Table of Contents

[1 Introduction 3](#_Toc264006965)

[1.1 Motivation and Objectives 3](#_Toc264006966)

[1.2 Contributions 3](#_Toc264006967)

[1.3 Publications 4](#_Toc264006968)

[1.4 Background 4](#_Toc264006969)

[1.4.1 Autonomic Systems 5](#_Toc264006970)

[1.4.2 Intelligent Agents and Pervasive Computing 5](#_Toc264006971)

[1.4.3 Artificial Intelligence Learning, Knowledge Representation and Reasoning 6](#_Toc264006972)

[1.4.4 An introduction to Green Computing, Virtualization and Server Consolidation 7](#_Toc264006973)

[1.4.5 An overview of existing negotiation and bargaining solutions 8](#_Toc264006974)

[1.4.6 An overview of existing self-adapting systems 10](#_Toc264006975)

[2 Problem Statement and Goals 12](#_Toc264006976)

[2.1 Problem Statement 12](#_Toc264006977)

[2.2 Problem Goals 12](#_Toc264006978)

[3 Analysis and Design 13](#_Toc264006979)

[3.1 The context model 14](#_Toc264006980)

[3.1.1 Context Ontology Representation 14](#_Toc264006981)

[3.1.2 Policies 15](#_Toc264006982)

[3.1.3 Context Entropy 15](#_Toc264006983)

[3.1.4 State reward 16](#_Toc264006984)

[3.1.5 Inter-Independent Resources Group 16](#_Toc264006985)

[3.1.6 RAP instantiation for the self-healing system 18](#_Toc264006986)

[3.1.7 RAP instantiation for the self-adapting datacenter 19](#_Toc264006993)

[3.1.8 Load Distribution Strategy 19](#_Toc264006994)

[3.2 Reinforcement Learning Algorithm 20](#_Toc264006995)

[3.3 QoS-Energy Consumption Negotiation 22](#_Toc264006996)

[3.4 The Conceptual Architecture 23](#_Toc264006997)

[4 Implementation 24](#_Toc264006998)

[4.1 Architecture 24](#_Toc264006999)

[4.2 System Input 25](#_Toc264007000)

[4.3 System Output 27](#_Toc264007001)

[4.4 Architecture Implementation 27](#_Toc264007002)

[4.5 Ontology Representation Implementation 28](#_Toc264007003)

[4.5.1 Environment Ontology Implementation 28](#_Toc264007004)

[4.5.2 Datacenter Ontology Representation 28](#_Toc264007005)

[4.6 Agents Implementation 30](#_Toc264007006)

[4.6.1 Context Interpreting Agent (CIA) 30](#_Toc264007007)

[4.6.2 Reinforcement Learning Agent (RLA) 31](#_Toc264007008)

[4.6.3 X3D Agent Implementation 32](#_Toc264007009)

[4.6.4 GUI Agent Implementation 33](#_Toc264007010)

[4.7 Utility components 36](#_Toc264007011)

[4.7.1 Fuzzy Logic Negotiator 36](#_Toc264007012)

[4.7.2 Server Management Proxy 37](#_Toc264007013)

[5 Testing and results 38](#_Toc264007014)

[5.1 Self-Healing Environment Test 38](#_Toc264007015)

[5.1.1 Running Time Test 40](#_Toc264007016)

[5.1.2 System Output Test 41](#_Toc264007017)

[5.2 Self-Adapting Datacenter Test 42](#_Toc264007018)

[6 Conclusions 43](#_Toc264007019)

[7 43](#_Toc264007020)

[8 References 43](#_Toc264007021)

# Introduction

## Motivation and Objectives

Worldwide data centers electricity consumption accounts for almost 2% of the world production and is expected to overcome the 40% of Total Cost of Ownership of worldwide IT by 2012 [ 1 ]. The US Environmental Protection Agency estimated that in 2006 the servers and datacenters power consumption accounted for 61 billion kWh (**!! De introdus in acronime**) , about 1.5 % of total U.S electricity consumption and for a cost of $4.5 billion [ 2 ].

The same organization points out that the average data center is only 30% efficient, with 70% of the electricity lost due to inefficiencies of power and heat dissipation, along with powering cooling equipment [ 3 ]. The environmental impact of datacenter expansion is of great importance, every server using 7,000 kWh of electricity and indirectly generating four tons of carbon dioxide emissions per year [ 4 ].

This has lead to the need of finding environmental friendly methods for managing datacenters while maintaining performance. This new research trend has been called Green IT or Green Computing. The philosophy of Green IT is designing and using computer resources in an environmentally friendly way. This work has as aim reducing the overall number of servers used worldwide and further minimizing the number of powered on servers by applying server consolidation.

The main goal of this research is to develop an agent-based framework for dynamically adaptive datacenters which achieves server consolidation by combining server and task virtualization with a self-adapting datacenter controller. The framework should also provide a datacenter self-healing environment control mechanism. A secondary goal is finding new ways and methods of tradeoff between QoS and power consumption for further increasing server consolidation.

## Contributions

This project seeks to provide the necessary tool support for datacenter virtualization, consolidation, environment management and to develop new methods of negotiating between QoS and Power Consumption. There are both theoretical and practical contributions made by this project.

**Theoretical contributions**

* Develop a reinforcement-learning policy-based algorithm for autonomic self-healing environments. Adapt the previously mentioned algorithm for data center task management.
* Develop methods for negotiating between QoS (!!!!!!!!ACRONIME) requirements and Server Optimum Load Values (implying power consumption).

**Practical contributions**

* + - Create a datacenter management framework using policy-based mobile agents.
    - Develop tools for monitoring datacenter resources usage and environment sensors.
    - Find and evaluate appropriate technologies for implementing a self-adapting datacenter based on the framework mentioned above.

## Publications

The research conducted for this project has generated the following publications. The report chapters which include material from these publications are listed for each publication.

* **A Reinforcement Learning based Self-healing Algorithm for Managing Context Adaptation** [ 5 ] presents a reinforcement learning based algorithm use for finding the optimum sequence of actions which enforce some user-specified policies. A proof of concept implementation is also presented, which uses an X3D (**!!! De pus in lista de achronimes)** representation of a smart environment in which a user click on an object to change its state. The algorithm detects if a user-specified policy is broken, searches for the best (having the highest reward) sequence of actions which “repairs” the context and executes them. (**!!!!** Material from this paper can be found in **chapter2onpage7,Chapter4onpage31andChapter5onpage47**
* **An Autonomic Algorithm for Energy Efficiency in Service Centers** [ 6 ] (!!!!**De modificat daca nu I acceptat)** is an adaptation of the algorithm presented in the previous paper to handle datacenter task management. The old algorithm is still used for monitoring the datacenter environment. Both algorithms are used in a proof of concept implementation which uses a simulated datacenter in which tasks are dynamically added and removed, triggering consolidation actions.
  + - **!!!! THESIS OVERVIEW?**

## Background

This chapter provides an overview of the background theory and results in the area of Artificial Intelligence, Self-Adapting Systems and Green Computing specific to the requirements of this project.

The following topics are discussed:

* An introduction to Autonomic Systems -**Self-\* systems – de completat – din Survery de Context, si The Authonomic Computing Paradigm**
* An introduction to Intelligent Agents and Pervasive Computing
* An overview of Artificial Intelligence Learning, Knowledge Representation and Reasoning
* An introduction to Green Computing, Virtualization and Server Consolidation
* An overview of existing negotiation and bargaining solutions
* An overview of existing self-adapting systems

### Autonomic Systems

**DACA MA HOTARASC SA PUN DESPRE AUTONOMIC STUFF**

### Intelligent Agents and Pervasive Computing

Pervasive computing, also named as **everywhere computing** or **ubiquitous computing** [ 7] is a computing paradigm in which information processing has been integrated in everyday life by means of small networked processing devices. These devices link communications and computing infrastructure to everyday life settings and commonplace tasks.

Agents are defined as anything perceives its environment through sensors and performs actions on that environment through actuators [ 8 ].

#### Agent Environment

By their nature, the environments are roughly classified by [ 8 ] in:

**Fully observable and partially observable environments:** As the name suggests, in fully observable environments the agent has all the needed information, while in partially observable not everything about the context is known.

**Deterministic and Stochastic environments**: In a deterministic environment the next state is determined entirely based on the current state and the action executed by the agent.

**Episodic and Sequential environments**: in an episodic environment there are independent episodes, while in sequential ones the next world state is always influenced by the previous one.

**Static and Dynamic environments**: In a static environment the world does not change while the agent thinks what to do.

**Discrete and Continuous environments**: Discrete environments have a finite number of states.

**Single-Agent and Multiple-Agent environments**: In multiple-agent environments agents compete for a set of resources.

#### Agent Types

Based on the complexity of the agent’s reasoning process the agents are classified in 5 categories [ 8 ]:

**Simple reflex agents:** Act only based on the current input data, ignoring the context history. The agent’s actions are determined by simple rules. Such an agent can succeed only in fully observable environments.

**Model-based agents:** This type of agents maintains an internal model of the world in order to be able to reason over partially observable environments. The model holds previously observed information that now is not observable anymore and reacts just as the simple reflex agent to input data.

**Goal-based agents:** Because in complex situations just reacting will not lead to the desired outcome this type of agent has appeared. These agents know some **goal states** and will try to reach those states.

**Utility–based agents:** Knowing about just the goal does not provide a measure of degree of how expensive is a set of actions the concept of utility is introduced. A **utility function** is used to map each an environment state to a desirability value, thus the agent being capable of comparing two courses of action and choose the one with the highest utility.

**Learning agents:** Are the most complex and can operate in unknown environments. These agents learn about the surrounding world as they go and continuously adapt themselves.

### Artificial Intelligence Learning, Knowledge Representation and Reasoning

#### Knowledge Representation

Any intelligent agent, even reflex agents use knowledge about the environment in their reasoning process. The process of knowledge engineering is a daunting one due to the complexity of the surrounding world.

**Ontologies** are the state of the art mechanism for knowledge representation. Ontology can be defined as a set of concepts from a particular domain together with the relationships between those concepts. These concepts are grouped in categories by the use of inheritance, thus providing a flexible and extensible means of representing the surrounding world.

#### Learning

The main idea behind learning is to use the gathered world data for improving the decision process, allowing an intelligent agent to improve its behavior trough study of its own experience. The goal of machine learning is for the intelligent system to be able to recognize complex patterns and make intelligent decisions based on them. For achieving this goal, machine learning has to borrow methods and concepts from other fields such as statistics, probability theory, data mining, pattern recognition and others.

Based on the type of feedback available, machine learning is classified in three categories : supervised, unsupervised and reinforcement learning.

**Supervised learning** is based on having the correct output for each input state. Mostly used in completely observable environments, in this type of learning the agent knows all possible outcomes from all possible actions and such it always knows what the best actions are.

**Unsupervised learning** is the opposite of supervised learning as it involves no knowledge of the system’s output for a given input. An unsupervised agent cannot learn what to do, because it has no information about what is desirable and what is not.

**Reinforcement learning** is a combination of the previous presented methods. The agent is not fully supervised nor is it left clueless. Instead, it receives a valuation function called reinforcement which indicates if a behavior is acceptable or not. The function value is the reinforcement, a negative value representing a penalty and a positive value an encouragement. Most important in this type of learning is the representation of the learned data and the accuracy with which the reinforcement function classifies the situation the agent encounters.

#### Reasoning

Reasoning in computer science can be regarded as the process of finding in the input data support for some concept. The most developed research area in this direction is in automated theorem proving algorithms, which are used in finding support for theorems based on the input data. Other research area as reasoning under uncertainty is of major importance in designing intelligent agent systems because the real world is not discrete and conclusions need to be drawn without knowing every outcome of every action.

### An introduction to Green Computing, Virtualization and Server Consolidation

Green Computing or Green IT is a computing paradigm that refers to environmental sustainable IT. All hardware and software components must use as little energy as possible and must have as small as possible impact on the environment. Another reason why this trend is important is because implementing its concepts reduces the total cost of ownership by reducing the energy cost.

#### Virtualization and Server Consolidation

A major problem in today’s datacenters is under usage of resources [ 3 ][ 4 ]. Due to the lack of dynamic datacenter management based on the current or incoming load each datacenter must have all servers online in the event that the traffic increases .Also, being prepared for the largest traffic possible leads to the need of having a large number of servers. These two facts combined generate a problem called **server sprawl**, a situation in which multiple, under-utilized servers take up more space and consume more resources than can be justified by their workload.

**Server Consolidation** has as goal solving this problem by finding means of reducing the number of idle servers. There are several methods for server consolidation, some based on more powerful servers, some on virtualization.

**Virtualization** is a technique for running several operating systems on a single system. By employing virtualization many servers can be transferred to dedicated virtual machines and run on a more powerful system (Fig 1), thus increasing server consolidation. In virtualized environments, the virtual machines run on a “virtual machine OS” called a **hypervisor** which manages the resource allocation and distribution among the virtual machines. A major advantage of this approach is that a virtual machine’s resources can be modified depending on the load. Also, with modern hypervisors, virtual machines can be migrated from one server to another without any visible downtime to increase server utilization or in the event of a hardware failure.

Figure 1: Server Consolidation trough Virtualization

#### Datacenter Load Distribution

**Load skewing** is a datacenter load distribution technique which involves several tasks. The entire datacenter load is fitted onto as few servers as possible. One or more servers are kept as **tails** in order to accommodate future workload. The rest of the servers are sent to hibernation.

**Throttling** is the mechanism of reducing the performance of underused hardware components and thus reducing the energy consumption. This mechanism is very useful in datacenters where the workload is evenly distributed among servers.

### An overview of existing negotiation and bargaining solutions

A further improvement to server consolidation is the use of negotiation techniques to find a tradeoff between power consumption and QoS. Although QoS requirements are of most importance, in some case a tradeoff is needed due to hardware failure or even unjustified power consumption. For example if a decrease in 5% of CPU requirements for the entire datacenter would allow, after virtual machine rearrangement, for one server to be turned off or send to low power state, then a negotiation technique can be used to find the best value to decrease the CPU for each virtual machine.

The research area of distributed negotiation or multi-agent negotiation has a lot of work associated to it. From this work is clear that negotiation is an important issue in almost any distributed or intelligent system, being present from web services negotiation to grid resource allocation and with the help of this paper, in datacenter consolidation efforts.

Web service discovery and invocation benefit from negotiation as described in [ 9 ], where a tradeoff between QoS and Cost of Service is achieved trough exchange of less desirable tokens with more desirable ones between parties. As presented in [ 9 ], very important in web service discovery is the process of finding the most appropriate Web Service providers for a specific Web Service requestor. For being able to conduct real world business by automated web service composition, a negotiation mechanism must exist and it must ensure an optimal “deal” for both sides. This approach introduces *logrolling*, a negotiation technique used mostly in politics, where one person can trade his vote in the exchange of a vote received for his law proposition. The negotiation framework is based on tokens, split in two categories: Quality of Service (QoS) tokens and Cost of Service (CoS) tokens. Logrolling is combined with a token-based negotiation framework augmented with token weights representing each token’s importance. Using this solution, the provider and supplier exchange tokens until both sides reach a consensus, a state in which the utility functions of both parties are the same.

Another approach to multiple-party negotiation is presented in [ 10 ] under the form of a logic programming framework. This approach, instead of negotiating the existing proposal focuses on creating a counter proposal, ignoring the process of deciding on the utility of a proposal. The system is centered on a knowledge-base which contains all the necessary information about creating a counter-proposal. The proposal is generated using Abductive Logic Programming [ 11 ], capable of dealing with the unknown goals of the other negotiating party. In case the counter-proposal is rejected, another proposal is generated from the previous one by relaxing the knowledge-base search criteria. This automated negotiation approach solution can be built on top of existing answer set solvers, thus providing a clear path to concrete results.

The research work in the field of multi-issue negotiation can be split into three categories after their understanding of the best negotiation result: minimum loss, maximum gain, and the more general utility function maximization. An approach that fits in the first category is [ 12 ] which describes negotiation as searching for envy-free states in multi-agent environments. The negotiation process is viewed as a resource allocation one for better understanding of the presented concepts. Given that each agent involved in the resource allocation process has a valuation function to indicate its resource preference, this work focuses on proving that in an envy-free state the resources are efficiently distributed. It proves that envy-free states can be reached if they exist and that resource allocation efficiency and envy-freenes are compatible.

[ 13 ] fits in the second category, searching for joint gains as negotiation result in multiple independent issue negotiation. This approach focuses on “creating value” instead of minimizing loss. For coping with the tendency of agents to hide their intentions in order to obtain as much profit as possible from a trade, an impartial mediator is introduced. Agents disclose secret information to the mediator, which based on that information, tries to achieve a fair deal. Another key element of this solution is that it computes a Pareto-optimal set of outcomes (outcome in which no improvement can be made for a party without worsening of the outcome for the other party). The Pareto-optimal set is further inspected for finding fair situations in which the inter-agent resource distribution is appropriate. Other than providing a generic multi-issue multi-agent negotiation framework, [ 13 ] also presents utility maximization methods and compares them with respect to their manipulation susceptibility for use in environments where agents tend to misrepresent their utility.

An improvement over existing negotiation techniques is brought by [ 14 ] which describes a involving multiple interdependent issues, as encountered in many real-world scenarios. A Distributed Mediator Protocol is defined for finding Pareto-optimal agreement points. In order to have an efficient solution, a genetic algorithm is used to find multiple Pareto-optimal agreement points in the nonlinear negotiation space. The genetic algorithm is compared with two other search methods: simulated annealing and hill climbing. Another algorithm called Direct Search is defined which maximizes Nash products [ 49 ] without finding Pareto-optimal agreements. The Direct Search algorithm is compared against the Distributed Mediator Protocol and an approximated fairness concept is introduced for maximizing Nash products.

(!! DACA SE POATE SA SCOT PAPERU ALA CU FUZZY SA BAG PE AICI K AM NEVOIE DE EL SA DEMONSTREZ K I NOU CE AM FACUT)

### An overview of existing self-adapting systems

The work in the area of self-adapting systems is usually centered on the use of ontologies for representing context information and policies for representing goals. This leads to the need of having a reliable mechanism for gathering data from the surrounding environment and representing it in a manner that supports reasoning. Such a mechanism is presented in [ 15 ] under the form of an event-driven publish/subscribe architecture for data gathering, processing and event creation. Events are created based on the input data and are used by subscribers to monitor different areas or activities within the surrounding context. The events are fed to the event processing system through a series of event streams. Adding semantical information to an event, each event is assign to a context. This approach also defines a mechanism for fusing similar events from the same context and combining the information given by the last event with the old information held in the knowledge base. Aside from event processing, an alert mechanism is presented, used to inform about specific situations detected based on generated events. The mechanism is applied to real world monitoring situations, like video monitoring of a room. The major contribution brought by this work is the event contextualization mechanism used to map events to contexts, thus providing a semantic event space which can be further extended and applied to various domains.

Another approach to context information gathering is presented in [ 16 ]. This work focuses on the self-configuring aspect of context-aware systems in the area of pervasive computing [ 7 ]. It provides a dynamically reconfigurable fault tolerant context management system. The system is based on a context model containing both information about the required context information and context information metadata. The use of sensor description standards is advocated, as they support opportunistic discovery and integration of sensors and thus adding flexibility to the overall system. By relying on the IEEE 1451 smart sensor interface and SensorML [ 18 ] sensor description framework , this approach allows smart sensors to advertise and describe themselves to higher level management systems. The proposed approach is testes for a rescue crew scenario having a central management unit and several sensors mounted on each person. Testing results demonstrated the adaptability of the system from using sensor description standards.

One important research direction in context-aware system is in Multi-Agent Systems. Such a system is presented in [ 19 ] for management of power and performance in datacenters. The purpose of this paper is to demonstrate practically that agents can be used for implementing a coherent automated datacenter management system. The system is centered on three agents: Performance Agent, Power Agent and Coordination Agent. The Performance Agent is responsible for load distribution among servers. The Power Agent is responsible for power consumption monitoring and setting power caps. Also, this agent uses a drastic power consumption reduction mechanism by turning off and on servers depending on the datacenter workload. Finally, the Coordination Agent handles the communication between the other two agents and uses predefined utility functions and policies in sending control signals to the system. The presented system was tested on a datacenter hosting IBM blade servers running Linux. Although the experimental results are promising, the technology is not there to help in implementing such a radical power management. A server running Linux takes 5-10 minutes to start after being turned off, an amount of time which makes it difficult to implement such an approach in real-world datacenters.

Under the pressure to make computing eco friendly an important research branch in self-adapting systems is creating energy-aware systems that are capable of reducing power consumption and maximize resource usage. One such approach is [ 20 ] , which presents a dynamic load management system for virtualized datacenters. Employing virtualization, this solution allows a single server to be shared by multiple services, thus improving server consolidation and resource utilization. Also, virtualization allows for on demand task resource allocation based on the datacenter workload. For maximizing the energy consumption, this solution, as the one presented before exploits the idea of turning the servers on and off depending on the datacenter load. Although turning off servers has the visible advantage of lower power consumption, the process of waking up a server implies a large amount of time, as discovered and described in [ 19 ]. In order to minimize the impact of the server wake-up time, a limited look ahead control mechanism is introduced. Also, the server switching costs are considered. From the experimental results, using server virtualization together with a lookahead needed for anticipating datacenter state results in an average of 26% power reduction while maintaining QoS requirements.

Very important in self-adapting systems is the system’s ability to learn the proper actions that solve some problematic situation. One school of thought is focused on applying reinforcement learning algorithms to smart environments. Reinforcement Learning advocates view this type of learning as a more “human” way of searching for solutions. Also, it seems to be more natural to think of goals as states with high rewards. Such an approach is presented in [ 22 ] which demonstrates that reinforcement learning can be used successfully in autonomic systems. The reward is computed based on the multi-attribute system state and represents how desirable is for the system to be in that particular state. The selected learning algorithm is State-Action-Reward-State-Action (SARSA) and it is compared with a Goal-Action Attribute Model technique [ 23 ] and. The solution is tested using a series of simulated models and the result is that reinforcement learning is feasible for the management of autonomic systems, but future research is needed because this type of approach does not provide a high performance solution.

# Problem Statement and Goals

## Problem Statement

The purpose of this book is to present a self-adapting framework based on mobile intelligent agents which uses ontologies for context representation and policies for goal description. This solution must provide a general extensible platform for building context-aware self-adapting systems applicable to any field, not just energy-aware datacenters. The framework should provide a uniform mechanism of data gathering, context representation and policy enforcement. The policy enforcement engine should detect when the context is in a state which violates at least one policy. It should use a reinforcement learning algorithm to search for the best course of action which brings the context back in an accepted state. The expected reward of being in the resulting sate must be the highest between reachable acceptable states. The reward mechanism must be flexible and should capture accurately the differences in severity between different unacceptable states based not only on the number of broken policies, but also on the distance to the closest acceptable state and the number of sub rules in the policy which are violated. For completing the environment management, the proposed framework must provide a flexible and extensible mechanism for actuators management.

## Problem Goals

1. Create a suitable context representation
2. Build a context management suite
3. Apply the context management solution to a self-healing environment
4. Extends and adapt the context management solution for datacenter management

In order to achieve this goals there are some important sub-goals to be met.

1. **Create a suitable context representation**
   1. Study the RAP[ 24 ]context model
   2. Discover the context object types and their relationships
   3. Build a context ontology representation. // using Protégé [19].
2. **Build a context management suite**
   1. Create an information gathering mechanism:

* Create an information gathering module for the System Under Test(SUT) endpoint
* Create a context information consumption mechanism for querying the SUT information gathering modules.
  1. Create an extensible actuator management mechanism:
* Insert the actuators in the context ontology representation
* Design a self-adapting algorithm which can handle in real time actuator list modifications.
  1. Build a logging mechanism for better context monitoring and debugging
* Create a PDF log writer and a log display window
  1. Build a distributed context management framework based on mobile agents
* Study intelligent mobile agents
* Define agents and their behaviors
* Choose an inter-agent communication mechanism
  1. Build a 3D context representation:
* Study the available 3D technologies
* Find a synchronization method between the 3D representation and the real context
* Make the representation interactive: interactions with the 3D context must generate actuator actions.
  1. Provide additional visual system information:
* Provide a real-time plot representing the algorithm running time
* Provide a visual representation of the context policies, sensors and their values

1. **Apply the context management solution to a smart environment**
   1. Create web services for the smart environment sensors
   2. Define policies, actuators and associated actions
   3. Test the context management solution under context patterns
   4. Extensively test the system using the interactive 3D representation
2. **Extends and adapt the context management solution for server cluster management**
   1. Create web services for interacting with the server operating system
   2. Adapt the context representation

* Split the policies into two categories : Energy policies and QoS policies
* Augment the context representation with information particular to datacenters
  1. Find techniques for improving the energy efficiency in datacenters
* Find a mechanism for dynamic task migration
* Find a mechanism for improving server consolidation
  1. Study datacenter management technologies
  2. Provide system information visually:
* Adapt the existing 3D representation
* Implement server resource monitors
* Create a visual representation of the tasks(both deployed and undeployed) for better task tracking
  1. Test the system on a real world server cluster
  2. Develop a workload generating tool for extensive system test.

# Analysis and Design

De inserat descrierea capitolului

CRED K ASTA LA PARTEA DE IMPLEMENTARE.

The “Environment Self-Healing Module” was developed by me and the author of [ 25 ] for the CONSENS research project[ 27 ] and as such it is a joint development effort, also described in [ 25 ].

The system’s core algorithm and associated concepts where developed by me and Copil Georgiana, author of [ 25 ], for the **CONSENS** research project [ 27 ] . The results of the research in the context of this project are described in [ 5 ].Due to these facts, the **reinforcement based self-healing algorithm** and associated concepts descriptions will appear both in this book and in [ 25 ]. The entire **4.1.The Context Model** and **4.2.The Reinforcement Learning Algorithm** subchapter is joint development effort, elements contained in it being also described in [ 25 ].

DE PROOF READ DACA OK JUSTIFICAREA

## The context model

The context model used by the self-adapting solution presented in this book is based on the RAP (Resources, Actors, Policies) model presented in [ 24 ]. This model was chosen because is general and powerful enough to be used in any self-adapting system.

### Context Ontology Representation

The core ontology representation only extends the Resources element of the RAP model with Sensor and Actuator concepts. This distinction between resources is needed as the system needs to know what element returns context information and what element can influence the context.



Figure 2: RAP Ontology Representation

Because the datacenter context needs more specific information, another ontology representation is defined based on the RAP model and used together with the previous basic representation. The previous representation is used for environmental representation and the next ontology is used to represent the datacenter physical and software components. Also, due to the fixed number of actions possible in a datacenter, the generality of the representation is reduced by creating a concrete class for each possible action. The reduction in generality is motivated by an increase in efficiency by eliminating the search for all possible actions at each step in the action selection algorithm.



Figure 3: Datacenter Ontology Representation

### Policies

The policies consist of a set of acceptable sensors values. In order to have a fine-grained view over any context situation and be able to differentiate between the severity of two different context situations, weights are used. Each policy has a weight associated to it which represents the importance of the policy in the overall context. Furthermore, each resource associated to a policy has a weight representing the importance of the sensor in that particular policy.

DE BAGAT FORMALIZARE A POLITICII

### Context Entropy

For understanding the concept of context state entropy the state must be defined. A context state is a snapshot of the execution environment taken at a specific moment in time.

For evaluating the degree of context state deviation from the acceptable value the concept of context state entropy is used [ 5 ]. The context state entropy is used to represent the desirability of the system to be in a particular state. Based on the definition of the policy, the entropy is defined as the product between the policy weight and the sum of products between the associated resources weight and their distance from the acceptable value:

Equation 1: Entropy Formula

|  |
| --- |
|  |
| , where:  - : entropy  - **:** the weight of i-th policy  - **:** the importance of resource j in policy i  - **:** is a value marking the degree to which resource j respects policy i. |

### State reward

For the self-healing environment management algorithm the reward is represented by the inverse of the entropy. The states with the highest reward are the states with the smallest entropy.

Equation 2: Reward function

|  |
| --- |
|  |
| , where:  - R**:** the state reward  - **:** the state entropy |

For the self-adapting datacenter management algorithm the reward is computed using a more fine-grained formula described in [ 25 ].

### Inter-Independent Resources Group

The time required to find the best sequence of actions that brings the system in an accepted state depends on the number of policies and on the number of resources attached to a policy. For improving the performance of the system the concept of Inter-Independent Resources Group (IIRG) is introduced. In order to understand the IIRG concept the dependency relation must be defined first. A resource is dependent on another resource when a change in the value of one of the resources triggers a change in the value of the other resource. Furthermore, the dependency relation is of two types: direct dependency and indirect dependency. Direct dependency between two resources and occurs when a change in the value of triggers a change in ’s value. Indirect dependency between two resources and occurs when a change in ’s value triggers a chain of changes in the values of several resources () which eventually affects . The dependency relation is represented under the form of a dependency matrix with has as entries the length of the dependency between the resources (1 for direct dependency and path length +1 for indirect dependency). A “0” in the dependency matrix means no dependency.

Table 1: Resource Dependency Matrix example

|  |  |  |  |
| --- | --- | --- | --- |
|  | R1 | R2 | R3 |
| R1 | 0 | 0 | 0 |
| R2 | X | 0 | 0 |
| R3 | Y | 0 | 0 |

From this it is defined that two resources are inter-independent if there is no direct dependency between them and if the length of the indirect dependency chain is over a certain limit. By separating the resources in IIRGs solutions for context repair can be searched in parallel over the IIRGs, thus reducing the system’s search space and improving the running time.

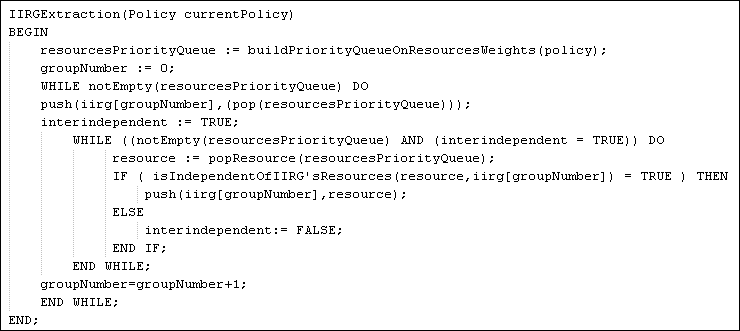


Figure 4: IIRG Extraction Pseudo-Code

For a better understanding of the IIRG extraction process a trace is provided on an example where it is assumed that the resources weights follow this rule : weight(R2) >weight (R3) >weight (R1).



Figure 5: IIRG Extraction Algorithm Trace

### RAP instantiation for the self-healing system



For using the RAP model for implementing a self-healing environment its elements must be mapped onto real world entities.

Each sensor can have associated one or more actuators which have an influence on it. Each actuator has at least one action defined, each action having its effect represented by a **+ value (increment action), - value (decrement action)** or **value (set value action).**

Table 2: Self-healing environment RAP mapping

|  |  |  |
| --- | --- | --- |
| <R,A,P> entity | Self-Adapting context entity | Details |
| Resources | **Sensors** | * Smart room sensors |
| **Actuators** | * Components capable of influencing the environment |
| Actors | **Actions** | * Each actuator has associated actions |
| Policies | **User specified policies** | * Acceptable sensor values combinations for the environment |

### RAP instantiation for the self-adapting datacenter

The previous RAP instantiation is still used in the self-adapting datacenter for environmental control but a new mapping is needed for task management. Due to the reduced number of possible datacenter actions (excepting environmental control) the new mapping is simpler.

Table 3:Self-Adapting Datacenter RAP Mapping

|  |  |  |
| --- | --- | --- |
| <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** | * Servers green performance indicators ranges |
| **QoS policies** | * Task requested resources ranges |

### Load Distribution Strategy

For optimizing the energy consumption server skewing with tails is employed. Based on workload predictions, one or more servers are kept online for accommodating future loads. Also based on this prediction a **load threshold** is computed for each tail and when that threshold is reached a new tail server is brought online.



Figure 6: Server Skewing with Tail

## Reinforcement Learning Algorithm

For finding the best sequence of actions to bring the system in an acceptable state an algorithm based on reinforcement learning was developed. The advantage of basing this approach on reinforcement learning is the generality provided by this type of learning. Any real world situation can be represented in terms of <action, reward> pairs. Also, by manipulating the expected reward the mobile agent that uses this algorithm can take any desired course of action.

The desired context states are described using policies. Based on these policies the algorithm computes the entropy value for the current context state. If the entropy value is above 0 (or above a user-defined threshold) the search for the repairing sequence of actions is triggered. For each broken policy, the algorithm takes each broken resource and tries to change its value in an acceptable one by executing actions on associated actuators. Each associated action executed is simulated, the resulting state’s entropy is computed and the state is placed in the states priority queue, sorted by entropy. The next state to be expanded is the top of the queue, which holds the best state found so far.

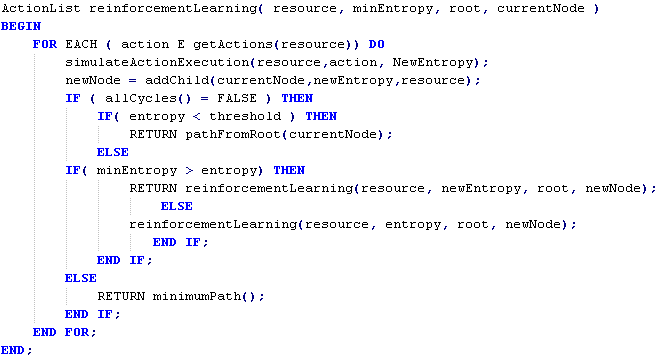


Figure 7: Reinforcement Learning Algorithm DE REFACUT EVENTUAL PSEUDOCODU

Due to the generality of the reinforcement learning approach, the previously described algorithm can be applied easily both to a self-healing smart environment and to a self-adapting datacenter.



Figure 8: Self-Healing system Reinforcement Learning Process

Due to the real-time performance required from real world datacenters, in the datacenter version the algorithm can be stopped at any given moment in time and the best action sequence found so far is used. This enables the system to stop searching for the perfect solution if a certain time period has passed and just use the best solution found so far.



Figure 9: Datacenter Reinforcement Learning Process

## QoS-Energy Consumption Negotiation

Having a mechanism for negotiating between QoS and Energy consumption is required for the algorithm to be applicable to the real world. As described in 3.1.7 RAP instantiation for the self-adapting datacenter, each server has associated green performance indicators. These indicators specify the optimum server load for maximizing the performance-per-watt ratio. The performance indicators are specified as ranges. For example, a server could have an optimum CPU load of 60%-80%. The task resource requirements are also specified in ranges, for example for CPU 500 MHz-800 Mhz.

In Figure 10 a situation where negotiation is required is presented. In such a situation the maximum resources required by a task can’t be allocated, but there is room to fit the task in the required resources range. A basic approach would be to allocate the minimum resources requested by the task. But this might lead to situations in which the maximum optimum server load is not reached, but some task performs in a degraded mode with the minimum resources needed to run. For avoiding this kind of situation, a tradeoff mechanism is needed. The tradeoff solution presented here follows the reinforcement approach taken to system management in this book.



Figure 10: Situation which requires negotiation

**Desirability** is defined here as the satisfaction of the system to be in a particular state. This concept can be applied both to server optimum load indicators and to task requested resources range. Desirability information can be added in the context by using a bidimensional function, one dimension for desirability and the other for the actual value. Such a function is presented in Figure 11. The desirability is represented in the Y axis, while the corresponding values are on the X axis.



Figure 11: Desirability-Based Negotiation

The negotiation is done by taking the requested task range and the available server range and checking if the two ranges overlap. If so, the overlapping range is extracted. A desirability function is assigned to the overlapping range based on the QoS and Energy Consumption importance. The result is another bidimensional function. In Figure 11 the desirability of the task is assigned to the overlapping range in order to achieve a greater QoS. Finally, the negotiation result is the **Center Of Gravity** of the area given by the previously generated bidimensional function using Equation 3.

Equation 3: Center Of Gravity Formula

|  |
| --- |
|  |

## The Conceptual Architecture

Having an agent-based architecture, the system mainly consists of four agents all interacting with the same ontology: Context Model Administering Agent (CMAA DE PUS SI ALEALALTE MODEL NU MANAGEMENT), Context Interpreting Agent (CIA), GUI Agent and Reinforcement Learning Agent (RLA). The CMAA reads a context description and loads the information into the shared ontology. CIA periodically queries the sensor data and refreshes the context information. The GUI Agent is responsible of the graphical context representation. RLA is the most important agent. It detects when the context is broken and searches for actions to bring it back in an acceptable state.

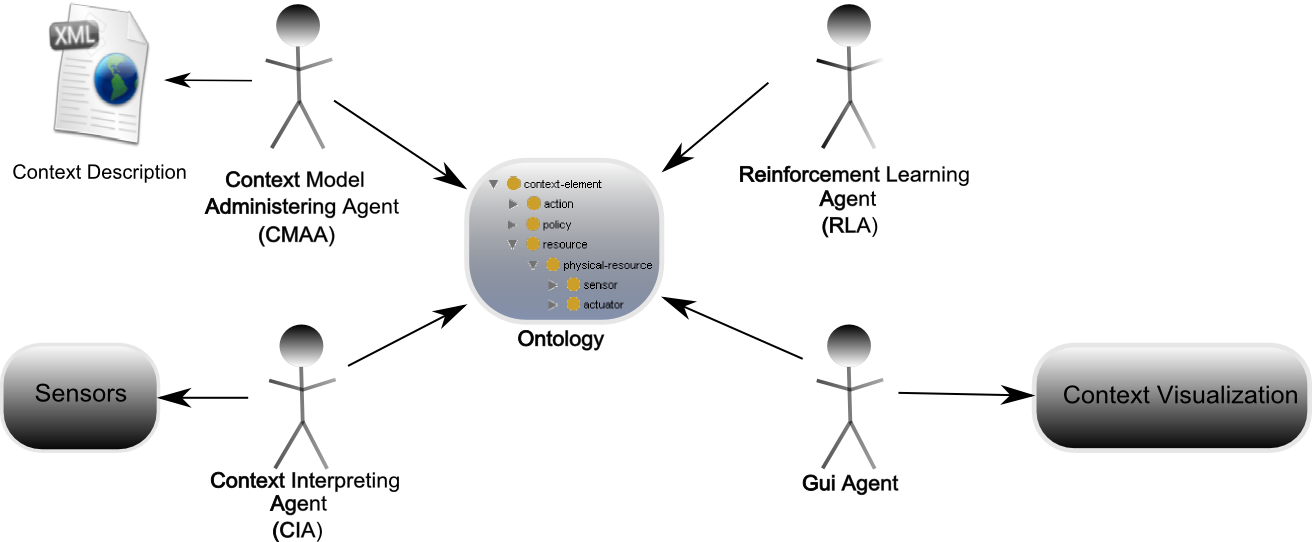


Figure 12: System Conceptual Architecture

# Implementation

The current chapter focuses on the development and implementation of the proposed agent- based context management framework. It gives a closer look at the technologies used to implement the context management components presented in the previous chapter, and then it explains how these technologies were used to obtain the proposed goals.

## Architecture

DE ASIGNAT CUIVA ACTION ENFRCEMENT UNIT

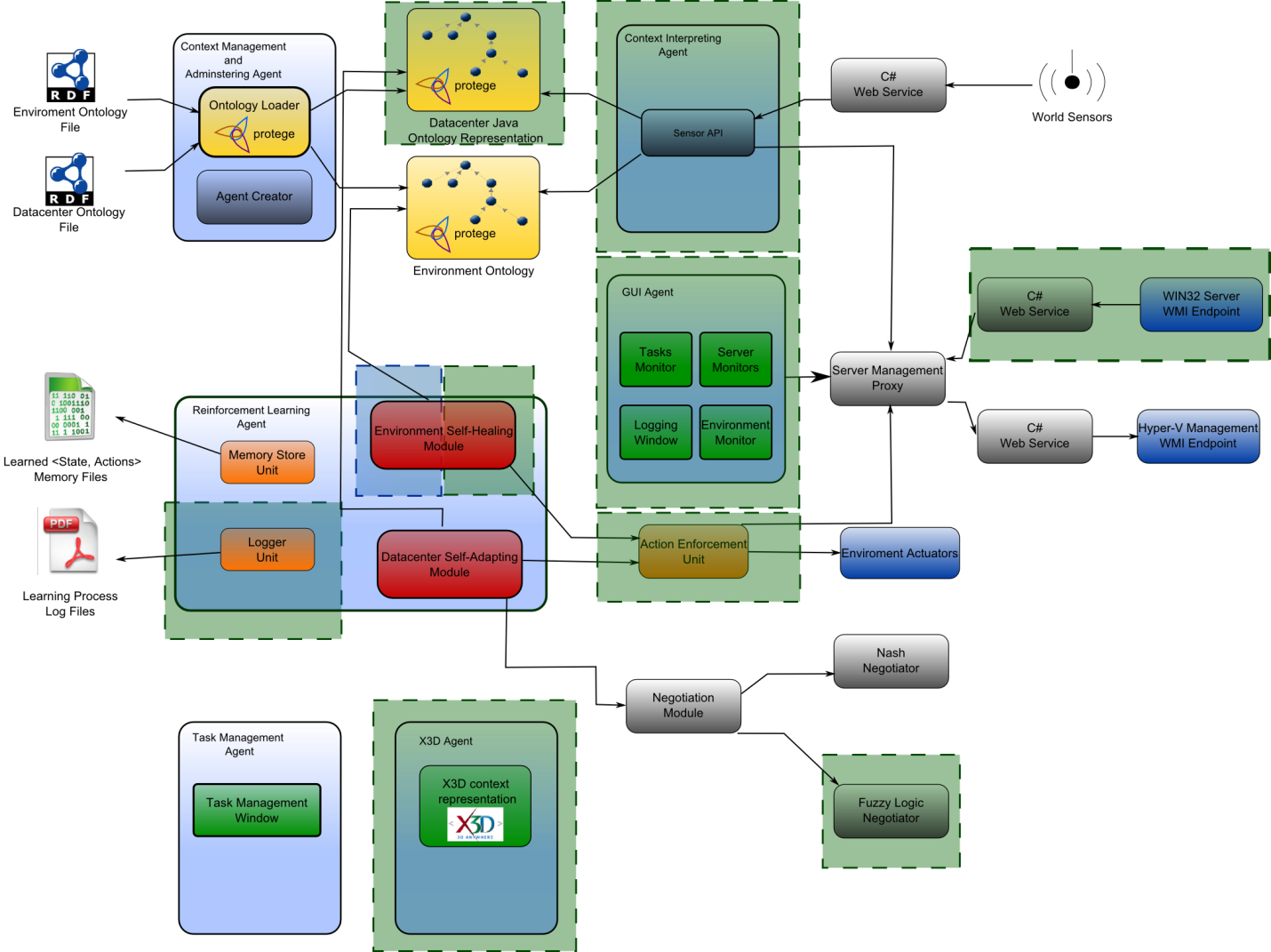


Figure 13: Architecture

Only the components developed by me (highlighted in Figure 13) will be presented in detail. The other components are detailed in [ 25 ] and will be mentioned and properly referenced when needed .

The “Environment Self-Healing Module” (highlighted both with blue and green) was developed by me and the author of [ 25 ] for the CONSENS research project [ 27 ] and as such it is a joint development effort, also described in [ 25 ].

DE BAGAT PARTEA CU CONSENSU SI ACOLO UNDE AM ZIS K JOINT LA DESIGN SAU SOMETHING

## System Input

For any knowledge-based system to be able to reason about the surrounding world it is necessary to have a mechanism of describing that world. The selected mechanism is **Ontology Representation**.

The **system** is described using the two previous ontology representations (Figure 2 and Figure 3). Both the datacenter and environment ontology representations are built using Protégé [ 28 ] and populated with instances. The instances represent real world physical elements.



Figure 14: System Ontology Representation

The **system acceptable states** are defined by policies specified under the form of **SWRL (Semantic Web Rule Language)** [ 30 ] rules. By specifying the policies in SWRL the system uses **Pellet** [ 31 ] OWL2 Reasoner for policy evaluation. The SWRL rules evaluation triggers at each change in the ontology variables, providing a reliable mechanism for context evaluation.

|  |
| --- |
| Listing 1: SWRL QoS Task 1 Policy |
| J:\cercetare\paper suceava\deliverables\source images\swrl_QoS_policy.png |

The **environment sensors** are described in xml in a world description file. The world description file is parsed and the specified values are set on existing sensors instances. ORI DE SCOS ORI DE BAGAT IN ARH MARE SI FISIERU ASTA K INPUT

|  |
| --- |
| Listing 2: Sensor XML Description |
|  |

The **world information** is accessed using **ASP.NET Web Services** [ 36 ]which provide the interface between world datacenter sensors and resources. This approach was chosen because Java [ 37 ] does not provide such a tight coupling with the underlying operating system. C# **Error! Reference source not found.**[ 38 ] provides access to the underlying operating system’s functions and thus allows access and real time monitoring of system’s hardware.

## System Output

The most important output provided by the system is the **context repairing sequence of actions**. Having two system management components, one for environment self-healing and one for datacenter self-adapting, there are separate outputs for each component. The repairing sequence of actions is as the name suggests, a sequence of actions which brings the system in an acceptable state from a state in which some policies where broken. The found actions are executed on the corresponding targets.

As secondary output the system stores the <context state, action sequence> pair for future use and creates a Pdf file containing the context monitoring and repair log, for future analysis.

|  |
| --- |
| Listing 3: Environment Management Log |
| J:\cercetare\licenta\pdf_log.png |

## Architecture Implementation

For implementing the agent-based architecture the Java Agent Development Framework (JADE) [ 33 ] was used. JADE is a software Framework fully implemented in Java which simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA (Foundation for Intelligent Physical Agents) specifications [ 35 ]. Another description of Jade would be : JADE 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 [ 34 ].

There are several advantages for using JADE. JADE is based on a peer-to-peer architecture so each peer is able to initiate a communication or be subject to a request, providing a fully distributed system. Also, it implements the agent paradigm, applying concepts from artificial intelligence to distributed systems. Agents are active entities, that can refuse a request and which are loosely coupled, thus providing a flexible infrastructure.

JADE supports agent migration. Each instance of the JADE run-time is called a *container.* The set of all containers is called a *platform* and the platform provides a layer that hides the complexity of the underlying hardware. This enables the agents to migrate from one container to the other in real time.



Figure 15: Jade Architecture

DE MAI BAGAT DESCRIERE JADE CU FIPA CU BLA BLA BLA

## Ontology Representation Implementation

### Environment Ontology Implementation

Based on the .owl ontology description, a *JenaOWLModel* [ 32 ] is generated and accessed with the specific Jena API. This is further detailed in [ 25 ] by its author.

### Datacenter Ontology Representation

From the datacenter ontology file description an *OWLModel* is created using Protégé OWL Loader [ 29 ]. Using the Protégé [ 29 ] Ontology Editor tool for generating Protégé-OWL Java code a class structure respecting the ontology hierarchy and containing all the ontology properties is generated. This hierarchy of generated classes has as core the *OWLModel* created from the ontology file.



Figure 16: Ontology to Java Code Mapping

This is a more object oriented approach, the ontology entities being translated into Java classes and used as any other class. Also, having Java classes to work with instead of just an underlying ontology model, the ontology concepts can be enhanced with Java specific properties such as *Serializable* or *Cloneable* , providing a better integration of the ontology representation with the rest of the regular Java code. Another major advantage of using Java classes instead of directly accessing the OWLModel or OntModel is that all the properties values can be set directly in the OWL slot without using the higher OntModel functionality. This proved to be an advantage in the experimental results phase when by using a home-brewed policy evaluation mechanism a major improvement in performance was obtained. This improvement was obtained if the properties values were not set also on the OntModel and such the Pellet SWRL evaluation mechanism did not trigger at every property change event.

Another advantage of avoiding using *OntModel* is that it throws *ConcurrentAccessException* if a read and write occur at the same time, due to the internal reasoning process, while the *OWLModel* does not enforce access ordering.



Figure 17 : CPU Entity Ontology Representation

|  |
| --- |
| Listing 4: CPU Entity Java Representation |
|  |

## Agents Implementation

### Context Interpreting Agent (CIA)

The Context Interpreting Agent is responsible for synchronizing the ontology representations with the real context. The agent is implemented by the *CIAgent* class found in contextawaremodel.agents package.

Upon creation, the agent receives from the Context Management Agent described in [ 25 ] the ontology models for the environment and datacenter. These models are used by the single behavior attached to the agent: *ReceiveMessagesCIABehaviour*. The *ReceiveMessagesCIABehaviour* extends *CyclicBehaviour* and handles the messages received by *CIA*, informing it of newly created instances. For each new instance of *Sensor* and *Server* the behavior adds the instance to the *SensorAPI*.

The **SensorAPI** is based on the API described in [ 26 ].It implements a pooling data gathering mechanism. At a certain time interval the web service associated to the world element (*Sensor* or *Server)* is accessed and the new data is used to update the ontology representations. This API provides two static methods used for registering a *Server* or a *Sensor* to the pooling mechanism.

|  |
| --- |
| Listing 5: SensorAPI functionality |
|  |

### Reinforcement Learning Agent (RLA)

This agent represents the core of the infrastructure. It contains the ontology reasoning and solution search processes. The implementation of the agent is available in the *ReinforcementLearningAgent* class located in the *contextawaremodel.agents* package.

Upon initialization, the agent receives from the CMA an *OntModel* together with its underlying Protégé *OWLModel* for both environment and datacenter ontology representations*.* The *OntModel* is used by Pellet [ 31 ] for evaluating SWRL [ 30 ] rules, while the *OwlModel* is used in conjunction with a Protégé [ 28 ] generated *Ontology Factory* for ontology management.

The two main components of this agent, the **Environment Self-Healing Module** and the **Datacenter Self-Adapting Module** are implemented as behaviors extending *TickerBehaviour* . The environment management module is implemented in *contextawaremodel.agents.behaviours* in *ReinforcementLearningEnvironmentManagementBehaviour* and the datacenter management module in *ReinforcementLearningDataCenterManagementBehavior*.

#### ReinforcementLearningEnvironmentManagementBehaviour

This behavior implements the algorithm presented in [ 5 ] in the context of maintaining the state of the environment within acceptable parameters. The implementation is a joint development effort done in the context of the CONSENS research project [ 27 ] and is also described in [ 25 ].

Extending the *TickerBehaviour* , the ReinforcementLearningEnvironmentManagementBehaviour implements an *onTick()* method which is called at a specific time interval specified when creating a new instance. The *onTick()* contains the business logic needed for the self-healing system. First the context entropy is computed using Equation 1. If the computed entropy is different than 0 then the reinforcement action search begins.

For each broken policy, for each broken resource from the policy and for each actuator associated to the policy all the possible actions are simulated and the corresponding states’ rewards are computed. Each action simulation has as effect the creation of a new state which is placed in a list of states, sorted after reward. If no resulting state had the entropy 0, the search continues with the next best state (the state with the highest reward from the states list) and repeats the actions described above. If a sequence of actions which can repair the context exists it will be found and enforced with the help of the *Action Enforcement* *Unit*.

#### ReinforcementLearningDataCenterManagementBehavior

This behavior extends the algorithm presented in [ 5 ] with enhanced reward computation and elements specific to datacenter management. This behavior is presented in detail in [ 25 ] by its developer.

#### Logger Unit

This component has the role of logging the context state, the broken policies and the actions taken to repair the context. The log is saved under the form of a PDF file as in **Error! Reference source not found.** using IText [ 39 ]. IText is a library used to generate PDF files. The advantage over other libraries is that it supports several mechanisms for PDF creation: java Graphics can be used to “draw” the PDF or specific library classes can be used to create a document structure under the form of paragraphs.

#### Memory Store Unit

This component stores <context state, actions> pairs for future use. When the context is evaluated, if some policy is broken it is checked if the current state hasn’t been encountered before and if it has the action sequence associated to it is executed. Described in detail in [ 25 ] by its author.

### X3D Agent Implementation

The implementation is available in the X3DAgent class located in the *contextawaremodel.agents* package.

For providing a realistic context representation X3D [ 40 ] was chosen. X3D is a scalable and open software standard for defining and communicating real-time, interactive 3D content for visual effects and behavioral modeling. X3D provides both the XML-encoding and the Scene Authoring Interface (SAI) to enable both web and non-web applications to incorporate real-time 3D data, presentations and controls into non-3D content. X3D is the successor to the Virtual Reality Modeling Language (VRML). It improves upon VRML with new features, advanced APIs, additional data encoding formats, stricter conformance, and a componentized architecture. The advantage of using X3D is that it provides a simple and easy to use mechanism for defining 3D scene navigation and interaction.

Xj3D [ 41 ] is a toolkit for VRML97 and X3D content written completely in Java. Although at first Xj3D was based entirely on Java-3D, it has moved from that state and now it provides a faster and less CPU intensive rendering engine.



Figure 18: Xj3D Architecture

The context representation is described in an .x3d XML file which is loaded by XJ3D and rendered. In the X3D description file every object is wrapped into at least one *Transform* node which specifies the object scale, rotation and translation. Inside the *Transform* node a *Shape* node represents the 3D shape. The *Shape* contains the object’s *Material* and definition. The *TouchSensor* node is used for capturing user input. And finally, everything is wrapped in a *Scene* node.

|  |
| --- |
| Listing 6: X3D File Shape Entry |
|  |

The role of the X3D Agent is to provide a meaningful context representation which can be used for simulating various scenarios or for real-time monitoring of the surrounding environment. More details about the simulated contexts are presented in the Experimental Results chapter.

### GUI Agent Implementation

The implementation is available in the GUIAgent class in the *contextawaremodel.agents* package. The GUI Agent is responsible of the creation and management of the user interface elements used for information output. There are four main user interface components handled by this agent: Environment Monitor, Tasks Monitor, Server Monitors and Logging Window.

#### Environment Monitor

This component is responsible for displaying the environment sensors values, the broken environment policies and the actions taken by the self-healing algorithm.



Figure 19: Environment Monitor GUI

#### Tasks Monitor

This component is responsible for displaying the pending tasks queue together with information about the requested resources.



Figure 20: Task Monitor GUI

#### Server Monitors

This component is the most complex context monitoring user interface module used in this application because it displays information about the running tasks, the total resource usage and the resource usage per task.

For managing the complexity of the Server Monitor components layout, the components are organized in a tree-like hierarchy, with smaller components being included in larger ones and so on until the entire monitor is created.

ADAUGA POZA CU SERVER MONITORU

#### Logging Window

The logging window is responsible for displaying periodically the context state, the broken policies and the actions taken for bringing the context in an acceptable state. This component is important because it provides the context management history and makes it easier to trace the system. Two log windows are provided, one for the environment self-healing and one for the datacenter self-adapting systems.



Figure 21: Datacenter Management Log

## Utility components

### Fuzzy Logic Negotiator

The component is implemented in the *FuzzyLogicNegotiator* class in the *negotiator.impl* package. The Fuzzy Logic Negotiator is the implementation of the negotiation mechanism described in 3.3 QoS-Energy Consumption Negotiation. The implementation of the FuzzyLogicNegotiator relies on JFuzyLogic [ 42 ] , which implements the Fuzzy Control Language(FCL) specification [ 43 ].

JFuzzyLogic is a package written in Java which provides a mechanism for loading and dynamic management of FCL files. Using JFuzzyLogic the user can define any membership function, modify values or ranges and evaluate the fuzzy rules described in the input file. For deffuzzification JFuzzyLogic implements entirely the FCL specification, supporting Center of Gravity, Centre of Area, Left Most Maximum, Right Most Maximum and Centre of Gravity for Singletons. In Fuzzy Logic, each value has associated a membership value. The membership represents the degree to which the value belongs to the specified range.

For implementing the 3.3 QoS-Energy Consumption Negotiation mechanism, the **desirability** is used as the membership function. By using desirability for membership, less desirable values are considered to belong in a smaller proportion to a specific range so they have a smaller influence. The bidimensional function used by the negotiation mechanism is defined as a (value, desirability) pair for each inflexion point as shown in Listing 7.

|  |
| --- |
| Listing 7: FCL specification |
|  |

### Server Management Proxy

For connecting the system described in this book to a real-world datacenter an interface between this system and the datacenter is needed. Also, the interface should have a high degree of flexibility.

In order to support different *ServerManagementProxy* subclasses, needed when porting the system from one datacenter to another, the Factory Method Design Pattern is used to provide a flexible and transparent mechanism for proxy instantiation. The pattern is realized using the *ProxyFactory* class located in the *contextawaremodel.worldInterface.datacenterInterface.proxies.impl* package and is depicted in Figure 22.



Figure 22: Server Management Interface Architecture

The *ServerManagementProxyInterface* interface specifies the functionality needed for managing a datacenter and represents the contract which every concrete server management proxy must implement, as presented in Listing 8.

|  |
| --- |
| Listing 8: Server Management Interface |
|  |

### Server Information Gathering Endpoint

This component was created to provide real-time server resources usage data. The implementation was developed for servers running Windows Server 2088 R2, using operating system specific functions.

For gathering server data the WIN32 Classes [ 48 ] provided trough the Windows Management Instrumentation(WMI) [ 47 ] where used. WMI offers the means to access system information using Win32 classes from any .NET supported language. C# was chosen as the host language due to its support for web services and tight coupling with the underlying operating system. The reason Java wasn’t used for information gathering is that the platform independency of the language makes it hard to query hardware specific information. C# does not have this drawback, being able to access both hardware and software information about the host system. Another advantage of using C# is the easiness with which functionality can be exposed under the form of web services.

The web service created for exposing the functionality of the WMI server information gathering module are hosted on Microsoft IIS Application Server [ 49 ] and accessed through HTTP [ 42 ].

|  |
| --- |
| Listing 9: WIN32 WMI Specific Functions |
|  |

# Testing and results

The concepts and algorithm described in 3 Analysis and Design where tested on a simulated smart room, on a simulated datacenter and on real servers.

## Self-Healing Environment Test

This test scenario involves applying the self-healing reinforcement learning algorithm described in 3.2 to a smart environment. It has as goal to demonstrate the versatility and correctness of the self-\* framework described in this book by using it to manage a smart laboratory. s

For our test scenario we have chosen an environment having a computer, a camera with face recognition, a light source and an alarm (Figure 2 and Figure 3). Each environment component has attached a sensor for monitoring its state. Also humidity, temperature and room person count sensors have been added to increase the environment’s complexity. Each sensor has a set of possible values encoded using integers for internal representation as described in Table 4. There are three policies which our algorithm has to enforce: *light policy,* *face* *recognition policy* and *temperature and humidity* policy ( Table 5). For interacting with the environment, actuators have been associated to the sensors they have an effect on as shown in Table 6.

Table 4: Sensors Values

|  |  |  |
| --- | --- | --- |
| Sensor | Possible values | Value encoding |
| Temperature | Z | Any integer |
| Humidity | [0-100]% | [0-100] |
| Light | {ON, OFF} | {1,0} |
| Face Recognition | {Professor, Student, Unknown} | {0,1,2} |
| Computer State | {ON, OFF} | {1,0} |
| Alarm State | {ON, OFF} | {1,0} |
| Room State | {Empty, Not Empty} | {0,1} |

Table 5: Policies

|  |  |
| --- | --- |
| Policy | Accepted values combination for policy |
| Temperature And Humidity |  |
| Face Recognition |  |
| Light |  |

Table 6: Sensors Associated Actuators

|  |  |  |
| --- | --- | --- |
| Sensor | Actuator | Available actions |
| Temperature | Air Conditioning Unit | {Decrease by 5 , Decrease by 2 } |
| Heater | { Increase by 5 } |
| Humidity | Humidity Controller | {Increase by 3 %,Decrease by 3 %} |
| Light | Light Controller | {Turn ON, Turn OFF} |
| Computer State | Computer Controller | {Turn ON, Turn OFF} |
| Alarm State | Alarm Controller | {Turn ON, Turn OFF} |
| Face Recognition | - | - |
| Room State | - | - |

Two tests have been performed on this scenario. The first test is meant to show the effect of learning over the algorithm running time. The second test is an interactive approach in which all the possible combinations of broken policies are tested by a human participant.

In order to have a scenario as realistic as possible, C# ASP.NET Web Services where used to simulate the environment sensors. One web service was built for each environment sensor The actuators influence the sensor values by changing the target web service value according to the specified effect.

|  |
| --- |
| Listing 9: Alarm Sensor Simulation Web Service |
|  |

### Running Time Test

For the first test, the program was run for 28 hours. During this time all sensors received random values as following: for Temperature from 15 to 25, for Humidity from 15 to 35, 0 or 1(“OFF”, “ON”) for Light, Room State, Computer State and Alarm State sensors and 0, 1, 2 (“Professor”, “Student”, “Unknown”) for the Face Recognition Sensor. After the algorithm found a solution the random values assignment was made. The running time of the algorithm is depicted below in Figure 23.



Figure 23: 28h Random Values Test

Each spike in the plot represents a situation in which an unknown context state has been encountered and a search for the best sequence of actions was performed. In the first 10000 seconds, almost all the running times of the action selection algorithm are larger than 50 seconds. As the knowledge base is being populated with learned knowledge the action search running time decreases, reaching 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.

### System Output Test

For the second test an interactive X3D context representation was built using the X3DAgent (Figure 24). The representation allows the user to interact with each of the environment actuators by mouse click events. By allowing the user to perform actions on the environment actuators and not on the sensors themselves, the representation can be used to test the self-healing solution on real-world systems. For altering the environment, the user has to click on the corresponding actuator and the actuator action will be performed. For the *AirConditioningUnit* actuator only the *Descrease* by 5 degrees action is available during simulation. Also, given that the *FaceRecognition* and the *RoomState* sensors do not have actuators, the user can click on their representation and the sensor state will scroll through the available states to the next state in the set.



Figure 24: X3D Smart Environment Representation

Using this simulated environment the output of the system is checked by a human for every possible input scenario. The user modifies the environment such that some policies are broken and then monitors the system’s output sand checks if the suggested sequence of actions is the best for bringing the environment in an acceptable state.

For better understanding, a trace of the algorithm is presented on the following case: a professor enters in the room, the Computer is “OFF” and the Alarm is “ON”, breaking the Face Recognition Policy. For each resource which doesn’t have an acceptable value for the current policy the algorithm simulates executing each action attached to the actuators of the current resource and generates a new context. The first resource which is broken is the Computer, on which we take every action which brings it to a state different from the current state. Here only one action is possible, so the Computer will be “Set” to “ON” (setting it to “OFF” does not change its state so this action is not considered). For each resulting state we continue with the next sensor that breaks the policy, the Alarm sensor. Similar to the previous sensor, the Alarm can only be “Set” to “OFF”. By performing these actions the Face Recognition Policy is fixed. This situation is depicted in Figure 25.

**

Figure 25: Face Recognition Policy broken

## Self-Adapting Datacenter Test

For this testing scenario the solution described in this book was extended in order to be applied to an energy-efficient datacenter. This test is meant to further prove the versatility of the solution proposed in this book and to show that self-adaptability and self-healing can be considered isomorphic. The isomorphism between the two self-\* properties can be deduced from the fact that they can be both realized with the same domain model and the same property enforcement protocols.

A second environment ontology representation shown in Figure 3 which contains datacenter specific information has been added and used in the datacenter self-adapting process. The original ontology representation was used for environment self-healing.

The self-adapting extension was tested first on a simulated datacenter and after the solution proved correct on a small home-brewed datacenter containing two servers and a shared storage.

### Simulated Datacenter Test

An X3D representation of a datacenter was built, containing 5 servers and a central control unit arranged in a spoke and arrow layout (Figure 26). The datacenter has one Temperature and one Humidity sensor. There is one sensor actuator depicted here, a giant fan representing the datacenter cooling system. The servers are represented as wireframe towers with a solid base. The base color represents the state of the server: gray means *offline* and blue means *online*. The tasks are represented by purple boxes. Each server has a power meter attached which shows the power consumption.



Figure 26: X3D Datacenter Representation

For this test all the tasks are considered having the same priority. Also, the algorithm is run for the first time so no action memory exists. For the real case, if the current context has a sequence of actions associated to it in the memory, that sequence of actions is executed or added to the existent path of actions reached so far. For each task, a policy is generated, containing the specified values for QoS and SLA attributes. After negotiation between this policy and the Energy policies, new relaxed QOS and Energy policies are generated as result.

The negotiation part described in 3.3 QoS-Energy Consumption Negotiation was not yet plugged in the algorithm, thus arriving the necessity of finding a deploy strategy for which at least a solution exists. Otherwise the algorithm would return the best path it has found and at the next context evaluation time it will search again for ways of repairing the context.

#### Hardware infrastructure

Concerning the hardware infrastructure, the simulated datacenter contains 5 servers described in the table below (Table 7). For each server a policy and associated SWRL rule containing key performance indicators are generated and used in the optimization process.

Table 7: 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 |

For simulating a real world scenario we have chosen 5 tasks which are received by the datacenter and are considered to have an infinite life span. The tasks are received in the order they are in the table below (Table 8).

Table 8: 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 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.

#### Trace

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(Figure 27), wake up server 3(Figure 28), deploy task 2 on server 3(Figure 29) and deploy task 3 on server 3(Figure 30). 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 accepted state.



Figure 27: Deploy Task 1 on Server 1 Action



Figure 28: Wake up Server 3 Action



Figure 29: Deploy Task 2 on Server 3 Action



Figure 30: 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.

#### Overhead

After running the algorithm a few times it became clear that the overhead generated by the **Pellet** **OWL 2 Reasoner for Java** [ 31 ] when evaluating SWRL rules is unacceptable in a real-time application, the rules being evaluated at each change in any ontology value. So SWRL rules where disabled and internal reasoning was implemented using Java. This leads to a decrease in running time of up to 4 times, depending on the type of processor the global loop runs on.

### Datacenter Test

The self-adapting extension was also tested on a home-made datacenter running virtual machines. The datacenter architecture, as presented in Figure 31 consists of a server cluster (2 servers in this case), a shared storage used by the cluster, environment sensors and a computer running the self-adapting controller. The shared storage is used to host the virtual machines in order to be accessible from any server in the task migration process. For this test the environment sensors are simulated as described in 5.1 Self-Healing Environment Test .



Figure 31: Test Datacenter Structure

The software architecture of the test datacenter’s servers is presented in Figure 32. The servers in the test datacenter are running Microsoft Windows Server 2008 R2 [ 45 ] as operating system. The hypervisor installed on the servers is Hyper-V Server R2 [ 47 ]. For this test 2 server where used, each server having a 2 GHz Dual Core processor, 2 GB of RAM and a 150 GB hard drive.



Figure 32: Server Technology Mapping

For interacting with the datacenter C# WMI classes where used both for gathering server resources usage data and to access the hyper-v server running on the servers in order to move, import and destroy virtual machines dynamically. The C# classes functionality was exposed trough ASP.NET web services, published on Microsoft IIS Application Server. The web services are called from within the *Hyper-VManagementProxy* class located in the *contextawaremodel.worldInterface.datacenterInterface.proxies.impl* package and which implements the *ServerManagementProxyInterface*. The Hyper-V management proxy is described in detail in [ 25 ] by its author.



Figure 33: Server Interaction

As datacenter workload three virtual machines running Windows XP [ 53 ] where created using the Hyper-V Management GUI. Due to the lack of time, no dynamic virtual machine resource allocation mechanism was built. Given this fact, the virtual machines have allocated the minimum resources amount specified in Table 9 and only inside the self-adapting logic the ranges are used. The task requirements were chosen based on the datacenter server’s configuration to use less RAM in order to be accommodated on the server. Windows Server 2008 R2 together with Hyper-V Server, IIS Server and other programs uses around 1GB of RAM, so the amount of RAM the virtual machine use was a limitation.

Table 9: Virtual Machines Resource Allocation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Task Nr | Task Requirements | | | | | | |
| CPU (Mhz) | | Memory (MB) | | Storage (GB) | | |
| Min | Max | Min | Max | Min | max | |
| 1 | 100 | 500 | 100 | 300 | 1 | | 2 |
| 2 | 2 x 300 | 2 x 500 | 100 | 300 | 1 | | 2 |
| 3 | 2 x 100 | 2 x 200 | 50 | 400 | 1 | | 2 |

Because Hyper-V Server disables the Sleep and Hibernate options, a little hack was employed. The Hyper-V Server was banned from running automatically when the system started. The send *server to low power state* action was implemented in two steps: the server is restarted and a small program is set to run on startup, which sends the server in sleep. To wake up a server a magic packet containing 16 times the server’s MAC address is send to the datacenter router which forwards it to port 9 of the server. By enabling Wake On LAN from the server network card, the event of receiving the magic packet will wake up the server.

# Conclusions

# 

# References

[ 1 ]Green **Active Management of Energy in IT Service centres:** Performance Computing Center Stuttgart (HLRS), <http://www.hlrs.de/organization/av/isis/research/distributed-systems/games/>

[ 2 ] **VMware Whitepaper**: How VMware Virtualization Right-sizes IT Infrastructure to Reduce Power Consumption, <http://www.vmware.com/files/pdf/WhitePaper_ReducePowerConsumption.pdf>, pages 4

[ 3 ] **EPA** [**Data Center Report to Congress -FINAL 7-25-07**](http://www.federalnewsradio.com/pdfs/EPADatacenterReporttoCongress-August2007.pdf)**,** <http://www.federalnewsradio.com/pdfs/EPADatacenterReporttoCongress-August2007.pdf>, Pages 6

[ 4 ] M. Poniatowski: **Foundations of Green IT: Consolidation, Virtualization, Efficiency, and ROI in the Data Center,** Prentice Hall, 2009, ISBN-10: 0137043759, ISBN-13: 9780137043750, 978-0137043750, pages xiii

[ 5 ]T.Cioara, I.Anghel, I.Salomie, M.Dinsoreanu, C.Georgiana and **M.Daniel**: **A Reinforcement Learning based Self-healing Algorithm for Managing Context Adaptation,** First International Workshop on Communication, Collaboration and Social Networking in Pervasive Computing Environments (PerCol 2010)

[ 6 ] ALA CU AUTONOMIC IN DATACENTERS

[ 7 ] Ubiquitous Computing, <http://en.wikipedia.org/wiki/Ubiquitous_computing>

[ 8 ] S.Russell, P.Norvig : **Artificial Intelligence A Modern Approach**, Second Edition, Prentice Hall,ISBN-10: 0137903952, ISBN-13: 978-0137903955

[ 9 ] P.C.K. Hung, H. Li: **Web Services Discovery based on the Trade-off between Quality and Cost of Service: A Token-based Approach**

[ 10 ] C. Sakama, K. Inoue: **Negotiation by Abduction and Relaxation**

[ 11 ]Abductive Logic Programming, <http://en.wikipedia.org/wiki/Abductive_logic_programming>

[ 12 ] Y.Chevaleyre, U.Endriss, S.Estivie, N.Maudet : **Reaching Envy-free States in Distributed Negotiation Settings**

[ 13 ] Q.B.Vo, L.Padgham: **Searching for Joint Gains in Automated Negotiations Based on Multi-criteria Decision Making Theory**

[ 14 ] K.Fujita, T.Ito, .Klein: **Finding Nash Bargaining Solutions for Multi-issue Negotiations: A Preliminary Result**

[ 15 ]D. Baker, D. Georgakopoulos, M. Nodine and A. Cichocki: **From Events to Awareness**

[ 16 ] P.Hu, J.Indulska, R,Robinson, An Autonomic Context Management System for Pervasive Computing

[ 17 ] IEEE 1451, <http://ieee1451.nist.gov/intro.htm>

[ 18 ]SensorML, <http://vast.nsstc.uah.edu/SensorML/>

[ 19 ] R.Das, J.o.Kephart, C.Lefurgy, G.Tesauro, D.W.Levine, H.Chan : Autonomic Multi-Agent Management of Power and

Performance in Data Centers D SI NOI AM AVUT PROBLEME EXACT K ASTA :A SEVRER TURN OFF SI TIMPU MAREEEE DE PONIRE ASA K AM PUS SLEEP :D

// DE BAGAT SI ASTA UNDEVA R.M.Bahati, M.A.Bauer : **Adapting to Run-time Changes in Policies Driving Autonomic Management**

[ 20 ]G.Chen, W.He, J.Liu,S.Nath,L.Rigas,L.Xiao,F.Zhao: **Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services**

[ 21 ] D.Kusic,J.Kephart, J.E.Hanson, N.Kandasamy and G.Jiang:**Power and Performance Management of Virtualized Computing Environments Via Lookahead Control**

[ 22 ] M.Amoui, M.Salehie, S.Mirarab, LTahvildari : Adaptive Action Selection in Autonomic Software Using Reinforcement Learning

[ 23 ]M.Salehieand,L.Tahvildari: A weighted voting mechanism for action selection problem in self-adaptive software.

[ 24 ]I.Salomie, T.Cioara, I.Anghel, M. Dinsoreanu : **RAP – A Basic Context Awareness Model**

[ 25 ] REFERINTA LICENTA GEORGIANA

[ 26 ] S.C.Patrasciuc : An Agent-Based Context Management Solution (1), diploma thesis

[ 54 ] Nash Bargaining Game, <http://en.wikipedia.org/wiki/Nash_bargaining_game>

[ 27 ] Distributed Systems Research Laboratory, <http://dsrl.coned.utcluj.ro/>

[ 28 ] **Protégé**, <http://protege.stanford.edu/>

[ 29 ] Protégé Owl Loader

[ 30 ] SWRL, <http://www.w3.org/Submission/SWRL/>

[ 31 ] Pellet, <http://clarkparsia.com/pellet/>

[ 32 ] Jena Reference

[ 33 ] JADE**,** <http://jade.tilab.com/>

[ 34 ] **JADE A White Paper,** <http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf>

[ 35 ] FIPA, <http://www.fipa.org/>

[ 36 ] ASP.NET Web Services

[ 37 ] Java

[ 38 ]C#

[ 39 ] IText , <http://itextpdf.com/>

[ 40 ] X3D, <http://www.web3d.org/x3d/>

[ 41 ] XJ3D, <http://www.xj3d.org/>

[ 42]HTTP, <http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol>

[ 43 ] JFuzzyLogic, <http://jfuzzylogic.sourceforge.net/html/index.html>

[ 44 ] FCL, <http://www.fuzzytech.com/binaries/ieccd1.pdf>

[ 45 ] Factory Method Design Pattern, <http://en.wikipedia.org/wiki/Factory_method_pattern>

[ 46 ] Windows Server 2008 R2, <http://www.microsoft.com/windowsserver2008/en/us/default.aspx>

[ 47 ]Windows Storage Server 2008 , <http://www.microsoft.com/windowsserver2008/en/us/wss08.aspx>

[ 48 ]Microsoft Hyper-V Server, <http://www.microsoft.com/hyper-v-server/en/us/default.aspx>

[ 49 ] Windows Management Instrumentation, <http://msdn.microsoft.com/en-us/library/aa394582(v=VS.85).aspx>

[ 50 ]Microsoft IIS, <http://www.iis.net/>

[ 51 ] WIN32 WMI Classes, <http://msdn.microsoft.com/en-us/library/aa394084(VS.85).aspx>

[ 52 ] Hyper-V WMI Provider, <http://msdn.microsoft.com/en-us/library/cc136992(VS.85).aspx>

[ 53 ] Microsoft Windows XP, <http://www.microsoft.com/windows/windows-xp/default.aspx>