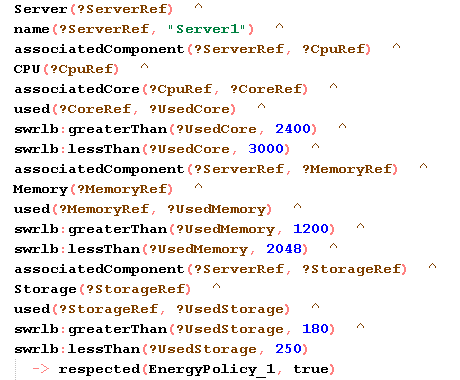


Figure 6.7: SWRL Rule for Server1 GPI Policy

Table 6.2 describes the poll of tasks which are waiting to be deployed. Each task has specific requirements regarding the CPU, Memory and Storage.

Table 6.2: Received tasks SLA requirements

|  |  |  |  |
| --- | --- | --- | --- |
| Task Nr | Task Requirements | | |
| CPU | Memory | Storage |
| 1 | 3000 MHz | 2048 MB | 200 MB |
| 2 | 2 x 1500 MHz | 512 MB | 400 MB |
| 3 | 2 x 2000 MHz | 1024 MB | 256 MB |
| 4 | 8 x 512 Mhz | 240 MB | 128 MB |
| 5 | 3 x 2000 MHz | 4096 | 300 GB |

As for servers, each task has associated a policy specifying KPI (Key Performance Indicators). Each of the resources has to be in a certain range for the customer who sent the task to be satisfied. This is exactly what is specified in the antecedent of the SWRL rule from figure 6.3. When all of the conditions of the antecedent are satisfied, the consequent conditions are asserted.

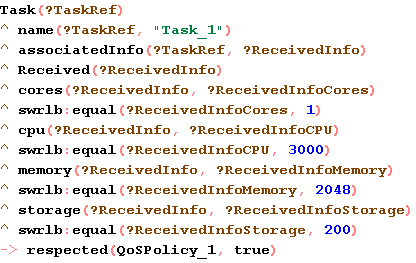


Figure 6.8: SWRL Rule for Task 1 KPI Policy

As initial context the Nr 1 server is active and the other servers are in a low power state. The basic flow of this example scenario is the following: (i) task 1 is received and is deployed on server 1 which reaches its load threshold and a wake up search which concludes by activating server 2; (ii) task 2 is received and deployed on server 2; (ii) task 3 is received; there is no room to deploy so server 3 is activated; all the task rearrangement and deployment options are analyzed and the solution is to move task 2 on server 3, deploy task 3 on the same server and send server 2 to low power state.

#### 6.2.1.2 Tracing

Tasks are received from the client together with an XML file containing pairs of KPI requirements and their corresponding priority. For each task, a policy is generated containing the specified values for QoS attributes. After negotiation between the task and the server, new relaxed QoS and Energy policies are generated as result. Policies are composed of weighted conditions for resources state.

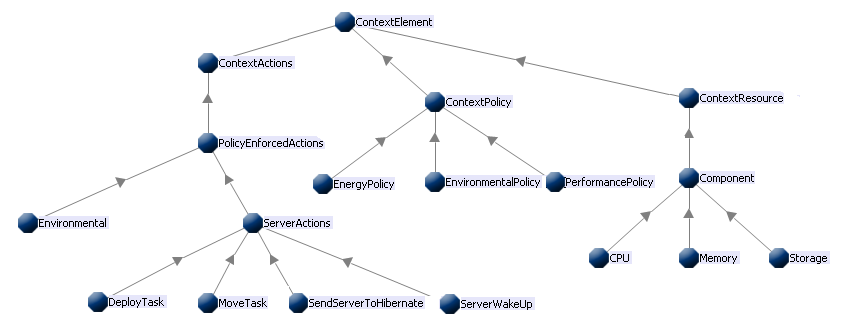


Figure 6.9 : Datacenter Context Ontology

For simplicity and for a better illustration of the algorithm all tasks are considered to have the same priority and the memory is assumed to be empty.For the real case, if the current context has a sequence of actions associated to it in the memory, we take that sequence of actions or we add it to the existent path of actions that we have reached so far.

At context evaluation time, the self-adapting algorithm notices that the entropy is larger than zero therefore we have one or more broken policies. The reinforcement learning algorithm will evaluate all possible paths and simulate taking the one with the largest reward. It will reach to a result composed of four actions: deploy task 1 on server 1, wake up server 3, deploy task 2 on server 3 and deploy task 3 on server 3. After this sequence of actions is returned by the reinforcement learning algorithm, the actions are taken in this exact order, bringing the system to an optimum context.

Considering that the Server 1 is turned on, the deploy action is possible and it is the best action to be taken since we need the tasks to wait as little time as possible. The Figure 6.7 shows the deploy action, the pink shape inside it being the task 1.



Figure 6.10: Deploy Task 1 on Server 1 Action

After the task 1 has been deployed, the next task having its KPI policy broken is the Task 3, but it doesn’t fit any server. The algorithm has found that the next decision is to wake up Server 3.



Figure 6.11: Wake up Server 3 Action

After the Server 3 was woken up, the next broken policy is taken. The KPI policy is associated to Task 2, therefore the self-adapting algorithm first tries to deploy Task 2 anywhere is possible, before waking up other servers or trying move actions. The Task 2 is deployed on Server 3.



Figure 6.12: Deploy Task 2 on Server 3 Action

After Task 2 is deployed on Server 3, the only remaining task which has an unsatisfied KPI policy is Task 3. Since it fits on Server 2, it is deployed there.



Figure 6.13: Deploy Task 3 on Server 3 Action

After finding a new action plan for a context, the pair (context, action plan) is stored for further reference. The reinforcement learning algorithm will always check first if this situation hasn’t been encountered before and if so, it will take the learned actions.

### Real-World Testing

For real-world testing, Windows Server 2008 R2 was installed on two computers, each of them having 2 GB of memory and a processor Intel Core 2 Duo at 3 GHz. Windows Storage Server was installed on a laptop having 1GB memory, and a processor Intel Dual Core of 1.6 GHz. This laptop is used for as a common storage, having a folder in which reside tasks as virtual machines, waiting to be deployed. This folder has sub-folders, one for each server, each of which with the server’s name, in which reside tasks which are running on the respective server.

The global loop controller which consists of self-healing capabilities for the environment and self-adapting capabilities for the datacenter resides on a different computer.

All the four computers are in the same domain. One of the two plays the role of DNS Server while the Storage Server has a File Server role. The two servers have Hyper-V hypervisor installed, being able to run up to four virtual machines simultaneously, since it is an enterprise version of Windows Server 2008 R2.The actions are enforced through the Action Enforcement Unit. Instead of having a server for each task, we will have the minimum number of servers awake at each moment of time.

#### Tracing

The following context is being traced: there are 3 tasks on the pool, and the above two servers in the cluster. The tasks have their associated KPI policies broken, since they are not deployed and therefore they haven’t received any part of resources.

The figure 6.14 shows the Hyper-V Manager on the WIN-RUQHITH265R, with no virtual machines running on it.

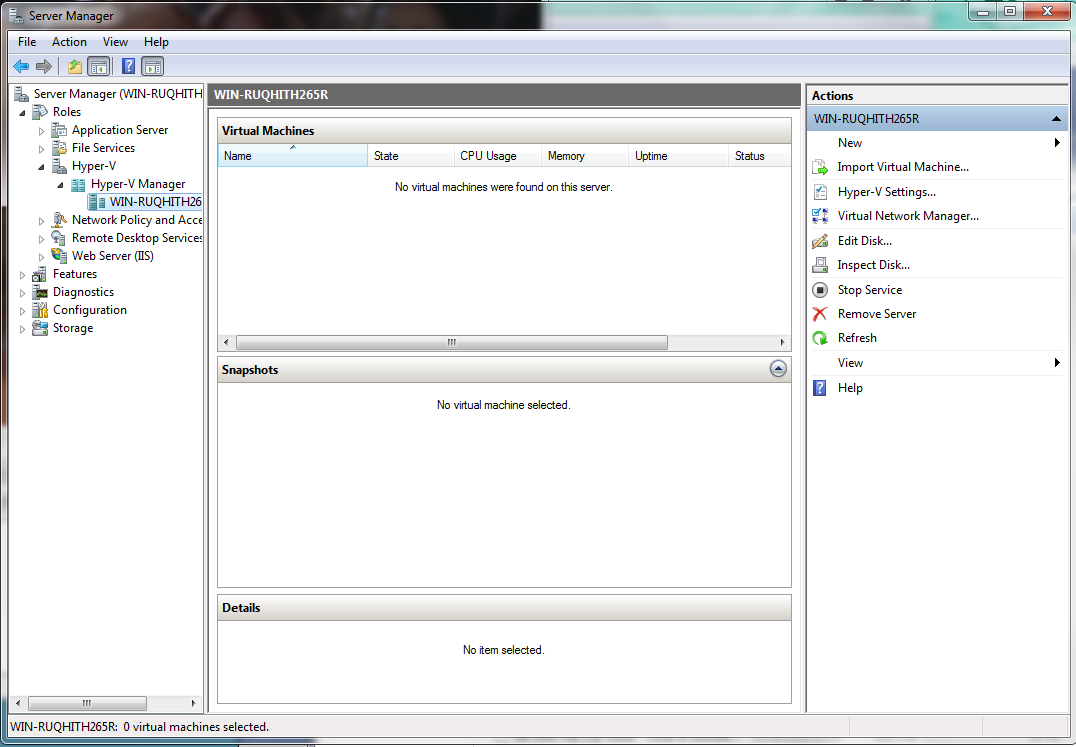


Figure 6.14: Before the Self-Adapting Algorithm

The figure 6.15 describes the resources associated to the two servers, with their associated resources usage, with two different representations.

The first window shows the running tasks, which at this time don’t exist. The second window shows the resources usage for the first server while the third window shows the resources usage for the second server. The second window is in task manager style, while the third window shows pies of free resources and resources used by the computer. The view for task usage exists in both representations for both of the servers.

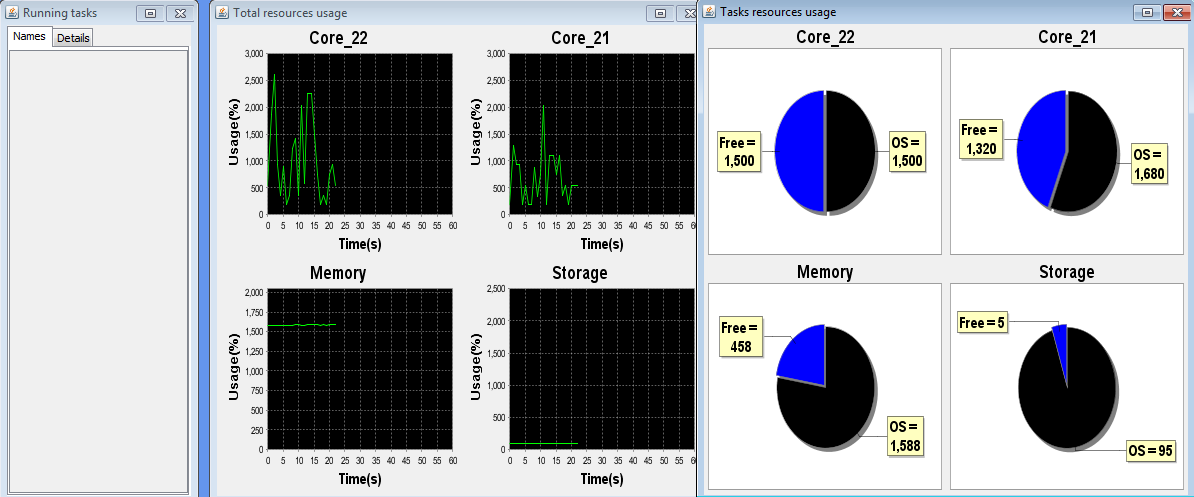


Figure 6.15: Resources Usage Before

All the three tasks are deployed on one single server, which is woken up first and then all the tasks fit on it therefore the optimal solution is found. The tasks are deployed all on the same computer, and started. The choice of the computer on which the tasks are deployed isn’t important since we have two similar servers in terms of CPU, memory and storage.

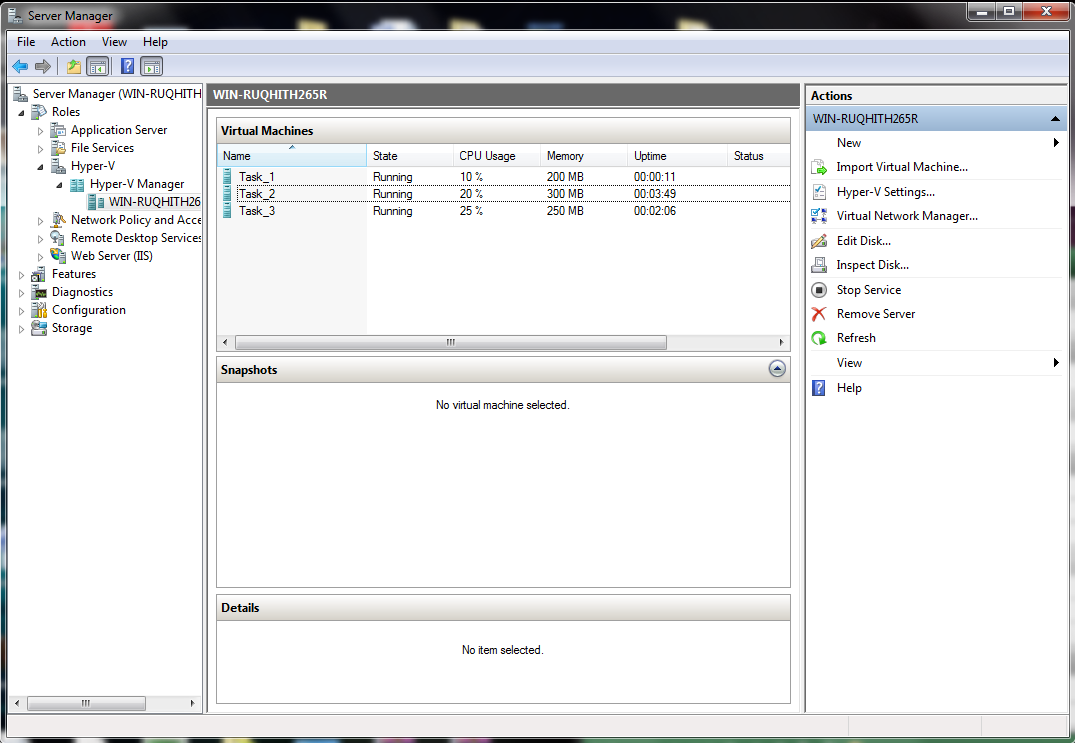


Figure 6.16: All the tasks are deployed

Figure 6.17 shows a connection to a virtual machine, namely to Task 2. It has an XP operating system running on it, which has a jar holding a simple program for the virtual machine to use the CPU.

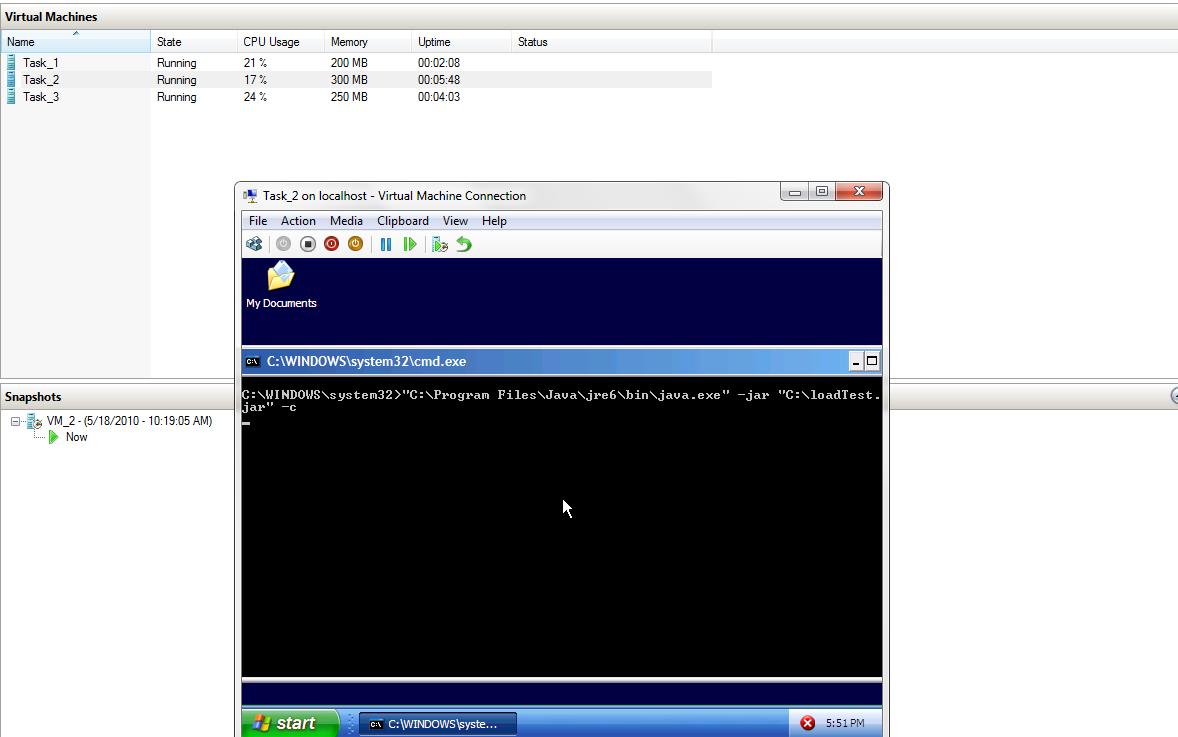


Figure 6.17: Task 2 runs a small XP

The following figure shows the state of the virtual machine and processor 1 with both of the resource usage views (task manager style and pie chart). Task 1 received 500 MHz on one core of the server, 300 MB of memory and 1 GB of storage. Task 2 received 500 MHz on both cores of the server, 300 MB if memory and 1GB of storage and Task 3 received 200 MHz on both cores of the server, 400 MB if memory and 1GB of storage.

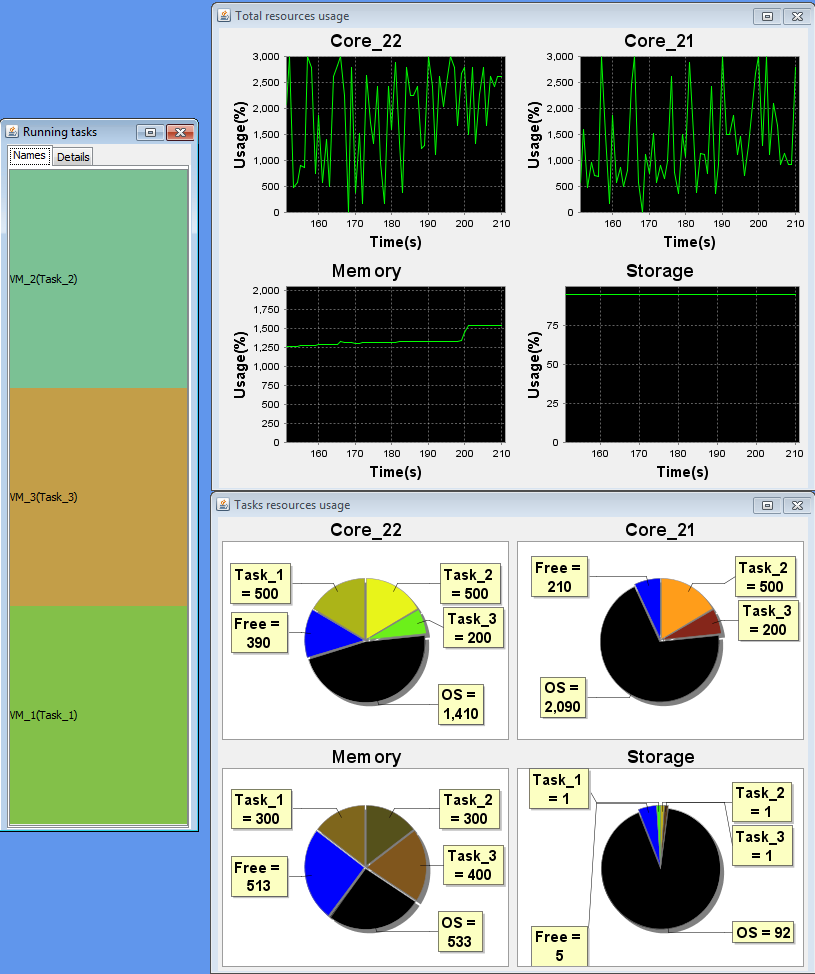


Figure 6.18: Resources Usage After

Chapter 7

# Conclusions

This chapter presents the conclusion drawn after analyzing, designing, implementing and testing the Global Loop Controller of the GAMES Datacenter. It underlines the advantages and disadvantages of the chosen solution, ending with some further development suggestions.

## Conclusions

The purpose of this paper was presenting a controller for decreasing energy consumption of datacenters, while keeping the environment situation under control. It has three main parts:

* Providing self-healing capabilities for keeping the room in which the datacenter resides in a reasonable context
* Providing self-adapting capabilities to the datacenter to be able to adapt to new situation with a minimum energy effort
* Negotiating the requirements that the clients ask for a certain task

The first two capabilities of the datacenter and the room it resides in are possible only if the context is aware of itself, or, in other words, the system is context-aware. The presented system is aware of itself by representing the context in which it resides as a (Resources, Actors, Policies) model. Each of the contexts are mapped on the <R, A, P> model, assigning roles to each existent object, and defining policies for the respective environment. In addition to mapping the context on the model defined in [2], the entropy is defined depending on the current context as the degree of disturbance in the context.

The self-healing capabilities were implemented through a reinforcement learning approach, in which the system has as reward the inverse of the entropy. The context describes the environment in terms of temperature and humidity since we need to continuously adjust these properties when they are too high or two low, considering that servers may fail or higher their energy consumption when put in unreasonable situations. For describing this self-healing approach some concepts were introduced like *recheck time* for needed time in order for the effect to be visible, *rollback* for rewinding actions which have been taken but don’t have the desired effect and *Inter-Independent Resources Group* for a group of resources which don’t have actions which affect other resources. The self-healing algorithm finds a sequence of actions which are being taken in exactly the same order in order to bring the context to an acceptable state.

The self-adapting capabilities were implemented by improving the reinforcement learning approach employed for enforcing the self-healing capabilities. The context consists of several servers which receive tasks coming from users and run them, without breaking the service level agreement. The servers play the role of resources, the actions are the actors which produce changes in resources and policies are defined for the current context. Therefore, the context is mapped on the <R, A, P> model. The reinforcement learning algorithm is improved by not expanding each of the possible future contexts, and by adding a complex reward which improves the running time of the algorithm. This solution is also tested on real servers, running Windows Server 2008 R2 operating system.

Since the requirements for each task are chosen by clients, the natural behavior is that the client chooses the maximum values for resources for the money that he pays, constructing the service level agreement for this transaction. This is why we need a negotiation process for the situation in which the task to be deployed doesn’t fit on any server, no matter how many move actions we try, or how many servers we try to start. The negotiation strategy proposed uses a cooperation game for finding the Nash equilibrium point. The Nash equilibrium point is the best solution for the two players involved in the game, namely the Energy and SLA. For this negotiation game, payoff functions were defined for finding the equilibrium point. The negotiation is done for each of the resources, and the result is constructed from the resulted negotiated values.

## Encountered Impediments

A huge impediment which seems to be a simple thing at the first sight is that the hibernate action is disabled on Windows Server 2008 R2 version. The hibernate action is needed for the command of putting to sleep the server, which is in this operating system impossible. This problem was overcome by not starting the hypervisor at computer start, through modifying a register. For each turn off server action, we need a restart and a hibernate command, which means a huge overhead.

The Hyper-V role can be installed only if the computer has a processor which includes virtualization option. If the BIOS version is too old, the virtualization option might not be included, therefore there is needed a BIOS update.

The Live Migration exposed as a further development point is possible in Windows Server 2008 only when the Failover Clustering feature is installed, and a highly available cluster is created. A step of creating a highly available cluster is creating a common storage, which has many different requirements out of which the most important one is to be an ISCSI target, in particular an ISCSI3 target needed for Windows Server 2008, supporting persistent reservation. This is possible only if the storage server has a Windows Storage Server 2008 operating system installed, therefore the solution used for this approach was using the move action composed of four moves.

## Further Development

For negotiation, the straightforward Nash negotiation method was used, considering that negotiation is a game. The existent approaches using Nash theorem are focused on the direction of bargaining Nash algorithm, which seems to be a more efficient approach from the time point of view. This is why one point of further development is trying to use the Nash Bargaining approach, and compare it with the straightforward Nash approach for negotiation. Also, still for the negotiation point, an improvement would be to try to use a package deal procedure or a sequential agreement approach.

Another improvement point from implementation point of view would be to create a cluster and implement move actions through Live Migration. This would improve very much the time wasted on moves which on the current implementation consist of turning off the virtual machine, exporting the virtual machine, importing it on the new server and starting it. Another great advantage of Live Migration is the fact that the virtual machines don’t need to be turned off in order for them to be moved. This is important both for the time and from the point of view of the task’s clients since the movement of virtual machines isn’t sensed externally.

# References

[1] **U.S. Environmental Protection Agency, ENERGY STAR Program**, “Report to Congress on Server and Data Center Energy Efficiency”, Public Law 109-431, 2 august 2007

[2] **Tudor Cioară, Ionuţ Anghel, Ioan Salomie,** “A Generic Context Model Enhanced with Self-configuring Features”, Journal of Digital Information Management (JDIM), Volume 7/ 3, pp.159-165, ISSN 0972-7272, 2009.

[3] **Ionuţ Anghel, Tudor Cioară, Ioan Salomie, Mihaela Dînşoreanu, *Georgiana Copil*, Daniel Moldovan**, “A Reinforcement Learning based Self-healing Algorithm for Managing Context Adaptation”, 8th Annual IEEE Int. Conf. on Perv. Comp. and Communications, 2010.

[4] **Tudor Cioară, Ionuţ Anghel, Ioan Salomie, Mihaela Dînşoreanu, *Georgiana Copil*, Daniel Moldovan**, “A Self-Adapting Algorithm for Context Adaptation”, !!!! roedunet?

[5]**Wikipedia**, Web Site: [www.en.wikipedia.org](http://www.en.wikipedia.org)

[6] **James Carter**, “**A look inside IBM's green data center research**”, Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design.

[7] **VMWare Team**, “A performance Comparison of Hypervisors”, Jan 31, 2007

[8] **Hyper-V WMI Provider**, Web Site: <http://msdn.microsoft.com/en-us/library/cc136992%28v=VS.85%29.aspx>

[9] **IBM manifesto,** Web Site:<http://www.research.ibm.com/autonomic/manifesto/>

[10] **Jeffery Kephart, David Chess**,”The vision of autonomic computing”,Computer Magazine, IEEE, 2003

[11] **Thomas Strang, Claudia Linnhoff-Popien**, “A context modeling survey”, Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England

[12] **M. Weiser**, “The Computer for the 21st Century,” in Scientific American, 1991.

[13] **MIT's Intelligent Room**, Web Site: [www.ai.mit.edu/projects/iroom](http://www.ai.mit.edu/projects/iroom)

[14] **Stanford’s iRoom project**, Web Site: <http://graphics.stanford.edu/projects/iwork>

[15] **Jason I. Hong, James A. Landay**, “An Architecture for Privacy-Sensitivity Ubiquitous Computing”, International Conference On Mobile Systems, Applications And Services,  
Proceedings of the 2nd international conference on Mobile systems, applications, and services

[16] **Paul Prekop, Mark Burnett,** “Activities, Context and Ubiquitous Computing”, Computer Communications, Volume 26, Number 11, 1 July 2003 , pp. 1168-1176(9)

[17] **Harry Chen, Tim Finin, Anupam Joshi,** “An Ontology for Context-Aware Pervasive Computing Environments”, The Knowledge Engineering Review, Volume 18 ,  Issue 3  (September 2003)

[18] **Lalana Kagal, Tim Finin, Anupam Joshi,** “A Policy Language for a Pervasive Computing Environment”, Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks

[19] **Damien Borgetto, Patricia Stolf,** “Energy Aware Autonomic Manager”

[20] **Josep Lluís Berral, Íñigo Goiri,Ramon Nou,Ferran Julià, Jordi Guitart, RicardGavaldà, Jordi Torres, “Towards energy-aware scheduling in data center using machine learning“,** 1st Int’l Conference on Energy-Efficient Computing and Networking (e-Energy 2010).  University of Passau, Germany. April 13-15, 2010.

[21] **Chandrakant Patel, Ratnesh Sharma, Cullen Bash, Sven Graupner, “**Energy Aware Grid: Global Workload Placement based on Energy Efficiency”, HP Technical Report HPL-2002-329

[22] **T.Srinivasan, J.B.Siddharth Jonathan, Jayesh Seshadri and Arvind Chandrasekhar**, “A Comprehensive Architectural Framework for Task Management in Scalable Computational Grids”- Annual National level technical symposium conducted by the Computer Science and Engineering Association of the Department of Computer Science and Engineering (DCSE), College of Engineering, Anna University, March 14-15,2005

[23] **Alexandre Mello Ferreira, Kyriakos Kritikos, Barbara Pernici**, “Energy-Aware Design of Service-Based Applications Source”, Proceedings of the 7th International Joint Conference on Service-Oriented Computing, 2009

[24] **Markus Huebscher, Julie McCann,** “A survey of Autonomic Computing – Degree, Models and Applications”, ACM Computing Survey, 40, 3, Article 7, August 2008

[25] **Antonia Carzaniaga, Alessandra Gorla, Mauro Pezzè,** “Self-Healing by Means of Automatic Workarounds”, Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems

[26] **Rajarshi Das, Jeffery Kephart, Charles Lefurgy,** “Autonomic Multi-Agent Management of Power and Performance in Data Centers”, Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track, 2008

[27] **IBM ETTK**, Web Site: <http://www.ibm.com/developerworks/webservices/library/ws-heal/>

[28] **Mazeiar Salehie, Ladan Tahvildari,** “Self-Adaptive Software: Landscape and Research Challenges”, ACM Transactions on Autonomous and Adaptive Systems (TAAS), May 2009

[29] **Alexandre Mello Ferreira, Kyriakos Kritikos, Barbara Pernici,**“Energy-aware Design of Service Based Applications”, Proceedings of the 7th International Joint Conference on Service-Oriented Computing, 2009

[30] **Katsuhide Fujita, Takayuki Ito, Mark Klein,** “Finding Nash Bargaining Solutions for Multi-issue Negotiations: A Preliminary Result”, Proceedings of the 1st International Working Conference on Human Factors and Computational Models in Negotiation, 2008

[31] **Jen-Hsiang Chen, Kuo-Ming Chao, Nick Godwin, Colin Reeves, Peter Smith,** “An Automated Negotiation Mechanism Based on Co-Evolution and Game Theory”, Proceedings of the 2002 ACM symposium on Applied computing, 2002

[32] **Ivona Brandic, Dejan Music, Schahram Dustdar**, “Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self Adaptable Grid and Cloud Services”, Proceedings of the 6th international conference industry session on Grids meets autonomic computing, 2009

[33] **Yan He, Miaoliang Zhu,** “A Complete and Efficient Strategy Based on Petri Net in Automated Trust Negotiation”, Proceedings of the 2nd international conference on Scalable information systems, 2007

[34] **Guoming Lai, Katia Sycara, Cuihong Li,** “A Decentralized Model for Multi-attribute Negotiations”, Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet,2006

[35] **Shaheen Fatima, Michael Woolbridge, Nicholas R. Jennings,** “An Analysis of Feasible Solutions for Multi-issue Negotiation involving Nonlinear Utility Functions”, Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, 2009

[36] **Shaheen Fatima, Michael Woolbridge, Nicholas R. Jennings,** “Multi-issue Negotiation with Deadlines”, Journal of Artificial Intelligence Research,  
Volume 27 ,  Issue 1  (September 2006)

[37] **Ville Kaila, Arto Annila** “Natural selection for least action.” Proceedings of The Royal Society A. 2008 PhysOrg.com.

[38]**George Santayana,** “The life of reason-The phases of human progress”, 1905-1906

[39] **Daniel Moldovan**, “Green Active Management of Energy in IT Service centres- Global Loop”

[40] **JADE whitepaper**, Web Site: <http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf>

# Appendix A: Java Source Code

|  |
| --- |
| NashNegotiator.java |

1 **package** negotiator.impl;

2

3 **import** greenContextOntology.Core;

4 **import** greenContextOntology.RequestedTaskInfo;

5 **import** greenContextOntology.Server;

6 **import** greenContextOntology.Task;

7 **import** negotiator.Negotiator;

8

9 **import** java.util.Collection;

10 **import** java.util.HashMap;

11 **import** java.util.Map;

12

13 **import** negotiator.Negotiator;

14

15 */\*\**

16  *\* Created by IntelliJ IDEA.*

17  *\* User: Administrator*

18  *\* Date: Apr 19, 2010*

19  *\* Time: 6:36:04 PM*

20  *\* To change this template use File | Settings | File Templates.*

21  *\*/*

22 **public class** NashNegotiator **implements** Negotiator {

23

24 **private int** quality = 10;

25

26 **public double** negotiateGivenRanges(**int** minValue1, **int** maxValue1, **int** minValue2, **int** maxValue2, **int** optimalValue1, **int** optimalValue2) {

27 **int** increment1 = (maxValue1 - minValue1) / quality; *//task*

28 **int** increment2 = (maxValue2 - minValue2) / quality; *//server*

29

30 **if** (minValue1 < minValue2) minValue1 = maxValue2;

31 **else** minValue2 = minValue1;

32

33 **boolean**[][] maxUtilities1 = **new boolean**[quality][quality];

34 **boolean**[][] maxUtilities2 = **new boolean**[quality][quality];

35 **double** utilities1[][] = **new double**[quality][quality];

36 **double** utilities2[][] = **new double**[quality][quality];

37 **for** (**int** i = 0; i < quality; i++) {

38 **int** currentValue1 = i \* increment1 + minValue1;

39 **double** max = 0;

40 **int** pos = 0;

41 **for** (**int** j = 0; j < quality; j++) {

42 **int** currentValue2 = j \* increment2 + minValue2;

43

44 **double** penalty1 = (currentValue2 - currentValue1 + 0.00001);

45 **double** penalty2 = (currentValue2 - currentValue1);

46 **if** (penalty1 < 0) {

47 penalty1 \*= (-1);

48 }

49 **if** (penalty2 < 0) {

50 penalty2 \*= (-1);

51 }

52 **double** reward1;

53 **if** (currentValue2 - currentValue1==0 )

54 reward1 = (currentValue1 - minValue1)/(maxValue1 - minValue1 + 1.00001);

55 **else**

56 reward1 = (currentValue1 - minValue1)/(maxValue1 - minValue1 + 1.00001);

57 **double** reward2 = (currentValue2 - maxValue2) / (maxValue2 - minValue2 + 1.00001);

58 utilities1[i][j] = reward1 - penalty2;

59 utilities2[i][j] = reward2 - penalty2;

60 */\**

61  *double penalty1 = 1/(currentValue2 - currentValue1+0.00001);*

62  *double penalty2 = (currentValue2 - currentValue1) ;*

63  *if (penalty1 < 0) {*

64  *penalty1 \*= (-1);*

65  *}*

66  *if (penalty2 < 0) {*

67  *penalty2 \*= (-1);*

68  *}*

69  *utilities1[i][j] = Math.exp(currentValue1/increment1) \*penalty1;*

70  *utilities2[i][j] = -Math.log(currentValue2/increment2) \*penalty2;*

71  *\*/*

72

73 System.out.print(utilities1[i][j] + **" "**);

74 **if** (j == 0) {

75 max = utilities1[i][j];

76 }

77 **if** (max < utilities1[i][j]) {

78 max = utilities1[i][j];

79 pos = j;

80 }

81 }

82 System.out.println(**"\n"**);

83 maxUtilities1[i][pos] = **true**;

84 }

85 System.out.println(**"\n"**);

86 **for** (**int** j = 0; j < quality; j++) {

87 **double** max = 0;

88 **int** pos = 0;

89 **for** (**int** i = 0; i < quality; i++) {

90 System.out.print(utilities2[i][j] + **" "**);

91 **if** (i == 0) {

92 max = utilities2[i][j];

93 }

94 **if** (max < utilities2[i][j]) {

95 max = utilities2[i][j];

96 pos = i;

97 }

98 }

99 System.out.println(**"\n"**);

100 maxUtilities2[pos][j] = **true**;

101

102 }

103 **double** x = 0;

104 **double** maxutility = -50000000;

105 **int** p = 0;

106 **int** q = 0;

107 **for** (**int** i = 0; i < quality; i++) {

108 **for** (**int** j = 0; j < quality; j++) {

109 **if** (maxUtilities1[i][j] && maxUtilities2[i][j]) {

110 **int** currentValue1 = i \* increment1 + minValue1;

111 **int** currentValue2 = j \* increment2 + minValue2;

112 **if** (((utilities1[i][j] + utilities2[i][j]) / 2.0) > maxutility) {

113 p = i;

114 q = j;

115 x = (**double**) (currentValue1 + currentValue2) / 2;

116 maxutility = (utilities1[i][j] + utilities2[i][j]) / 2.0;

117 }

118 *//TODO : if not equal, optimize function :D*

119 System.out.println(**"Equilibrium Found!!!"** + **"("** + i + **","** + j + **") negotiatedValue: "** + (currentValue1 + currentValue2) / 2 + **" utility1 "**+ utilities1[i][j]+**"utility2 "**+utilities2[i][j] );

120

121 **if** ((i == 0) && (j == 0) && (x < currentValue1)) x = currentValue1;

122

123 }

124 }

125 }

126 System.out.println(**"Returned "** + p + **" "** + q + **" value:"** + x);

127 **return** (x);

128

129 }

130

131 **public** Map<String, Double> negotiate(Server server, Task task) {

132 RequestedTaskInfo reqTask = task.getRequestedInfo();

133 **double** negotiatedCpu = 0.0d;

134 **double** negotiatedStorage = 0.0d;

135 **double** negotiatedMemory = 0.0d;

136 Map<String, Double> negotiatedValues = **new** HashMap<String, Double>();

137 */\*\*\*Memory\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

138 **if** (server.getAssociatedMemory().getMaxAcceptableValue() - (server.getAssociatedMemory().getUsed() + reqTask.getMemoryMinAcceptableValue()) < 0) {

139 **int** optimalValueForServer = server.getAssociatedMemory().getMaxAcceptableValue() - server.getAssociatedMemory().getUsed();

140 **if** (optimalValueForServer == 0) {

141 optimalValueForServer++;

142 }

143 negotiatedMemory = negotiateGivenRanges(reqTask.getMemoryMinAcceptableValue(), reqTask.getMemoryMaxAcceptableValue(), 0,

144 reqTask.getMemoryMinAcceptableValue(), server.getAssociatedMemory().getTotal() - server.getAssociatedMemory().getUsed(), optimalValueForServer);

145 negotiatedValues.put(NEGOTIATED\_MEMORY, negotiatedMemory + optimalValueForServer);

146 }

147 */\*\*\*Storage\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

148 **if** (server.getAssociatedStorage().getMaxAcceptableValue() - (server.getAssociatedStorage().getUsed() + reqTask.getStorageMinAcceptableValue()) < 0) {

149 **int** optimalValueForServer = server.getAssociatedStorage().getMaxAcceptableValue() - server.getAssociatedStorage().getUsed();

150 **if** (optimalValueForServer == 0) {

151 optimalValueForServer++;

152 }

153 negotiatedStorage = negotiateGivenRanges(reqTask.getStorageMinAcceptableValue(), reqTask.getStorageMaxAcceptableValue(), 0,

154 server.getAssociatedStorage().getTotal() - server.getAssociatedStorage().getUsed(),

155 reqTask.getStorageMinAcceptableValue(), optimalValueForServer);

156 negotiatedValues.put(NEGOTIATED\_STORAGE, negotiatedStorage + optimalValueForServer);

157 }

158 */\*\*\*Cpu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

159 Collection<Core> cores = server.getAssociatedCPU().getAssociatedCore();

160 Core core = cores.iterator().next();

161 **if** (core.getMaxAcceptableValue() - (core.getUsed() + reqTask.getCpuMinAcceptableValue()) < 0) {

162 **int** optimalValueForServer = core.getMaxAcceptableValue() - core.getUsed();

163 **if** (optimalValueForServer == 0) {

164 optimalValueForServer++;

165 }

166 negotiatedCpu = negotiateGivenRanges(reqTask.getCpuMinAcceptableValue() - optimalValueForServer, reqTask.getCpuMaxAcceptableValue() - optimalValueForServer,

167 0, core.getTotal() - core.getUsed() - optimalValueForServer, 0, 0);

168 System.out.println(**"Negotiated value "** + (negotiatedCpu + optimalValueForServer));

169 negotiatedValues.put(NEGOTIATED\_CPU, negotiatedCpu + optimalValueForServer);

170 }

171

172

173 **return** negotiatedValues;

174 }

175 }

176

|  |
| --- |
| ReinforcementLearningDataCenterManagementBehavior.java |

1 */\**

2  *\* To change this template, choose Tools | Templates*

3  *\* and open the template in the editor.*

4  *\*/*

5 **package** contextawaremodel.agents.behaviours;

6

7 **import** actionselection.command.Command;

8 **import** actionselection.command.selfHealingCommand.IncrementCommand;

9 **import** actionselection.command.selfOptimizingCommand.\*;

10 **import** actionselection.context.ContextSnapshot;

11 **import** actionselection.context.DatacenterMemory;

12 **import** actionselection.context.DatacenterMockupContext;

13 **import** actionselection.context.Memory;

14 **import** actionselection.utils.MessageDispatcher;

15 **import** actionselection.utils.Pair;

16 **import** benchmark.RandomWorkLoadGenerator;

17 **import** benchmark.TaskLifeManager;

18 **import** benchmark.WorkLoadFileIO;

19 **import** benchmark.WorkLoadLoader;

20 **import** com.hp.hpl.jena.ontology.OntModel;

21 **import** contextawaremodel.GlobalVars;

22 **import** contextawaremodel.agents.ReinforcementLearningAgent;

23 **import** edu.stanford.smi.protegex.owl.jena.JenaOWLModel;

24 **import** edu.stanford.smi.protegex.owl.model.OWLModel;

25 **import** edu.stanford.smi.protegex.owl.swrl.model.SWRLFactory;

26 **import** greenContextOntology.\*;

27 **import** greenContextOntology.Component;

28 **import** greenContextOntology.impl.DefaultServer;

29 **import** greenContextOntology.impl.DefaultTask;

30 **import** jade.core.Agent;

31 **import** jade.core.behaviours.TickerBehaviour;

32 **import** negotiator.Negotiator;

33 **import** negotiator.impl.NegotiatorFactory;

34 **import** org.apache.log4j.Logger;

35 **import** selfHealingOntology.SelfHealingProtegeFactory;

36

37 **import** java.awt.\*;

38 **import** java.io.IOException;

39 **import** java.util.\*;

40 **import** java.util.Queue;

41

42 */\*\**

43  *\** ***@author*** *Administrator*

44  *\*/*

45 **public class** ReinforcementLearningDataCenterManagementBehavior **extends** TickerBehaviour {

46

47 **private** OWLModel datacenterOwlModel;

48 **private** OntModel datacenterPolicyConversionModel;

49 **private** OntModel selfHealingPolicyConversionModel;

50

51 **private** JenaOWLModel selfHealingOwlModel;

52 **private** ContextSnapshot smallestEntropyContext;

53 **private** Memory memory;

54 **private** DatacenterMemory datacenterMemory;

55 **private** ReinforcementLearningAgent agent;

56 **private boolean** contextBroken = **false**;

57 **private** ProtegeFactory protegeFactory;

58 **private** SWRLFactory swrlFactory;

59 **private** Logger logger;

60 **private** Negotiator negotiator;

61 *// private TaskManagement taskManagementWindow;*

62

63 **public** ReinforcementLearningDataCenterManagementBehavior(Agent a, **int** interval, OWLModel datacenterOwlModel, OntModel datacenterPolicyConversionModel,

64 JenaOWLModel jenaDatacenterOwlModel, OntModel selfHealingPolicyConversionModel,

65 JenaOWLModel selfHealingOwlModel, Memory memory, DatacenterMemory datacenterMemory) {

66 **super**(a, interval);

67 agent = (ReinforcementLearningAgent) a;

68 **this**.datacenterOwlModel = datacenterOwlModel;

69 **this**.datacenterPolicyConversionModel = datacenterPolicyConversionModel;

70 **this**.selfHealingPolicyConversionModel = selfHealingPolicyConversionModel;

71 **this**.selfHealingOwlModel = selfHealingOwlModel;

72 **this**.datacenterMemory = datacenterMemory;

73 **this**.memory = memory;

74 protegeFactory = **new** ProtegeFactory(datacenterOwlModel);

75 Collection<Task> tasks = protegeFactory.getAllTaskInstances();

76

77 **try** {

78 Thread.sleep(10000);

79 } **catch** (InterruptedException e) {

80 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

81 }

82 **for** (Task task : tasks) {

83 TaskLifeManager.addTask(protegeFactory, task, 10, datacenterPolicyConversionModel);

84 }

85

86 WorkLoadLoader generator = RandomWorkLoadGenerator.generateRandomWorkLoad(3, 3, 1000, protegeFactory, datacenterPolicyConversionModel);

87 **try** {

88 WorkLoadFileIO.writeWorkLoadToFile(generator, **"WorkLoadFile"**);

89 } **catch** (IOException e) {

90 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

91 }

92

93

94 **this**.memory = memory;

95 swrlFactory = **new** SWRLFactory(datacenterOwlModel);

96

97 negotiator = NegotiatorFactory.getFuzzyLogicNegotiator();

98

99

100 agent.sendAllTasksToClient();

101

102

103

104

105 logger = Logger.getLogger(ReinforcementLearningDataCenterManagementBehavior.**class**);

106 }

107

108 *// function for computing the contribution to the entropy of task @param task*

109

110 **private double** taskRespectanceDegree(Task task) {

111 ReceivedTaskInfo received = task.getReceivedInfo();

112 RequestedTaskInfo requested = task.getRequestedInfo();

113

114 *//TODO : poate trebe bagat si minimu in task respectance cum is range-uri*

115

116 **double** respectance = task.getCpuWeight()

117 \* (requested.getCores() - received.getCores() + requested.getCpuMaxAcceptableValue() - received.getCpuReceived())

118 + task.getMemoryWeight() \* (requested.getMemoryMaxAcceptableValue() - received.getMemoryReceived())

119 + task.getStorageWeight() \* (requested.getStorageMaxAcceptableValue() - received.getStorageReceived());

120 **return** respectance;

121 }

122

123 **private double** energyRespectanceDegree(Server server) {

124 **double** respectance = 0.0;

125 CPU cpu = server.getAssociatedCPU();

126 greenContextOntology.Memory serverMemory = server.getAssociatedMemory();

127 Storage storage = server.getAssociatedStorage();

128 **double** cpuCores = 0.0;

129 Collection<Component> cores = cpu.getAssociatedCore();

130 **double** diff = 0.0;

131 **for** (Component core : cores) {

132 diff = 0.0;

133 **int** usedCore = core.getUsed();

134 **int** coreMaxAcceptableValue = core.getMaxAcceptableValue();

135

136 **if** (usedCore > coreMaxAcceptableValue) {

137 diff = usedCore - coreMaxAcceptableValue;

138 } **else if** (usedCore < coreMinAcceptableValue) {

140 diff = coreMinAcceptableValue - usedCore;

141 }

142 cpuCores += diff;

143 }

144 cpuCores /= cores.size();

145 respectance += cpu.getWeight() \* cpuCores;

146 diff = 0.0;

147

148 **int** usedMemory = serverMemory.getUsed();

149 **int** memoryMaxAcceptableValue = serverMemory.getMaxAcceptableValue();

150 **int** memoryMinAcceptableValue = serverMemory.getMinAcceptableValue();

151

152 **if** (usedMemory > memoryMaxAcceptableValue) {

153 diff = usedMemory - memoryMaxAcceptableValue;

154 } **else if** (usedMemory < memoryMinAcceptableValue) {

155 diff = usedMemory - memoryMinAcceptableValue;

156 }

157 respectance += serverMemory.getWeight() \* diff;

158 diff = 0.0;

159

160 **int** usedStorage = storage.getUsed();

161 **int** storageMaxAcceptableValue = storage.getMaxAcceptableValue();

162 **int** storageMinAcceptableValue = storage.getMinAcceptableValue();

163

164 **if** (usedStorage > storageMaxAcceptableValue) {

165 diff = usedStorage - storageMaxAcceptableValue;

166 } **else if** (usedStorage < storageMinAcceptableValue) {

168 diff = storageMinAcceptableValue - usedStorage;

169 }

170

171 respectance += storage.getWeight() \* diff;

172 **return** respectance;

173 }

174

175 **private** Pair<Double, Policy> computeEntropy() {

176 Policy brokenPolicy = **null**;

177 **double** entropy = 0.0;

178 Collection<QoSPolicy> qosPolicies = protegeFactory.getAllQoSPolicyInstances();

179

180 **for** (QoSPolicy policy : qosPolicies) {

181

182 Task task = policy.getReferenced();

183 *//if task has been deleted*

184 **if** (task == **null**) {

185 **continue**;

186 }

187

189 **if** (!policy.getRespected(datacenterPolicyConversionModel)) {

193 **if** (brokenPolicy == **null**) {

194 brokenPolicy = policy;

195 }

196

197 **if** (policy.hasPriority()) {

198 entropy += policy.getPriority() \* taskRespectanceDegree(task);

199 }

200 }

201 }

202

203

204 **if** (brokenPolicy == **null**) {

205

206 Collection<EnergyPolicy> policies = protegeFactory.getAllEnergyPolicyInstances();

207 **for** (EnergyPolicy policy : policies) {

208 Server server = policy.getReferenced();

209 **if** (!policy.getRespected(datacenterPolicyConversionModel)) {

210 **if** (brokenPolicy == **null**) {

211 brokenPolicy = policy;

212 }

213 **if** (policy.hasPriority()) {

214 entropy += policy.getPriority() \* energyRespectanceDegree(server);

215 }

216

217 }

218 }

219 }

220 **return new** Pair<Double, Policy>(entropy, brokenPolicy);

221 }

222

223 **private double** computeRewardFunction(ContextSnapshot previous, ContextSnapshot current, Command c) {

224 **double** function = 0.0d;

225 **if** (previous != **null**) {

226 function += previous.getRewardFunction();

227 **double** temp = previous.getContextEntropy() - current.getContextEntropy() - c.getCost() - current.getActions().size();

228 function += ContextSnapshot.gamma \* temp;

229 } **else** {

230 function -= current.getContextEntropy();

231 }

232

233 **return** function;

234 }

235

236

237 **private** ContextSnapshot reinforcementLearning(PriorityQueue<ContextSnapshot> queue) {

238

239 ContextSnapshot newContext = queue.poll();

240 **if** (newContext == **null**) {

241 Pair<Double, Policy> entropyAndPolicy = computeEntropy();

242 System.out.println(**"Could not repair the context totally. Returning best solution."**);

243 System.out.println(**"Broken "** + entropyAndPolicy.getSecond().getLocalName() + **"\n Referenced "** + entropyAndPolicy.getSecond().getReferenced().toString());

244 **return** smallestEntropyContext;

245 }

246 System.out.println(**"---A"**);

247 Collection<Server> servers = protegeFactory.getAllServerInstances();

248 newContext.executeActions(datacenterPolicyConversionModel);

249 datacenterMemory.restoreProtegeFactory(protegeFactory);

250

251 Queue<Command> commands = datacenterMemory.getActionsForTasks(protegeFactory);

252 **if** (commands != **null**) {

253 System.out.println(**"Remembered...!!"**);

254 newContext.addActions(commands);

255 }

256 Pair<Double, Policy> entropyAndPolicy = computeEntropy();

257

258 **if** (smallestEntropyContext != **null**) {

259 **if** (entropyAndPolicy.getFirst() < smallestEntropyContext.getContextEntropy())

260 smallestEntropyContext = newContext;

261 } **else** {

262 smallestEntropyContext = newContext;

263 }

264

265 System.out.println(**"\n"** + entropyAndPolicy.getFirst() + **" "** + newContext.getRewardFunction() + **" "** + entropyAndPolicy.getSecond() + **"\n"**);

266 System.out.println(**"---B"**);

267 DefaultTask task = **null**;

268 DefaultServer server = **null**;

269

270 **if** (entropyAndPolicy.getFirst() > 0) {

271 **if** (entropyAndPolicy.getSecond() != **null**) {

272 Policy policy = entropyAndPolicy.getSecond();

273 **if** (policy **instanceof** QoSPolicy) {

274 task = (DefaultTask) policy.getReferenced();

275 } **else** {

276 server = (DefaultServer) policy.getReferenced();

277 }

278 }

279 **boolean** deployed = **false**; */// sa zica daca ii deployed sau nu*

280 **if** (task != **null**) {

281 *// deploy actions*

282 **for** (Server serverInstance : servers) {

283 **if** (!serverInstance.getIsInLowPowerState() && serverInstance.hasResourcesFor(task)

284 && !serverInstance.containsTask(task) && !task.isRunning()) {

285 SelfOptimizingCommand newAction = **new** DeployTaskCommand(protegeFactory, serverInstance.getName(), task.getName());

286 **if** (!newContext.getActions().contains(newAction)) {

287 ContextSnapshot cs = **new** ContextSnapshot(**new** LinkedList(newContext.getActions()));

288 cs.getActions().add(newAction);

289

290 newAction.execute(datacenterPolicyConversionModel);

291

292 Double afterExecuteEntropy = computeEntropy().getFirst();

293 cs.setContextEntropy(afterExecuteEntropy);

294 cs.setRewardFunction(computeRewardFunction(newContext, cs, newAction));

295 newAction.rewind(datacenterPolicyConversionModel);

296

297 queue.add(cs);

298 }

299 }

300 }

301

302 *// move actions*

303 Collection<Server> servers1 = protegeFactory.getAllServerInstances();

304 **for** (Server serverInstance : servers) {

305 **if** (!serverInstance.getIsInLowPowerState()) {

306 Iterator it = serverInstance.listRunningTasks();

307 **while** (it.hasNext()) {

308 Task myTask = (DefaultTask) it.next();

309 **for** (Server otherServerInstance : servers1) {

310 **if** (!otherServerInstance.getIsInLowPowerState() && otherServerInstance.hasResourcesFor(myTask)

311 && !otherServerInstance.containsTask(myTask)) {

312 SelfOptimizingCommand newAction = **new** MoveTaskCommand(protegeFactory, serverInstance.getName(), otherServerInstance.getName(), myTask.getName());

313

314 ContextSnapshot cs = **new** ContextSnapshot(**new** LinkedList(newContext.getActions()));

315 *//if action is not already in the actions list*

316 **if** (!cs.getActions().contains(newAction)) {

317 cs.getActions().add(newAction);

318 newAction.execute(datacenterPolicyConversionModel);

319

320 cs.setContextEntropy(computeEntropy().getFirst());

321 cs.setRewardFunction(computeRewardFunction(newContext, cs, newAction));

322 newAction.rewind(datacenterPolicyConversionModel);

323

324 queue.add(cs);

325 }

326 }

327 }

328 }

329 }

330 }

331 }

332 **if** (server != **null**) {

333 Iterator it = server.listRunningTasks();

334 *// move tasks from server*

335 **while** (it.hasNext()) {

336 Task myTask = (DefaultTask) it.next();

337 **for** (Server serverInstance : servers) {

338 **if** (!serverInstance.getIsInLowPowerState() && !serverInstance.containsTask(myTask)

339 && serverInstance.hasResourcesFor(myTask)) {

340 Command newAction = **new** MoveTaskCommand(protegeFactory, server.getName(), serverInstance.getName(), myTask.getName());

341 **if** (!newContext.getActions().contains(newAction)) {

342 ContextSnapshot cs = **new** ContextSnapshot(**new** LinkedList(newContext.getActions()));

343 cs.getActions().add(newAction);

344

345 newAction.execute(datacenterPolicyConversionModel);

346 cs.setContextEntropy(computeEntropy().getFirst());

347 cs.setRewardFunction(computeRewardFunction(newContext, cs, newAction));

348 newAction.rewind(datacenterPolicyConversionModel);

349

350 queue.add(cs);

351 }

352 }

353 }

354 }

355 }

356 *// wake up*

357

358 **for** (Server serverInstance : servers) {

359 **if** (serverInstance.getIsInLowPowerState()) { *//&& (task!=null) && serverInstance.hasResourcesFor(task)) {*

360 System.out.println(serverInstance.getLocalName() + **" "** + serverInstance.getIsInLowPowerState() + **" is waking up"**);

361 Command newAction = **new** WakeUpServerCommand(protegeFactory, serverInstance.getName());

362 ContextSnapshot cs = **new** ContextSnapshot(**new** LinkedList(newContext.getActions()));

363 *//if action is not already in the actions list*

364 **if** (!cs.getActions().contains(newAction)) {

365 cs.getActions().add(newAction);

366

367 newAction.execute(datacenterPolicyConversionModel);

368 cs.setContextEntropy(computeEntropy().getFirst());

369 cs.setRewardFunction(computeRewardFunction(newContext, cs, newAction));

370 newAction.rewind(datacenterPolicyConversionModel);

371

372 queue.add(cs);

373 }

374 }

375 }

376 *// sleep*

377 **for** (Server serverInstance : servers) {

378 **if** (!serverInstance.getIsInLowPowerState() && !serverInstance.hasRunningTasks()) {

379 Command newAction = **new** SendServerToLowPowerStateCommand(protegeFactory, serverInstance.getName());

380

381 **if** (!newContext.getActions().contains(newAction)) {

382 ContextSnapshot cs = **new** ContextSnapshot(**new** LinkedList(newContext.getActions()));

383 cs.getActions().add(newAction);

384

385 newAction.execute(datacenterPolicyConversionModel);

386 cs.setContextEntropy(computeEntropy().getFirst());

387 cs.setRewardFunction(computeRewardFunction(newContext, cs, newAction));

388

389 newAction.rewind(datacenterPolicyConversionModel);

390 queue.add(cs);

391 }

392 }

393 }

394

395 newContext.rewind(datacenterPolicyConversionModel);

396

397 newContext = reinforcementLearning(queue);

398 } **else** {

399 newContext.rewind(datacenterPolicyConversionModel);

400 }

401 **return** newContext;

402 }

403

404 **protected** Server getMinDistanceServer(Task task) {

405 Server retServer = **null**;

406 RequestedTaskInfo requestedTask = task.getRequestedInfo();

407 **double** difference;

408 **double** minDif = 10000000.0d;

409 Collection<Server> servers = protegeFactory.getAllServerInstances();

410 **for** (Server server : servers) {

411 Collection<Core> cores = server.getAssociatedCPU().getAssociatedCore();

412 difference = 0.0d;

413 **for** (Core core : cores) {

414 difference += Math.pow(core.getTotal() - core.getUsed() - requestedTask.getCpuMinAcceptableValue(), 2);

415 }

416 greenContextOntology.Memory mem = server.getAssociatedMemory();

417 difference += Math.pow(mem.getTotal() - mem.getUsed() - requestedTask.getMemoryMinAcceptableValue(), 2);

418 Storage storage = server.getAssociatedStorage();

419 difference += Math.pow(storage.getTotal() - storage.getUsed() - requestedTask.getStorageMinAcceptableValue(), 2);

420 difference = Math.sqrt(difference);

421 **if** (server.hasResourcesToBeNegotiatedFor(task))

422 **if** (difference < minDif) {

423 minDif = difference;

424 retServer = server;

425 }

426 }

427 **return** retServer;

428

429 }

430

431 @Override

432 **protected void** onTick() {

433

434 smallestEntropyContext = **null**;

435

436 TaskLifeManager.kill(protegeFactory, datacenterPolicyConversionModel);

437

438 DatacenterMockupContext initialDataCenterContext = **new** DatacenterMockupContext();

439 initialDataCenterContext.createMockupContextFromOntology(protegeFactory);

440 System.out.println(**"Datacenter behavior on Tick"**);

441 PriorityQueue<ContextSnapshot> queue = **new** PriorityQueue<ContextSnapshot>();

442 ContextSnapshot initialContext = **new** ContextSnapshot(**new** LinkedList<Command>());

443 Pair<Double, Policy> entropyAndPolicy = computeEntropy();

444

445 System.out.println(entropyAndPolicy.getFirst() + **" "** + entropyAndPolicy.getSecond());

446

447 initialContext.setContextEntropy(entropyAndPolicy.getFirst());

448 initialContext.setRewardFunction(computeRewardFunction(**null**, initialContext, **null**));

449 queue.add(initialContext);

450 *// resultsFrame.setActionsList(null);*

451

452 **if** (entropyAndPolicy.getSecond() != **null**) {

453

454 contextBroken = **true**;

455

456 *//TODO: activate only after a deploy or delete to recollect*

457 *//if context broken gather the extra resources allocated to tasks in order to properly evaluate the context*

458 **for** (Server server : protegeFactory.getAllServerInstances()) {

459 server.collectPreviouslyDistributedResources(datacenterPolicyConversionModel);

460 }

461

462 *//Gather data for logging purposes*

463

464 ArrayList<String> brokenQoSPolicies = **new** ArrayList<String>();

465 Collection<QoSPolicy> qoSPolicies = protegeFactory.getAllQoSPolicyInstances();

466 **for** (Policy policy : qoSPolicies) {

467 **if** (policy.getReferenced() == **null**) {

468 **continue**;

469 }

470 **if** (!policy.getRespected(datacenterPolicyConversionModel)) {

471 brokenQoSPolicies.add(policy.getName().split(**"#"**)[1]);

472 }

473 }

474 Collection<EnergyPolicy> brokenEnergyPolicies = protegeFactory.getAllEnergyPolicyInstances();

475 **for** (Policy policy : brokenEnergyPolicies) {

476 **if** (!policy.getRespected(datacenterPolicyConversionModel)) {

477 brokenQoSPolicies.add(policy.getName().split(**"#"**)[1]);

478 }

479 }

480 **try** {

481 MessageDispatcher.sendMessage(agent, GlobalVars.GUIAGENT\_NAME, **new** Object[]{**"DatacenterLogger"**, Color.ORANGE, **"Broken policies"**, brokenQoSPolicies});

482 } **catch** (IOException e) {

483 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

484 }

485 *//agent.getSelfOptimizingLogger().log(Color.ORANGE, "Broken policies", brokenQoSPolicies);*

486

487 Collection<Server> servers = protegeFactory.getAllServerInstances();

488 Collection<Task> tasks = protegeFactory.getAllTaskInstances();

489 ArrayList<String> currentState = **new** ArrayList(servers.size() + tasks.size());

490

491 **for** (Server server : servers) {

492 currentState.add(server.toString());

493 }

494

495 **for** (Task task : tasks) {

496 currentState.add(task.toString());

497 }

498 **try** {

499 MessageDispatcher.sendMessage(agent, GlobalVars.GUIAGENT\_NAME, **new** Object[]{**"DatacenterLogger"**, Color.red, **"Current state"**, currentState});

500 } **catch** (IOException e) {

501 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

502 }

503 *// agent.getSelfOptimizingLogger().log(Color.red, "Current state", currentState);*

504

505 **try** {

506 MessageDispatcher.sendMessage(agent, GlobalVars.GUIAGENT\_NAME, **new** Object[]{**"DatacenterLogger"**, Color.red, **"Current state"**, currentState});

507 } **catch** (IOException e) {

508 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

509 }

510

511 ContextSnapshot result = reinforcementLearning(queue);

512

513 Collection<Command> resultQueue = result.getActions();

514 ArrayList<String> message = **new** ArrayList<String>();

515 **for** (Command o : resultQueue) {

516 message.add(o.toString());

517 System.out.println(o.toString());

518 o.execute(datacenterPolicyConversionModel);

519 o.executeOnX3D(agent);

520 o.executeOnWebService(); *//---> to be decommented when running on servers*

521 **try** {

522 Thread.sleep(3000);

523 } **catch** (InterruptedException e) {

524 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

525 }

526

527 }

528

529 **if** (result.getContextEntropy() > 0) {

530 Negotiator negotiator = NegotiatorFactory.getFuzzyLogicNegotiator();

531 System.out.println(**"Negotiating...."**);

532 Collection<Task> allTasks = protegeFactory.getAllTaskInstances();

533 **for** (Task task : allTasks) {

534 **if** (!task.isRunning()) {

535 ContextSnapshot cs = **new** ContextSnapshot(**new** LinkedList(result.getActions()));

536

537 Server server = getMinDistanceServer(task);

538 **if** (server == **null**) {

539 **continue**;

540 }

541 NegotiateResourcesCommand negotiateResourcesCommand = **new** NegotiateResourcesCommand(protegeFactory, negotiator, server.getServerName(), task.getName());

542 negotiateResourcesCommand.execute(datacenterPolicyConversionModel);

543 negotiateResourcesCommand.executeOnWebService();

544

545 cs.getActions().add(negotiateResourcesCommand);

546 cs.setContextEntropy(computeEntropy().getFirst());

547

548 **if** (cs.getContextEntropy() < result.getContextEntropy()) {

549 result = cs;

550 }

551

552 }

553 }

554

555 datacenterMemory.memorize(initialDataCenterContext, result.getActions());

556

557 **for** (Server server : servers) {

558 server.distributeRemainingResources(datacenterPolicyConversionModel);

559 }

560 }

561

562 datacenterMemory.memorize(initialDataCenterContext, result.getActions());

563

564

565 **try** {

566 MessageDispatcher.sendMessage(agent, GlobalVars.GUIAGENT\_NAME, **new** Object[]{**"DatacenterLogger"**, Color.BLUE, **"Corrective actions"**, currentState});

567 } **catch** (IOException e) {

568 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

569 }

570 **int** resultsSize = resultQueue.size();

571 **if** (resultsSize > 0) {

572

573 *//datacenter load influences temperature*

574 SelfHealingProtegeFactory selfHealingProtegeFactory = **new** SelfHealingProtegeFactory(selfHealingOwlModel);

575 IncrementCommand c = **new** IncrementCommand(selfHealingProtegeFactory, selfHealingProtegeFactory.getSensor(**"TemperatureSensorI"**).getName(), resultsSize);

576

577 System.out.println(**"\nDatacenter load temperature influence: "**);

578 System.out.println(c);

579 c.execute(selfHealingPolicyConversionModel);

580 c.executeOnX3D(myAgent);

581

582 }

583 *//wait for effect to be noticeable on X3D*

584

585 **try** {

586 Thread.sleep(3000);

587 } **catch** (InterruptedException e) {

588 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

589 }

590

591

592 } **else** {

593 **if** (contextBroken) {

594 contextBroken = **false**;

595 Collection<Server> servers = protegeFactory.getAllServerInstances();

596 Collection<Task> tasks = protegeFactory.getAllTaskInstances();

597 ArrayList<String> currentState = **new** ArrayList(servers.size() + tasks.size());

598

599 **for** (Server server : servers) {

600 currentState.add(server.toString());

601 }

602

603 **for** (Task task : tasks) {

604 currentState.add(task.toString());

605 }

606

607 **try** {

608 MessageDispatcher.sendMessage(agent, GlobalVars.GUIAGENT\_NAME, **new** Object[]{**"DatacenterLogger"**, Color.GREEN, **"Current state"**, currentState});

609 } **catch** (IOException e) {

610 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

611 }

612

613 }

614 }

615

616 }

617

618 }

|  |
| --- |
| TaskManagementAgent.java |

1 **package** contextawaremodel.agents;

2

3 **import** contextawaremodel.GlobalVars;

4 **import** contextawaremodel.agents.behaviours.ReceiveMessageTMBehaviour;

5 **import** contextawaremodel.worldInterface.dtos.TaskDto;

6 **import** contextawaremodel.gui.TaskManagement;

7

8 **import** jade.core.AID;

9 **import** jade.core.Agent;

10 **import** jade.lang.acl.ACLMessage;

11

12 **import** java.io.IOException;

13 **import** java.io.Serializable;

14

15

16 */\*\**

17  *\* Created by IntelliJ IDEA.*

18  *\* User: Me*

19  *\* Date: Apr 15, 2010*

20  *\* Time: 2:18:33 PM*

21  *\* To change this template use File | Settings | File Templates.*

22  *\*/*

23 **public class** TaskManagementAgent **extends** Agent {

24 **private** TaskManagement taskManagementWindow;

25

26 **protected void** setup() {

27 System.out.println(**"Task Management Agent "** + getLocalName() + **" started."**);

28 **this**.addBehaviour(**new** ReceiveMessageTMBehaviour(**this**));

29 taskManagementWindow = **new** TaskManagement(**this**);

30 taskManagementWindow.setVisible(**true**);

31 }

32

33 **public void** reenableTasks() {

34 taskManagementWindow.reenableTasksList();

35 }

36

37 **public void** clearFields() {

38 taskManagementWindow.setEmptyFields();

39 }

40

41 **public void** populateTaskWindow(TaskDto[] tasks) {

42 taskManagementWindow.setTasks(tasks);

43 }

44

45

46 **public void** sendTaskMessageToRL(Object message, **int** aCode) {

47 ACLMessage msg;

48

49 System.out.println(**"Code"** + aCode);

50 **switch** (aCode) {

51 **case** GlobalVars.INDIVIDUAL\_CREATED:

52 msg = **new** ACLMessage(ACLMessage.INFORM);

53 **try** {

54 msg.setContentObject((Serializable) message);

55 } **catch** (IOException e) {

56 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

57 }

58 **break**;

59 **case** GlobalVars.INDIVIDUAL\_DELETED:

60 msg = **new** ACLMessage(ACLMessage.INFORM\_IF);

61 msg.setContent((String) message);

62

63 **break**;

64 **case** GlobalVars.INDIVIDUAL\_MODIFIED:

65 msg = **new** ACLMessage(ACLMessage.INFORM\_REF);

66 **break**;

67 **default**:

68 msg = **null**;

69 **break**;

70 }

71

72 **if** (msg != **null**) {

73 msg.addReceiver(**new** AID(GlobalVars.RLAGENT\_NAME + **"@"** + **this**.getContainerController().getPlatformName()));

74

75 System.out.println(**"Sending"** + msg);

76 **this**.send(msg);

77 }

78 }

79 }

80

|  |
| --- |
| DeployTaskCommand.java |

1 */\**

2  *\* To change this template, choose Tools | Templates*

3  *\* and open the template in the editor.*

4  *\*/*

5 **package** actionselection.command.selfOptimizingCommand;

6

7 **import** actionselection.utils.MessageDispatcher;

8 **import** benchmark.TaskLifeManager;

9 **import** com.hp.hpl.jena.ontology.OntModel;

10 **import** contextawaremodel.GlobalVars;

11 **import** contextawaremodel.agents.X3DAgent;

12 **import** contextawaremodel.worldInterface.datacenterInterface.proxies.ServerManagementProxyInterface;

13 **import** contextawaremodel.worldInterface.datacenterInterface.proxies.impl.ProxyFactory;

14 **import** greenContextOntology.ProtegeFactory;

15 **import** greenContextOntology.Server;

16 **import** greenContextOntology.Task;

17 **import** jade.core.Agent;

18

19 **import** java.io.IOException;

20 **import** java.util.UUID;

21

22 */\*\**

23  *\** ***@author*** *Me*

24  *\*/*

25 **public class** DeployTaskCommand **extends** SelfOptimizingCommand {

26

27 **private** String serverName;

28 **private** String taskName;

29

30 **public** DeployTaskCommand(ProtegeFactory protegeFactory, String serverName, String taskName) {

31 **super**(protegeFactory);

32 **this**.serverName = serverName;

33 **this**.taskName = taskName;

34 cost = 100;

35 }

36

37 */\*\**

38  *\* Sets the task <code>taskName</code> associated server to <code>serverName</code>*

39  *\*/*

40 @Override

41 **public void** execute(OntModel model) {

42 Server server = protegeFactory.getServer(serverName);

43 Task task = protegeFactory.getTask(taskName);

44 task.setAssociatedServer(server);

45 server.addRunningTasks(task, model);

46 }

47

48 */\*\**

49  *\* Implements Undo capability*

50  *\*/*

51 @Override

52 **public void** rewind(OntModel model) {

53 Server server = protegeFactory.getServer(serverName);

54 Task task = protegeFactory.getTask(taskName);

55 server.removeRunningTasks(task, model);

56 }

57

58 @Override

59 **public** String toString() {

60 String description;

61 description = **"Deploy task \""** + taskName.split(**"#"**)[1] + **"\" to server \""** + serverName.split(**"#"**)[1] + **"\""**;

62 **return** description;

63 }

64

65 @Override

66 **public void** executeOnWebService() {

67 Server server = protegeFactory.getServer(serverName);

68 Task task = protegeFactory.getTask(taskName);

69 ServerManagementProxyInterface proxy = ProxyFactory.createServerManagementProxy(server.getServerIPAddress());

70 **if** (proxy != **null**) {

71 String path = (String) server.getVirtualMachinesPath().iterator().next();

72 proxy.deployVirtualMachine(**"\\\\192.168.2.110\\SharedStorage"**,

73 **"\\\\192.168.2.110\\SharedStorage\\"** + server.getServerName(),

74 task.getTaskName(), task.getLocalName());

75

76 } **else** {

77 System.err.println(**"Proxy is null"**);

78 }

79 TaskLifeManager.startTaskTimer(task);

80 *// throw new UnsupportedOperationException("Not supported yet.");*

81 }

82

83 @Override

84 **public** String[] toStringArray() {

85 String[] array = **new** String[3];

86 array[0] = **"Deploy"**;

87 array[1] = taskName.split(**"#"**)[1];

88 array[2] = serverName.split(**"#"**)[1];

89 **return** array;

90 }

91

92 **public void** executeOnX3D(Agent agent) {

93 Server server = protegeFactory.getServer(serverName);

94 **try** {

95 MessageDispatcher.sendMessage(agent, GlobalVars.X3DAGENT\_NAME, **new** Object[]{X3DAgent.ADD\_TASK\_COMMAND, taskName.split(**"#"**)[1], serverName.split(**"#"**)[1], server.getRunningTasks().size() + 1});

96

97

98 } **catch** (IOException e) {

99 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

100 }

101

102 }

103

104

105 **public void** rewindOnX3D(Agent agent) {

106 **try** {

107 MessageDispatcher.sendMessage(agent, GlobalVars.X3DAGENT\_NAME, **new** Object[]{X3DAgent.REMOVE\_TASK\_COMMAND, taskName.split(**"#"**)[1]});

108

109

110 } **catch** (IOException e) {

111 e.printStackTrace(); *//To change body of catch statement use File | Settings | File Templates.*

112 }

113

114 }

115

116 **public** String getServerName() {

117 **return** serverName;

118 }

119

120 **public void** setServerName(String serverName) {

121 **this**.serverName = serverName;

122 }

123

124 **public** String getTaskName() {

125 **return** taskName;

126 }

127

128 **public void** setTaskName(String taskName) {

129 **this**.taskName = taskName;

130 }

131 }

132

|  |
| --- |
| TaskDto.java |

1 **package** contextawaremodel.worldInterface.dtos;

2

3 **import** greenContextOntology.ReceivedTaskInfo;

4 **import** greenContextOntology.RequestedTaskInfo;

5 **import** greenContextOntology.Task;

6

7 **import** java.io.Serializable;

8

9 */\*\**

10  *\* Created by IntelliJ IDEA.*

11  *\* User: GEORGY*

12  *\* Date: May 12, 2010*

13  *\* Time: 11:44:23 AM*

14  *\* To change this template use File | Settings | File Templates.*

15  *\*/*

16 **public class** TaskDto **implements** Serializable {

17 **private** String taskName;

18 **private int** requestedCores;

19 **private int** requestedCPUMax;

20 **private int** requestedCPUMin;

21 **private int** requestedMemoryMax;

22 **private int** requestedMemoryMin;

23 **private int** requestedStorageMax;

24 **private int** requestedStorageMin;

25

26 **private int** receivedCores;

27 **private int** receivedCPU;

28 **private int** receivedMemory;

29 **private int** receivedStorage;

30

31 **private boolean** isRunning;

32

33 @Override

34 **public boolean** equals(Object task) {

35 **if** (**this** == task) **return true**;

36 **if** (task **instanceof** Task) {

37 Task t = (Task) task;

38 RequestedTaskInfo rti = t.getRequestedInfo();

39 **if** (requestedCores != rti.getCores()) **return false**;

40 **if** (requestedCPUMax != rti.getCpuMaxAcceptableValue()) **return false**;

41 **if** (requestedCPUMin != rti.getCpuMinAcceptableValue()) **return false**;

42 **if** (requestedMemoryMax != rti.getMemoryMaxAcceptableValue()) **return false**;

43 **if** (requestedMemoryMin != rti.getMemoryMinAcceptableValue()) **return false**;

44 **if** (requestedStorageMax != rti.getStorageMaxAcceptableValue()) **return false**;

45 **if** (requestedStorageMin != rti.getStorageMinAcceptableValue()) **return false**;

46

47 ReceivedTaskInfo rci = t.getReceivedInfo();

48 **if** (receivedCores != rci.getCores()) **return false**;

49 **if** (receivedCPU != rci.getCpuReceived()) **return false**;

50 **if** (receivedMemory != rci.getMemoryReceived()) **return false**;

51 **if** (receivedStorage != rci.getStorageReceived()) **return false**;

52 **if** (t.isRunning() == isRunning()) **return false**;

53 }

54 **if** (task **instanceof** TaskDto) {

55 TaskDto taskDto = (TaskDto) task;

56 **if** (requestedCores != taskDto.getRequestedCores()) **return false**;

57 **if** (requestedCPUMax != taskDto.getRequestedCPUMax()) **return false**;

58 **if** (requestedCPUMin != taskDto.getRequestedCPUMin()) **return false**;

59 **if** (requestedMemoryMax != taskDto.getRequestedMemoryMax()) **return false**;

60 **if** (requestedMemoryMin != taskDto.getRequestedMemoryMin()) **return false**;

61 **if** (requestedStorageMax != taskDto.getRequestedStorageMax()) **return false**;

62 **if** (requestedStorageMin != taskDto.getRequestedMemoryMin()) **return false**;

63

64

65 **if** (receivedCores != taskDto.getReceivedCores()) **return false**;

66 **if** (receivedCPU != taskDto.getReceivedCPU()) **return false**;

67 **if** (receivedMemory != taskDto.getReceivedMemory()) **return false**;

68 **if** (receivedStorage != taskDto.getReceivedStorage()) **return false**;

69 **if** (taskDto.isRunning() == isRunning()) **return false**;

70 }

71

72 **return true**;

73 }

74

75 **public** String getTaskName() {

76 **return** taskName;

77 }

78

79 **public void** setTaskName(String taskName) {

80 **this**.taskName = taskName;

81 }

82

83 **public int** getRequestedCores() {

84 **return** requestedCores;

85 }

86

87 **public void** setRequestedCores(**int** requestedCores) {

88 **this**.requestedCores = requestedCores;

89 }

90

91 **public int** getRequestedCPUMax() {

92 **return** requestedCPUMax;

93 }

94

95 **public void** setRequestedCPUMax(**int** requestedCPUMax) {

96 **this**.requestedCPUMax = requestedCPUMax;

97 }

98

99 **public int** getRequestedCPUMin() {

100 **return** requestedCPUMin;

101 }

102

103 **public void** setRequestedCPUMin(**int** requestedCPUMin) {

104 **this**.requestedCPUMin = requestedCPUMin;

105 }

106

107 **public int** getRequestedMemoryMax() {

108 **return** requestedMemoryMax;

109 }

110

111 **public void** setRequestedMemoryMax(**int** requestedMemoryMax) {

112 **this**.requestedMemoryMax = requestedMemoryMax;

113 }

114

115 **public int** getRequestedMemoryMin() {

116 **return** requestedMemoryMin;

117 }

118

119 **public void** setRequestedMemoryMin(**int** requestedMemoryMin) {

120 **this**.requestedMemoryMin = requestedMemoryMin;

121 }

122

123 **public int** getRequestedStorageMax() {

124 **return** requestedStorageMax;

125 }

126

127 **public void** setRequestedStorageMax(**int** requestedStorageMax) {

128 **this**.requestedStorageMax = requestedStorageMax;

129 }

130

131 **public int** getRequestedStorageMin() {

132 **return** requestedStorageMin;

133 }

134

135 **public void** setRequestedStorageMin(**int** requestedStorageMin) {

136 **this**.requestedStorageMin = requestedStorageMin;

137 }

138

139 **public int** getReceivedCores() {

140 **return** receivedCores;

141 }

142

143 **public void** setReceivedCores(**int** receivedCores) {

144 **this**.receivedCores = receivedCores;

145 }

146

147 **public int** getReceivedCPU() {

148 **return** receivedCPU;

149 }

150

151 **public void** setReceivedCPU(**int** receivedCPU) {

152 **this**.receivedCPU = receivedCPU;

153 }

154

155 **public int** getReceivedMemory() {

156 **return** receivedMemory;

157 }

158

159 **public void** setReceivedMemory(**int** receivedMemory) {

160 **this**.receivedMemory = receivedMemory;

161 }

162

163 **public int** getReceivedStorage() {

164 **return** receivedStorage;

165 }

166

167 **public void** setReceivedStorage(**int** receivedStorage) {

168 **this**.receivedStorage = receivedStorage;

169 }

170

171 **public boolean** isRunning() {

172 **return** isRunning;

173 }

174

175 **public void** setRunning(**boolean** running) {

176 isRunning = running;

177 }

178 }

179

|  |
| --- |
| ReceiveMessageRLBehaviour.java |

1 */\**

2  *\* To change this template, choose Tools | Templates*

3  *\* and open the template in the editor.*

4  *\*/*

5 **package** contextawaremodel.agents.behaviours;

6

7 **import** actionselection.command.selfOptimizingCommand.RemoveTaskFromServerCommand;

8 **import** com.hp.hpl.jena.ontology.Individual;

9 **import** com.hp.hpl.jena.ontology.OntModel;

10 **import** com.hp.hpl.jena.rdf.model.Property;

11 **import** com.hp.hpl.jena.rdf.model.RDFNode;

12 **import** contextawaremodel.GlobalVars;

13 **import** contextawaremodel.agents.ReinforcementLearningAgent;

14 **import** contextawaremodel.worldInterface.dtos.TaskDto;

15 **import** edu.stanford.smi.protegex.owl.model.OWLModel;

16 **import** edu.stanford.smi.protegex.owl.model.RDFProperty;

17 **import** edu.stanford.smi.protegex.owl.model.RDFResource;

18 **import** edu.stanford.smi.protegex.owl.swrl.model.SWRLFactory;

19 **import** greenContextOntology.\*;

20 **import** jade.core.Agent;

21 **import** jade.core.behaviours.CyclicBehaviour;

22 **import** jade.lang.acl.ACLMessage;

23

24 */\*\**

25  *\** ***@author*** *Moldovanus*

26  *\*/*

27 **public class** ReceiveMessageRLBehaviour **extends** CyclicBehaviour {

28

29 **private** ReinforcementLearningAgent agent;

30 **private** OWLModel owlModel;

31 **private** OntModel jenaModel;

32 **private** SWRLFactory swrlFactory;

33 **private** ProtegeFactory protegeFactory;

34 **private** OntModel ontModelDataCenter;

35

36 **public** ReceiveMessageRLBehaviour(Agent agent, OWLModel owlModel, OntModel jenaModel, OWLModel owlModelDataCenter, OntModel ontModelDataCenter) {

37 **super**(agent);

38 **this**.protegeFactory = **new** ProtegeFactory(owlModelDataCenter);

39 **this**.agent = (ReinforcementLearningAgent) agent;

40 **this**.owlModel = owlModel;

41 **this**.jenaModel = jenaModel;

42 **this**.ontModelDataCenter = ontModelDataCenter;

43 }

44

45 @Override

46 **public void** action() {

47 ACLMessage message = agent.receive();

48 **if** (message == **null**) {

49 **return**;

50 }

51

52

53

54 **try** {

55 **switch** (message.getPerformative()) {

56 **case** ACLMessage.INFORM\_REF:

57 String individualName = (String) message.getContentObject();

58 **final** RDFResource individual = owlModel.getRDFResource(individualName);

59 **if** (!individual.getProtegeType() .getNamedSuperclasses(**true**). Contains (owlModel.getRDFSNamedClass(**"sensor"**))) {

60 **return**;

61 }

62

63 RDFProperty urlProperty = owlModel.getRDFProperty(**"has-web-service-URI"**);

64 String url = individual.getPropertyValue(urlProperty).toString();

65

66 *//register the web service URL read from the external file into the jena ont model*

67 Individual sensor = jenaModel.getIndividual(GlobalVars.base + **"#"** + individualName);

68 Property urlJenaProperty = jenaModel.getDatatypeProperty(GlobalVars.base + **"#has-web-service-URI"**);

69 sensor.setPropertyValue(urlJenaProperty, jenaModel.createLiteralStatement(

70 sensor, urlJenaProperty, url).getLiteral().as(RDFNode.**class**));

71 **break**;

72

73 **case** ACLMessage.INFORM:

74 String content = (String) message.getContent();

75 **if** (content.equals(**"BROKEN"**)) {

76 agent.setContextIsOK(**false**);

77 } **else if** (content.equals(**"OK"**)) {

78 agent.setContextIsOK(**true**);

79 } **else** {

80 TaskDto taskDto = (TaskDto) message.getContentObject();

81 Task task = protegeFactory.createTask(**"http://www.owl-ontologies.com/Datacenter.owl#"** + taskDto.getTaskName());

82 System.out.println(**"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAa "** + taskDto.getTaskName());

83 QoSPolicy policy = protegeFactory.createQoSPolicy(taskDto.getTaskName() + **"Policy"**);

84

85 RequestedTaskInfo requestedInfo = protegeFactory.createRequestedTaskInfo(taskDto.getTaskName() + **"RequestedInfo"**);

86 ReceivedTaskInfo receivedInfo = protegeFactory.createReceivedTaskInfo(taskDto.getTaskName() + **"ReceivedInfo"**);

87

88

89 requestedInfo.setCores(taskDto.getRequestedCores(), ontModelDataCenter);

90 requestedInfo.setCpuMaxAcceptableValue(taskDto.getRequestedCPUMax(), ontModelDataCenter);

91 requestedInfo.setCpuMinAcceptableValue(taskDto.getRequestedCPUMin(), ontModelDataCenter);

92

93 requestedInfo.setMemoryMaxAcceptableValue(taskDto.getRequestedMemoryMax(), ontModelDataCenter);

94 requestedInfo.setMemoryMinAcceptableValue(taskDto.getRequestedMemoryMin(), ontModelDataCenter);

95

96 requestedInfo.setStorageMaxAcceptableValue(taskDto.getRequestedStorageMax());

97 requestedInfo.setStorageMinAcceptableValue(taskDto.getRequestedStorageMin());

98

99 receivedInfo.setCores(0, ontModelDataCenter);

100 receivedInfo.setCpuReceived(0, ontModelDataCenter);

101 receivedInfo.setMemoryReceived(0, ontModelDataCenter);

102 receivedInfo.setStorageReceived(0, ontModelDataCenter);

103

104 task.setReceivedInfo(receivedInfo);

105 task.setRequestedInfo(requestedInfo);

106 task.setCpuWeight(0.33f);

107 task.setStorageWeight(0.33f);

108 task.setMemoryWeight(0.33f);

109

110 policy.setReferenced(task);

111 policy.setRespected(**false**);

112 policy.setPriority(1);

113 }

114 **break**;

115 **case** ACLMessage.INFORM\_IF:

116 System.out.println(**"http://www.owl-ontologies.com/Datacenter.owl#"** + message.getContent().split(**"-"**)[0]);

117 Task selectedTask = protegeFactory.getTask(**"http://www.owl-ontologies.com/Datacenter.owl#"** + message.getContent().split(**"-"**)[0]);

118 **if** (selectedTask.getAssociatedServer() != **null**) {

119 RemoveTaskFromServerCommand command = **new** RemoveTaskFromServerCommand(protegeFactory, selectedTask.getName(), selectedTask.getAssociatedServer().getName());

120 command.execute(jenaModel);

121 command.executeOnX3D(agent);

122 selectedTask.deleteInstance(jenaModel, swrlFactory);

123

124 agent.sendAllTasksToClient();

125 } **else** {

126 agent.sendRefuseMessage();

127 }

128 **break**;

129 **case** ACLMessage.SUBSCRIBE:

130 Boolean value = (Boolean) message.getContentObject();

131 agent.setContextDirty(value);

132 System.err.println(**"Context dirty received"**);

133 **break**;

134 *//http://www.owl-ontologies.com/Datacenter.owl#Task\_1*

135 }

136 } **catch** (Exception ex) {

137 ex.printStackTrace(System.err);

138 }

139 }

140 }

141

|  |
| --- |
| DefaultQoSPolicy.java |

1 **package** greenContextOntology.impl;

2

3 **import** com.hp.hpl.jena.ontology.OntModel;

4 **import** edu.stanford.smi.protege.model.FrameID;

5 **import** edu.stanford.smi.protegex.owl.model.OWLModel;

6 **import** edu.stanford.smi.protegex.owl.model.RDFProperty;

7 **import** greenContextOntology.QoSPolicy;

8 **import** greenContextOntology.ReceivedTaskInfo;

9 **import** greenContextOntology.RequestedTaskInfo;

10 **import** greenContextOntology.Task;

11

12 */\*\**

13  *\* Generated by Protege-OWL (http://protege.stanford.edu/plugins/owl).*

14  *\* Source OWL Class: http://www.owl-ontologies.com/Datacenter.owl#QoSPolicy*

15  *\**

16  *\** ***@version*** *generated on Sun Mar 07 13:11:11 EET 2010*

17  *\*/*

18 **public class** DefaultQoSPolicy **extends** DefaultPolicy

19 **implements** QoSPolicy {

20

21 **public** DefaultQoSPolicy(OWLModel owlModel, FrameID id) {

22 **super**(owlModel, id);

23 }

24

25

26 **public** DefaultQoSPolicy() {

27 }

28

29

30 **public boolean** getRespected(OntModel ontModel) {

31 Task task = **this**.getReferenced();

32 ReceivedTaskInfo receivedInfo = task.getReceivedInfo();

33 RequestedTaskInfo requestedInfo = task.getRequestedInfo();

34 **if** (receivedInfo.getCores() < requestedInfo.getCores()) {

35 **return false**;

36 }

37 **if** (receivedInfo.getCpuReceived() < requestedInfo.getCpuMinAcceptableValue()) {

38 **return false**;

39 }

40 **if** (receivedInfo.getMemoryReceived() < requestedInfo.getMemoryMinAcceptableValue()) {

41 **return false**;

42 }

43 **if** (receivedInfo.getStorageReceived() < requestedInfo.getStorageMinAcceptableValue()) {

44 **return false**;

45 }

46 **return true**;

47 }

48

49 *// Property http://www.owl-ontologies.com/Datacenter.owl#referenced*

50

51 **public** Task getReferenced() {

52 **return** (Task) getPropertyValueAs(getReferencedProperty(), Task.**class**);

53 }

54

55

56 **public** RDFProperty getReferencedProperty() {

57 **final** String uri = **"http://www.owl-ontologies.com/Datacenter.owl#referenced"**;

58 **final** String name = getOWLModel().getResourceNameForURI(uri);

59 **return** getOWLModel().getRDFProperty(name);

60 }

61

62

63 **public boolean** hasReferenced() {

64 **return** getPropertyValueCount(getReferencedProperty()) > 0;

65 }

66

67

68 **public void** setReferenced(Task newReferenced) {

69 setPropertyValue(getReferencedProperty(), newReferenced);

70 }

71 }

72

# Appendix B: C# Source Code

|  |
| --- |
| VMHandling.cs |

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Management;

using System.IO;

using System.Runtime.InteropServices;

using System.Windows.Forms;

using ServerMonitorUtils;

namespace ServerManagement

{

class VMHandling

{

[DllImport("PowrProf.dll")]

public static extern bool SetSuspendState(bool Hibernate, bool ForceCritical, bool DisableWakeEvent);

static ManagementObject GetLastVirtualSystemSnapshot(ManagementObject vm)

{

ManagementObjectCollection settings = vm.GetRelated(

"Msvm\_VirtualSystemsettingData",

"Msvm\_PreviousSettingData",

null,

null,

"SettingData",

"ManagedElement",

false,

null);

ManagementObject virtualSystemsetting = null;

foreach (ManagementObject setting in settings)

{

Console.WriteLine(setting.Path.Path);

Console.WriteLine(setting["ElementName"]);

virtualSystemsetting = setting;

}

return virtualSystemsetting;

}

static string GetVirtualSystemExportSettingDataInstance(ManagementScope scope, ManagementObject snapshotSettingData)

{

ManagementPath settingPath = new ManagementPath("Msvm\_VirtualSystemExportSettingData");

ManagementClass exportSettingDataClass = new ManagementClass(scope, settingPath, null);

ManagementObject exportSettingData = exportSettingDataClass.CreateInstance();

exportSettingData["CopySnapshotConfiguration"] = 2;

exportSettingData["CreateVmExportSubdirectory"] = true;

exportSettingData["SnapshotVirtualSystem"] = snapshotSettingData.Path.Path;

string settingData = exportSettingData.GetText(TextFormat.CimDtd20);

exportSettingData.Dispose();

exportSettingDataClass.Dispose();

return settingData;

}

public static void ExportVirtualSystemExSnapshots(string vmName, string exportDirectory)

{

ManagementScope scope = new ManagementScope(@"\root\virtualization", null);

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("ExportVirtualSystemEx");

ManagementObject vm = Utility.GetTargetComputer(vmName, scope);

CreateVirtualSystemSnapshot(vmName);

ManagementObject snapshotSettingData = GetLastVirtualSystemSnapshot(vm);

if (snapshotSettingData == null)

{

Console.WriteLine("The snapshot wasn't created");

return;

}

inParams["ComputerSystem"] = vm.Path.Path;

if (!Directory.Exists(exportDirectory))

{

Directory.CreateDirectory(exportDirectory);

}

inParams["ExportDirectory"] = exportDirectory;

inParams["ExportSettingData"] = GetVirtualSystemExportSettingDataInstance(scope, snapshotSettingData);

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("ExportVirtualSystemEx", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("VM '{0}' were exported successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Failed to export VM");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("VM '{0}' were exported successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Export virtual system failed with error:{0}", outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

vm.Dispose();

virtualSystemService.Dispose();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Config only\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static string GetConfigOnlyVirtualSystemExportSettingDataInstance(ManagementScope scope)

{

ManagementPath settingPath = new ManagementPath("Msvm\_VirtualSystemExportSettingData");

ManagementClass exportSettingDataClass = new ManagementClass(scope, settingPath, null);

ManagementObject exportSettingData = exportSettingDataClass.CreateInstance();

// Do not copy VHDs and AVHDs but copy the Snapshot configuration and Saved State information (Runtime information) if present

exportSettingData["CopySnapshotConfiguration"] = 0;

exportSettingData["CopyVmRuntimeInformation"] = true;

exportSettingData["CopyVmStorage"] = false;

exportSettingData["CreateVmExportSubdirectory"] = true;

string settingData = exportSettingData.GetText(TextFormat.CimDtd20);

exportSettingData.Dispose();

exportSettingDataClass.Dispose();

return settingData;

}

static void ExportVirtualSystemExConfigOnly(string vmName, string exportDirectory)

{

ManagementScope scope = new ManagementScope(@"root\virtualization", null);

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("ExportVirtualSystemEx");

ManagementObject vm = Utility.GetTargetComputer(vmName, scope);

inParams["ComputerSystem"] = vm.Path.Path;

if (!Directory.Exists(exportDirectory))

{

Directory.CreateDirectory(exportDirectory);

}

inParams["ExportDirectory"] = exportDirectory;

inParams["ExportSettingData"] = GetConfigOnlyVirtualSystemExportSettingDataInstance(scope);

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("ExportVirtualSystemEx", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("VM '{0}' were exported successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Failed to export VM");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("VM '{0}' were exported successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Export virtual system failed with error:{0}", outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

vm.Dispose();

virtualSystemService.Dispose();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Create Snapshot \*\*\*\*\*\*\*\*\*\*\*\*\*/

static void CreateVirtualSystemSnapshot(string vmName)

{

ManagementScope scope = new ManagementScope(@"root\virtualization", null);

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementObject vm = Utility.GetTargetComputer(vmName, scope);

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("CreateVirtualSystemSnapshot");

inParams["SourceSystem"] = vm.Path.Path;

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("CreateVirtualSystemSnapshot", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("Snapshot was created successfully.");

}

else

{

Console.WriteLine("Failed to create snapshot VM");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("Snapshot was created successfully.");

}

else

{

Console.WriteLine("Create virtual system snapshot failed with error {0}", outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

vm.Dispose();

virtualSystemService.Dispose();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public static ManagementObject CreateSwitch(ManagementScope scope, string name, string friendlyName, int learnableAddress)

{

ManagementObject switchService = Utility.GetServiceObject(scope, "Msvm\_VirtualSwitchManagementService");

ManagementObject createdSwitch = null;

ManagementBaseObject inParams = switchService.GetMethodParameters("CreateSwitch");

inParams["FriendlyName"] = friendlyName;

inParams["Name"] = name;

inParams["NumLearnableAddresses"] = learnableAddress;

inParams["ScopeofResidence"] = null;

ManagementBaseObject outParams = switchService.InvokeMethod("CreateSwitch", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("{0} was created successfully", inParams["Name"]);

createdSwitch = new ManagementObject(outParams["CreatedVirtualSwitch"].ToString());

}

else

{

Console.WriteLine("Failed to create {0} switch.", inParams["Name"]);

}

return createdSwitch;

}

public static void ModifyVirtualSystem(string vmName, string vmNewName)

{

ManagementScope scope = new ManagementScope(@"root\virtualization", null);

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("ModifyVirtualSystem");

ManagementObject vm = Utility.GetTargetComputer(vmName, scope);

inParams["ComputerSystem"] = vm.Path.Path;

ManagementObject settingData = null;

ManagementObjectCollection settingsData = vm.GetRelated("Msvm\_VirtualSystemSettingData",

"Msvm\_SettingsDefineState",

null,

null,

"SettingData",

"ManagedElement",

false,

null);

foreach (ManagementObject data in settingsData)

{

settingData = data;

}

settingData["ElementName"] = vmNewName;

inParams["SystemsettingData"] = settingData.GetText(TextFormat.CimDtd20);

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("ModifyVirtualSystem", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("VM '{0}' was renamed successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Failed to Modify VM");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("VM '{0}' was renamed successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Failed to modify VM. Error {0}", outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

vm.Dispose();

virtualSystemService.Dispose();

}

public static void ImportVirtualSystem(string importDirectory)

{

ManagementScope scope = new ManagementScope(@"root\virtualization", null);

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("ImportVirtualSystem");

inParams["GenerateNewID"] = true;

inParams["ImportDirectory"] = importDirectory;

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("ImportVirtualSystem", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("VM were imported successfully.");

}

else

{

Console.WriteLine("Failed to import VM");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("VM were imported successfully.");

}

else

{

Console.WriteLine("Import virtual system failed with error {0}", outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

virtualSystemService.Dispose();

}

public static ManagementBaseObject GetVirtualSystemImportSettingData(ManagementScope scope, string importDirectory,string vmName, string rootDirectoryToCopy)

{

string targetVhdResourcePath = rootDirectoryToCopy; //Directories specified should exist

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementBaseObject importSettingData = null;

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("GetVirtualSystemImportSettingData");

inParams["ImportDirectory"] = importDirectory;

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("GetVirtualSystemImportSettingData", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

importSettingData = (ManagementBaseObject)outParams["ImportSettingData"];

Console.WriteLine("Import Setting Data for the ImportDirectory '{0}' was retrieved successfully.", importDirectory);

}

else

{

Console.WriteLine("Failed to get the Import Setting Data");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

importSettingData = (ManagementBaseObject)outParams["ImportSettingData"];

Console.WriteLine("Import Setting Data for the ImportDirectory '{0}' was retrieved successfully.", importDirectory);

}

else

{

Console.WriteLine("Failed to get the Import Setting Data for the Import Directory :{0}", (UInt32)outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

virtualSystemService.Dispose();

importSettingData["GenerateNewId"] = true;

importSettingData["CreateCopy"] = true;

importSettingData["Name"] = vmName;

importSettingData["TargetVmDataRoot"] = rootDirectoryToCopy;

importSettingData["TargetSnapshotDataRoot"] = rootDirectoryToCopy;

importSettingData["TargetVhdDataRoot"] = rootDirectoryToCopy;

ManagementObject newSwitch = CreateSwitch(scope, Guid.NewGuid().ToString(), "Switch", 1024);

importSettingData["TargetNetworkConnections"] = new string[] { (newSwitch.GetPropertyValue("Name").ToString()) };

//Console.WriteLine("", ((string[])importSettingData["TargetResourcePaths"])[0]);

return importSettingData;

}

public static void ImportVirtualSystemEx(string importDirectory, string vmName , string importCopyDirectory)

{

try

{

if (Directory.Exists(importCopyDirectory))

{

Directory.Delete(importCopyDirectory, true);

}

}

catch(Exception ex)

{

Console.WriteLine(ex.ToString());

}

Directory.CreateDirectory(importCopyDirectory);

ManagementScope scope = new ManagementScope(@"root\virtualization", null);

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementBaseObject importSettingData = GetVirtualSystemImportSettingData(scope, importDirectory, vmName, importCopyDirectory);

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("ImportVirtualSystemEx");

inParams["ImportDirectory"] = importDirectory;

inParams["ImportSettingData"] = importSettingData.GetText(TextFormat.CimDtd20);

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("ImportVirtualSystemEx", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("VM were Imported successfully.");

}

else

{

Console.WriteLine("Failed to Imported VM");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("VM were Imported successfully.");

}

else

{

Console.WriteLine("Import virtual system failed with error:{0}", outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

virtualSystemService.Dispose();

}

/\*copy the content of a directory into another directory\*/

public static void CopyAll(DirectoryInfo source, DirectoryInfo target)

{

// Check if the target directory exists, if not, create it.

if (Directory.Exists(target.FullName) == false)

{

Directory.CreateDirectory(target.FullName);

}

FileInfo fileInfo;

// Copy each file into it’s new directory.

foreach (FileInfo fi in source.GetFiles())

{

Console.WriteLine(@"Copying {0}\{1}", target.FullName, fi.Name);

fileInfo=fi.CopyTo(Path.Combine(target.ToString(), fi.Name), true);

}

// Copy each subdirectory using recursion.

foreach (DirectoryInfo diSourceSubDir in source.GetDirectories())

{

DirectoryInfo nextTargetSubDir =

target.CreateSubdirectory(diSourceSubDir.Name);

CopyAll(diSourceSubDir, nextTargetSubDir);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Modify virtual machine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public static void RequestStateChange(string vmName, string action)

{

ManagementScope scope = new ManagementScope(@"root\virtualization", null);

ManagementObject vm = Utility.GetTargetComputer(vmName, scope);

ManagementBaseObject inParams = vm.GetMethodParameters("RequestStateChange");

const int Enabled = 2;

const int Disabled = 3;

if (action.ToLower() == "start")

{

inParams["RequestedState"] = Enabled;

}

else if (action.ToLower() == "stop")

{

inParams["RequestedState"] = Disabled;

}

else

{

throw new Exception("Wrong action is specified");

}

ManagementBaseObject outParams = vm.InvokeMethod("RequestStateChange", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("{0} state was changed successfully.", vmName);

}

else

{

Console.WriteLine("Failed to change virtual system state");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("{0} state was changed successfully.", vmName);

}

else

{

Console.WriteLine("Change virtual system state failed with error {0}", outParams["ReturnValue"]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Delete virtual machine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public static void DestroyVirtualSystem(string vmName)

{

ManagementScope scope = new ManagementScope(@"root\virtualization", null);

ManagementObject virtualSystemService = Utility.GetServiceObject(scope, "Msvm\_VirtualSystemManagementService");

ManagementBaseObject inParams = virtualSystemService.GetMethodParameters("DestroyVirtualSystem");

ManagementObject vm = Utility.GetTargetComputer(vmName, scope);

inParams["ComputerSystem"] = vm.Path.Path;

ManagementBaseObject outParams = virtualSystemService.InvokeMethod("DestroyVirtualSystem", inParams, null);

if ((UInt32)outParams["ReturnValue"] == ReturnCode.Started)

{

if (Utility.JobCompleted(outParams, scope))

{

Console.WriteLine("VM '{0}' were deleted successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Failed to delete VM");

}

}

else if ((UInt32)outParams["ReturnValue"] == ReturnCode.Completed)

{

Console.WriteLine("VM '{0}' were deleted successfully.", vm["ElementName"]);

}

else

{

Console.WriteLine("Destroy virtual system failed with error {0}", outParams["ReturnValue"]);

}

inParams.Dispose();

outParams.Dispose();

vm.Dispose();

virtualSystemService.Dispose();

}

}

}

|  |
| --- |
| Service1.cs |

using System;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Linq;

using System.Web;

using System.Web.Services;

using System.Web.Services.Protocols;

using System.Xml.Linq;

using ServerManagement;

using System.Net;

using System.Net.Sockets;

using System.Windows.Forms;

using System.Diagnostics;

using Microsoft.Win32;

using System.IO;

namespace ASPWebService

{

/// <summary>

/// Summary description for Service1

/// </summary>

[WebService(Namespace = "http://www.SelfOptimizingDatacenter.edu/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

[ToolboxItem(false)]

// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.

// [System.Web.Script.Services.ScriptService]

public class Service1 : System.Web.Services.WebService

{

private static bool operationCompleted = false;

[WebMethod]

public ServerInfo GetServerInfo()

{

return ServerMonitor.CollectServerInfo();

}

[WebMethod]

public void MoveDestinationActions(String path1, String path2, String vmName)

{

operationCompleted = false;

DirectoryInfo dfrom = new DirectoryInfo(path1 + "\\" + vmName);

DirectoryInfo dto = new DirectoryInfo(path2 + "\\" + vmName);

VMHandling.CopyAll(dfrom, dto);

VMHandling.ImportVirtualSystem(path2 + "\\" + vmName);

VMHandling.ModifyVirtualSystem(vmName, vmName );

VMHandling.RequestStateChange(vmName , "start");

operationCompleted = true;

}

[WebMethod]

public bool MoveSourceActions(String path, String vmName)

{

operationCompleted = false;

VMHandling.RequestStateChange(vmName, "stop");

VMHandling.ExportVirtualSystemExSnapshots(vmName, path); // export existing snapshot of a virtual machine

VMHandling.DestroyVirtualSystem(vmName);

operationCompleted = true;

return true;

}

[WebMethod]

public void DeployVirtualMachine(String from, String to, String vmName, String vmCopyName)

{

//just to make sure it is running

Process.Start("C:\\StartHVBoot.bat");

operationCompleted = false;

DirectoryInfo dfrom = new DirectoryInfo(from + "\\"+vmName);

DirectoryInfo dto = new DirectoryInfo(to + "\\" + vmCopyName);

VMHandling.CopyAll(dfrom, dto);

VMHandling.ImportVirtualSystem(to + "\\" + vmCopyName);

VMHandling.ModifyVirtualSystem(vmName, vmCopyName);

operationCompleted = true;

}

[WebMethod]

public void StartVirtualMachine(String vmName)

{

//just to make sure it is running

Process.Start("C:\\StartHVBoot.bat");

operationCompleted = false;

VMHandling.RequestStateChange(vmName, "start");

operationCompleted = true;

}

[WebMethod]

public void StopVirtualMachine(String vmName)

{

operationCompleted = false;

VMHandling.RequestStateChange(vmName, "stop");

operationCompleted = true;

}

[WebMethod]

public void DeleteVirtualMachine(String vmName)

{

VMHandling.DestroyVirtualSystem(vmName);

}

//must be "02 17 31 65 C3 5F"

[WebMethod]

public void WakeUpServer(String mac, String ipAddress, int port)

{

operationCompleted = false;

string[] values = mac.Split('-');

UdpClient client = new UdpClient();

client.Connect(IPAddress.Parse(ipAddress), port);

//

// WOL packet contains a 6-bytes trailer and 16 times a 6-bytes sequence containing the MAC address.

//

byte[] packet = new byte[17 \* 6];

//

// Trailer of 6 times 0xFF.

//

for (int i = 0; i < 6; i++)

packet[i] = 0xFF;

//

// Body of magic packet contains 16 times the MAC address.

//

for (int i = 1; i <= 16; i++)

for (int j = 0; j < 6; j++)

packet[i \* 6 + j] = (byte)Convert.ToInt32(values[j], 16);

//

// Submit WOL packet.

//

client.Send(

packet, packet.Length);

}

[WebMethod]

public void SendServerToSleep()

{

RegistryKey key = Registry.LocalMachine.CreateSubKey("SYSTEM\\CurrentControlSet\\Services\\hvboot");

key.SetValue("Start", 3);

RegistryKey runOnceKey = Registry.LocalMachine.CreateSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce");

runOnceKey.SetValue("SendToSleepAutomatically", "C:\\SendToSleep.exe");

Process.Start("shutdown.exe", " -r -f -t 0");

//System.Windows.Forms.Application.SetSuspendState(PowerState.Suspend, false, false);

}

[WebMethod]

public bool IsOperationCompleted()

{

return operationCompleted;

}

}

}