INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Université d'Ottawa · University of Ottawa

Design and Implementation of a Policy Management in an Agent-Based System

By
Mouhcine Guennoun

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

M.A.Sc.
in

Electrical Engineering

Ottawa Carleton Institute for Electrical & Computer Engineering
School of Information and Technology Engineering (SITE)
University of Ottawa

December, 2000

Table of Contents

Table of Figures

Abstract

Personal Mobility Management System, a distributed application developed at the Multimedia & Mobile Agent Research Laboratory, allows users to move across a virtual network of predefined sites. Each site provides personalized services to its visitors.

The purpose of this thesis is to introduce an agent-based architecture to create and manage policies on each site involved in the virtual network. The architecture defines an administrative infrastructure that allows each site to restrict access to the local resources and to sensitive information by visiting users. These restrictions are expressed in terms of policies that are stored in a policy server at each site. Policies are also related to the privacy and ownership levels attached to resources as well as to profiles of visiting users. The thesis presents a model to detect conflicts that may occur between different agents as a result of rules imposed by different policies. These conflicts are resolved through negotiation between agents.

Acknowledgment

I would like to extend special appreciation to my thesis supervisor Professor Ahmed Karmouch, for the time and effort he had invested, and his continued guidance and support throughout this thesis work.

I also want to specially thank Tom Gray and Serge Mankovski from Mitel Corporation, Hamid Harroud, Research Assistant in MMARL, for theirs comments and criticisms that were very helpful for the achievement of this work.

Thanks are also due to the Multimedia & Mobile Agents Research Laboratory members for their help and provision of information.

A major thanks goes to my parents for their love and dedication. Also, I would like to thank my family and my friends, for the assistance and understanding during all these years of education, without which this work wouldn't have been possible.

Acronyms

ACL: Agent Communication Language.

AEE: Agent Execution Environment.

AI: Artificial Intelligence.

API: Application Programming Interface.

ARPA: Advanced Research Projects Agency.

CNP: Contract Net Protocol.

COPS: Common Open Policy Service.

IETF: Internet Engineering Task Force.

JESS: Java Expert System Shell.

JVM: Java Virtual Machine.

KIF: Knowledge Interchange Format.

KQML: Knowledge Query Manipulation Language.

KSE: Knowledge Sharing Effort.

LDAP: Lightweight Directory Access Protocol.

MAS: Multi Agent System.

PDP: Policy Decision Point.

PEP: Policy Enforcement Point.

PMMS: Personal Mobility Management System.

SNMP: Simple Network Management Protocol.

UMTS: Universal Mobile Telecommunication System.

VHE: Virtual Home Environment.

XML: eXtensible Markup Language.

Chapter 1

Introduction

1.1 Overview

Researchers have utilized the principles of multi-agent architectures in designing distributed and federated multi-enterprise systems. One example of such a system is the Personal Mobility Management System designed and implemented by the Mobile Agent Alliance, a collaborative consortium comprising the University of Ottawa, Mitel Corporation and the National Research Council of Canada. The Personal Mobility Management System (PMMS) provides personalized services and access to resources for nomadic users over a network [33]. It provides a nomadic user with a working environment similar to her home environment. In the home site, a user has a profile that describes her preferences: the services she uses, the quality of service required, the cost she is willing to pay for certain billable services, etc. A dynamic mapping between the user profile and the local policies on each site determines a set of services that a visiting site can offer to its guest. A nomadic user can be requested to pay a cost for services such as long distance calling. For example, in the instance in which the visiting site is a hotel, which has made arrangements with a guest's employers having regard to access to hotel facilities, costs that the guest incurs in the use of services can be billed directly to the employer. Thus, it is desirable for the employer to be able to institute policies on which billable services are allowable. Hence, agent negotiation techniques are used between the home site and the visiting site in order to agree on the cost or to find a service that can fit the requirements of the user with a lower cost. Negotiation is also used for resource reservation when a user starts to use a service on a visiting site.

1.2 Objectives

The use of policies in a multi-enterprise agent based systems is required to maintain privacy, accountability of proper usage of communication resources, maximize the availability of communication access for users, and provide security definitions for accessing an organization services. Very little research has been undertaken to date in the area of integrating policies and agents together [40]. With the use of agents, we believe that real automated and policy driven management can be accomplished. Policies can be used to monitor agents' behavior in specific situations and to ensure that such behavior is consistent with the global strategy of the system.

The present work aims to develop a policy platform for the control of multi-enterprise agent-based systems. Policies are defined to regulate the actions of software agents in the Inter-Sites Personal Mobility Management System [33], which is a system designed to provide personalized services and access to resources for nomadic users over a network.

1.3 Main Contributions

The main contribution of this thesis is the design and implementation of a multi-agent system for managing system policies on a site within a virtual network. System policies include authorization policies, which are actions an agent is authorized to carry out, and obligation policies for specifying actions that an agent is responsible for performing, comprising:

- A policy server for receiving and downloading obligation policies into agents representing system services and devices, for distributing authorization policies to

authorization server, and for managing system policies in accordance with changes in system modes.

- An authorization server operating in accordance with authorization policies for receiving and authenticating requests from agents. The server processes the request to decide whether the requester agent is permitted to execute a specific action.
- An event server for enabling shared communication between multiple agents.
- A mechanism to detect conflicts that may occur between agents as result of conflicting policies.
- A negotiation model to resolve conflicts.

The validity of the architectural design is demonstrated in the Personal Mobility Management System [33], which aims to provide a nomadic user with a working environment similar to her home environment.

1.4 Thesis Outline

The rest of this thesis is structured as follows. Chapter 2 introduces the agent technology, gives an overview of different concepts of policies in the literature, and presents the policy framework developed by the Internet Engineering Task Force. Chapter 3 introduces the Personal Mobility Management System, describes the architecture of the policy service and the elements involved to setup and enforce policies within each site of a virtual network. Chapter 4 illustrates the application of the policy architecture to efficiently monitor a multi-agent system such as the PMMS system. Chapter 5 describes the implementation of different modules introduced in the design section in Chapter 3. Finally, Chapter 6 presents the summary and conclusions of this work, and provides suggestions for future work.

Chapter 2

Agents & Policies

2.1 Agent Technology

2.1.1. Definition

The term 'agent' has its background in the early work on Artificial Intelligence (AI) when researchers focused on trying to create artificial entities which can mimic human abilities [4], hence, earlier definitions of agents have been more oriented toward AI. In [5], Russel defined agents as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors." Based on this definition, each program is an agent. Thus, to make clear distinction between programs and agents, some notions of environment, sensing, and acting should be restricted. In [6], Maes defines autonomous agents as "computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed." Therefore, it restricts the agent environment to be complex and dynamic. In Maes' definition, the agents are goal oriented and have the autonomy to decide how to realize the tasks they are designed to complete. Other definitions of agents have completed the previous ones by adding new properties. Consequently, agents must perform their actions with some level of proa-ctivity and reactivity [7]. In particular, agent does not simply act in response to the environment.

Agents should be able to interact with each other and manage conflicts that may occur between them as a result of these interactions. This property has been described in literature as social ability [8]. From all these definitions, we believe that an agent is an autonomous computational entity that acts on behalf of another entity (person/agent) and exhibits some level of the key attributes of proactivity, reactivity, learning, and co-operation. Agents are software systems that reside in computers and networks, assist users with computer-based tasks. Computers are now more ubiquitous and complex than before, therefore, the gap between sophisticated computers and untrained users is becoming more apparent. Agents can reduce this gap by assisting users to complete and monitor the new generation of software. They are designed to learn the user's interests and to act autonomously on their behalf. Hence, people will be engaged in a cooperative process in which both human and computer initiate communications, monitor events and perform tasks to meet the user's goal.

2.1.3 Adaptive User Agents

One important category of agents that is relevant to this work is the adaptive user agents. Such agents try to modify their behavior to maximize the productivity of the current users interactions with the system. They assist the user to communicate her tasks to the system by presenting a simple to use interface, which hides the actual underlying system, which may be very complex. Another interesting aspect of this category of agents is their ability to learn the user interests and represent them in a custom user profile [3]. Based on their experience with the user, the Interface Agents try to build up a profile of the user. One issue in building the user profile is gathering accurate information regarding the user's interest. Many approaches have been proposed to address this

problem. In [9], Mazur proposes the learning by observation method also known as imitative learning. Agents can learn user's interest simply by watching and copying users as they display their usual behavior. With today's growth of personal communication services, the user agents has evolved to provide integrated systems in which the user can access her personalized services independently of his location or his end device (PDA, laptop...). The Universal Mobility Telecommunication System (UMTS) [39] intends to offer users the ability to access their particular set of services from any device, anywhere at any time. In [10], an agent-based framework aims to provide and deliver "personalized services across network and terminal boundaries with the same look and feel". The framework is based on the concept of Virtual Home Environment (VHE) in which the user can be constantly presented with the same personalized services, user interface customization in whatever location or end device she is using [39].

2.1.4. Agent Communication Language

The interoperation between agents that use different languages has created a need for a universal communication language. A universal language will have the advantage of eliminating incompatibilities and national variations.

The ARPA Knowledge Sharing Effort (KSE) [42] has defined an Agent Communication Language (ACL) that satisfies the following requirements:

- Sufficiently expressive to communicate information of widely varying sorts.
- Sufficiently compact to avoid excessive growth over specialized languages.

The Knowledge Query & Manipulation Language (KQML) [11], developed by KSE, can be viewed as consisting of three layers: content, message and communication layers. The content layer specifies the actual content of the message; the message layer

specifies the set of performatives for agents' communication (e.g., tell, ask-if, subscribe, etc); and the third layer consists of the communication protocol.

A message is a KQML expression in which the arguments are expressed in Knowledge Interchange Format (KIF) [12] formed from the ACL vocabulary. The vocabulary of the language is comprised of a large set of words appropriate to common application areas. Each area is expressed with a specific ontology. For instance, geometry ontology contains vocabulary for describing three-dimensional geometry in terms of polar coordinates, rectangular coordinates, and cylindrical coordinates.

However, the ACL language has an important shortcoming, which is the lack of semantics as stated in [13]. ARCOL, a language developed by France Telecom [14], overcomes this limitation by adding a semantic layer in the ACL.

2.1.5 Multiple Agent Systems

Recent works on the field of Artificial Intelligence have investigated the development of software to simulate the capabilities of human beings, such as reasoning, natural language communication, and learning; that computational agents need to be part of the systems composed of computers and humans [15]. Such systems are called Multi-Agent Systems (MAS), and can be defined as a set of independent software agents that work together to resolve problems that can not be resolved individually [15].

As stated in [43][44], Multi-Agent Systems offer the following advantages:

- Give an alternative to centralized systems, which have many weaknesses, such as performance, reliability, security, etc.
- Permit the interoperation of different legacy systems.

- Increase the reliability of the system, which can be defined as the ability to recover from the failure of individual components, with minimal performance degradation.
- Decrease the response time of the system.
- Provide efficient organization of agents to enhance collaboration.

Coordination is a key element in the performance of MAS. It has for purpose to prevent chaos in the system; to coordinate agents with different expertise to meet the global goals of the system; and to avoid dead-locks, since in a cooperative approach, an agent may have to wait for another agent to complete its tasks before it executes its own tasks [16].

There are two approaches for addressing coordination in a MAS: direct communication and assisted coordination [16][37].

2.1.5.1 Direct Communication

In direct communication, agents handle their own coordination, without relying on a third party to facilitate their communication. One example of a protocol based on the direct communication approach is the Contract Net Protocol (CNP) [16].

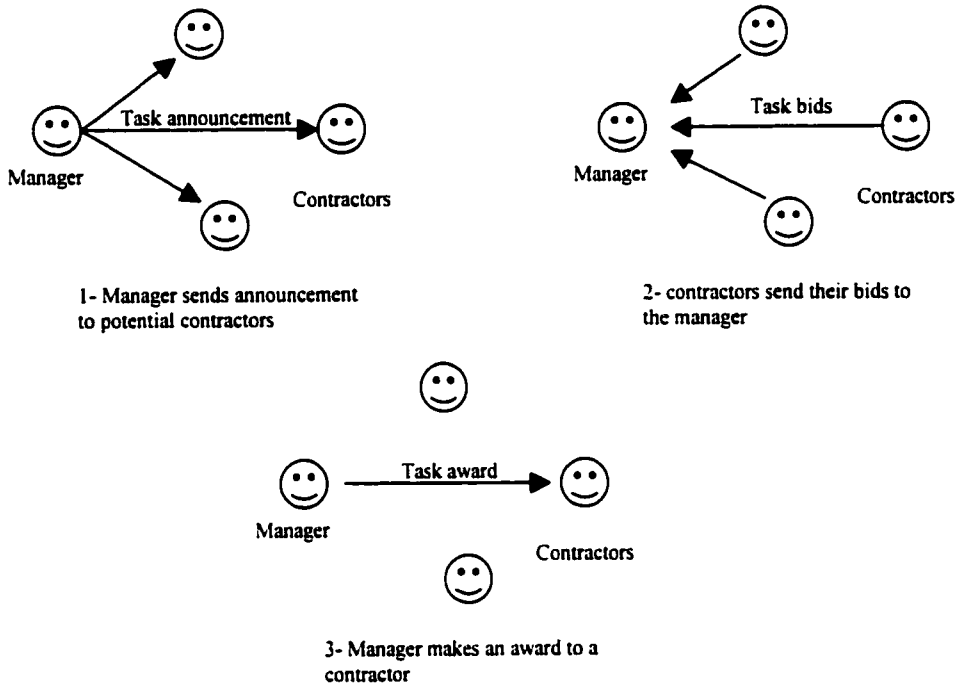


Figure 2.1: The Contract Net Protocol [16]

The protocol is in essence very simple, as it identifies two roles of agents: manager and contractor. A manager that cannot resolve a problem using local expertise has to distribute requests for proposal to other willing agents with the necessary expertise to solve the problem. Upon receipt of this announcement, the contractors submit their bids to the manager. The manager uses these bids to decide which agent has submitted the most appropriate bid, and therefore, awards it a contract. Figure 2.1 illustrates the CNP operations.

However, the CNP protocol has many disadvantages. The most obvious one is the excessive amount of communication generated. In a real world application, like an Internet-based application, it's prohibitive to broadcast and process bids of thousand of programs. Another disadvantage is that CNP assumes that there are no conflicts between

agents, and hence, doesn't detect and resolve them. The manager doesn't include its constraints in the announcement and has no possibility to bargain the bids.

2.1.5.2 Assisted Coordination

Assisted Coordination offers an alternative approach that eliminates to some extent the disadvantages of Direct Communication. In this approach, agents are organized in a federated system as illustrated in figure 2.2, in which, the communication is handled by system agents called facilitators [37].

Agents send the list of services they can offer and their needs to the Facilitator. Based on this information, facilitators build a knowledge base of system capabilities, which is used for routing future announcements. Assisted coordination has the advantage of being able to detect potential conflicts between agents plans. However, it requires that the system shares and process substantial amounts of information, and therefore, involves more computation and communication than the Direct Communication approach.

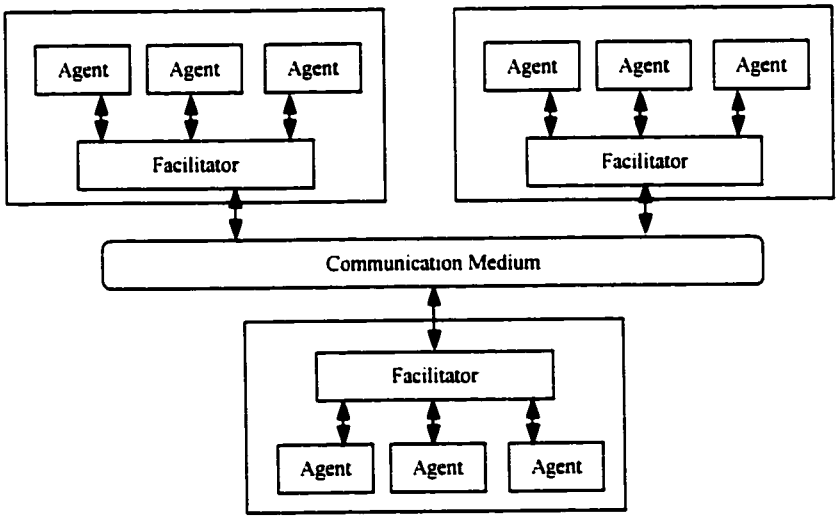


Figure 2.2: Federate System Architecture [37]

In order to resolve this issue, Assisted Coordination implementations use powerful search control techniques, developed in the Artificial Intelligence and Database communities, to enhance the processing of messages.

2.1.6. Mobile Agents

Mobile agents offer unique opportunities for structuring and implementing open distributed systems. A wide range of applications has been indicated for mobile agent technology, including electronic commerce, telecommunications services and network management [17].

Mobile agents technology was introduced to overcome the weaknesses of the traditional client/server model which are: the network overhead caused by the communication between client and the server, the latency and the unavailability of the network and the rigidity of the above model, where clients can only request a predefined set of operations from a remote server [2]. The mobile agents technology tackles these problems by enabling a program, called mobile agent, to travel to the remote server to request its services. Furthermore, the server can be customized to the need of the applications, that is, each new operation that is not supported by the server can be sent as a mobile agent . Thus, server components are no more installed by the administrators but are dynamically installed by the application itself.

The General Magic's Telescript [17] technology is the first commercial implementation of the mobile agents concept. The platform provides the basis for a complete remote programming. The concepts of this technology are [41]:

- Agent: A mobile entity that can travel over the network to complete its goal.

- Travel: Lets an agent travel from one place to another to obtain a service offered remotely and then return to its starting place.
- Meeting: Allows two agents to exchange messages on the same computer.
- Connection: Enables two remote agents to open a private communication channel to share information.
- Authority: Represents the certificate of the agent. It is a way to verify the entity of an agent. It does identify an agent as a passport identifies a human.
- Permits: The set of capabilities of an agent which indicate what operations a visitor agent can request, and what resources it can use.

The technology provides a complete, object oriented, dynamic and persistent programming language to facilitate the development of mobile agents. Furthermore, an engine, called Agent Execution Environment (AEE) [41], maintains and executes places running in its environment, as well as the agents occupying those places. A communication protocol suite enables two engines to communicate in order to transport agents between them.

2.2 Policy Overview

The term "policies" may be defined as a set of rules in a domain-managed object [23]. These rules are derived from management goals and related business strategies to define the desired behavior of distributed heterogeneous systems, networks and applications. They provide means for expressing how tasks are to be undertaken and how a system is to respond to a given situation. Policies may also be defined as a binary relationship between objects [18]. Binary relations, consisting of a subject and a target, can be used to express different types of policies. Management policies may be expressed

by a human manager in a high level manner, i.e. policy language. These policies must then be transformed and refined into scripts or rules that can be interpreted by software agents [18]. One advantage of policies is that they can be expressed independently from the agents that interpret them. This separation permits modification of the policies without changing the code of the agents [23]. Hence, agents can be reused in different networks where different policies apply.

Because of the large number of policies, classification systems have been developed to give semantics to the policies and to assist in automating the process of transforming and refining of the policies. Wies [38] has proposed a template to represent policies. The attributes of this template are:

Mode: Policies can be an obligation, permission, or prohibition.

Lifetime: A short, medium, or long term application can characterize the duration of a policy.

Trigger mode: The enforcement of a policy can be constantly active, repeated periodically for a specific period time, triggered by asynchronous events or a combination of the last two.

Activity: A policy can monitor or enforce actions on its target objects or react to a new situation that occurs in the system.

Functionality of targets: This class includes policies applicable to resources with common characteristics (e.g. printers, hubs, and routers...).

Below, we give other views and definitions of policies as described in literature.

2.2.1 Deontic logic

Many approaches to represent policies using the deontic logic, logic of permissions and obligations, have been described in literature. An early one was made by Hughes and Creswell in 1968 as described in [19].

In [20], Jonsher models the access behavior of database users by using deontic logic. The work divides the user rights into two categories: access rights, and normative rights. Access rights define the traditional users access to databases, like reading, deleting, and modifying. A positive access right is permission, and negative one is a prohibition.

The normative rights are defined as the duties of a user, i.e. the actions the user is obliged to do, whereas, the liberties are the actions the user is free to do.

Duties are seen as a way of controlling the fulfillment of some critical tasks in an organization. However, a duty may not have an authorization associated with it, and hence cannot be accomplished. Therefore, a relationship is needed to be defined between the authorizations and the obligations. A simple association is to consider an obligation to execute an action is an authorization by itself, based on the axiom "obligation implies permission"[20]. A second approach is to consider them as independent. In this case, a specific permission should be defined in order that the obligation can be completed.

2.2.2 Obligation policies

Becker [21] defines an obligation policy as the management actions a manager must perform in order to keep the network operational. A policy defines a relationship between a set of managers and managed objects specified as domains instead of particular elements. A domain is a logical grouping of objects and it can be organized in a

hierarchy of domains making the system management more scalable for large systems. All objects belonging to a specific domain will have to carry the policies that apply to this particular domain. The advantage of this approach is that the creation/deletion of an object doesn't require the alteration of predefined policies.

Koch [22] defines the management tasks as a set of production rules that are represented in an expert system model. Authorization policies are represented as preconditions to rules. When the rules' conditions are satisfied, the rules' actions are executed. Like rules of an expert system, new conditions may be satisfied when these actions are performed. Triggers of obligation policies are defined using an event definition language.

2.2.3 Organizational Policies

Many researchers have recognized the role played by the goals and objectives of an organization in defining the behavior of its information systems. Wies [23] views management policies as derived from the goals of management and related business strategies. In [24], Mcbrien proposed the External Rule Language (ERL) as a way to model organizational policy. In [25], system management is defined as the set of processes that are applied to the administration and control of information systems within an organization according to a given set of policies. Policies are to refine processes that are designed to achieve management goals, define priorities to resolve conflicts among concurrent processes, and schedules the management processes. Therefore, policies are recognized to not be actions in themselves, but mechanisms used to make the decisions.

2.2.4 Policy Language

In [26], Marzullo defines a rule-based language for programming the management layer in a system. The language called, Lomita, is a set of tools for building distributed management software. It allows the configuration of system components, the scheduling of management tasks, and the monitoring of management software. Each system component is represented as an object with three kinds of attributes: properties, which represents the states of the management application; sensors, which are functions that returns the values of the application's states and environment; and actuators, which can set new values of the application's properties. The two later attributes are similar to the set/get operations defined by the Simple Network Management Protocol (SNMP) [27]. Policies are represented in the form "if condition then action". The sensors provide a way of detecting policy conditions, in order that, the actuators perform the policy actions.

2.3 IETF Policy Framework

2.3.1 Introduction

Policies provide a mean to manage large, heterogeneous and evolving systems [28]. A traditional approach to manage such systems consists of the usage of a centralized management application that alters the behavior of each component comprising the network. The management software should have explicit knowledge of the management interfaces of each managed entity, hence, the management of standard interfaces common to the majority of the components of a network [28].

The Internet Engineering Task Force (IETF) Policy Framework [28] represents an alternative approach of network management. Instead of centralizing the network

management into a single software entity, the framework centralizes the storage of predefined rules. These rules are comprised of conditions and actions that are designed to be device and vendor-independent. Hence, the Policy Rule is the common point between entities that are involved in a policy system.

2.3.2 Policy Schema

The IETF Policy Schema [29] defines an LDAP [30] schema to represent and store policies and the characteristics of devices controlled by policy rules, as well as, the interpretation of actions and conditions represented in a policy rule. The schema consists of thirteen classes organized in two class hierarchies: structural classes and relationship classes, as illustrated in figure 2.3.

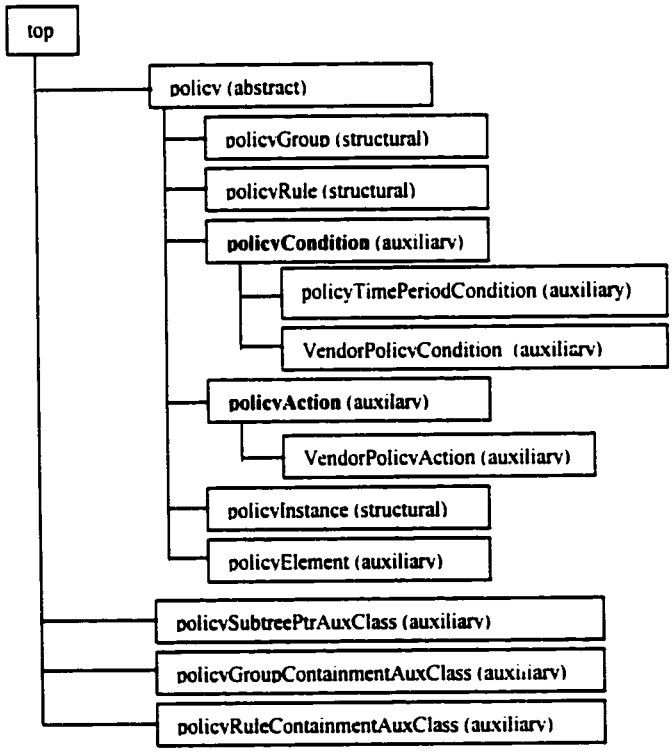


Figure 2.3: Class hierarchy in core policy schema [29]

The structural classes represent policy information and control of policies. For example, structural classes **policyAction** and **policyCondition** define the conditions and the actions of a policy.

The second type of classes represents the relationship between the instances of the structural classes. They are specific to the LDAP core schema definition. Hence, these classes are implementation-dependent and may differ if other storage technologies are used.

2.3.3 Policy Definition

In [28], Policies are defined as a set of rules, and rules are a set of conditions and actions. Policies can contain other policies in a nesting fashion, or reference a group of rules, but not both. The Policy Schema [29] defines two relationship classes: **policyGroupContainmentAuxClass** and **policyRuleContainmentAuxClass**, to represent the nesting relationship between policies and rules. Figure 2.4 illustrates the nesting relationship.

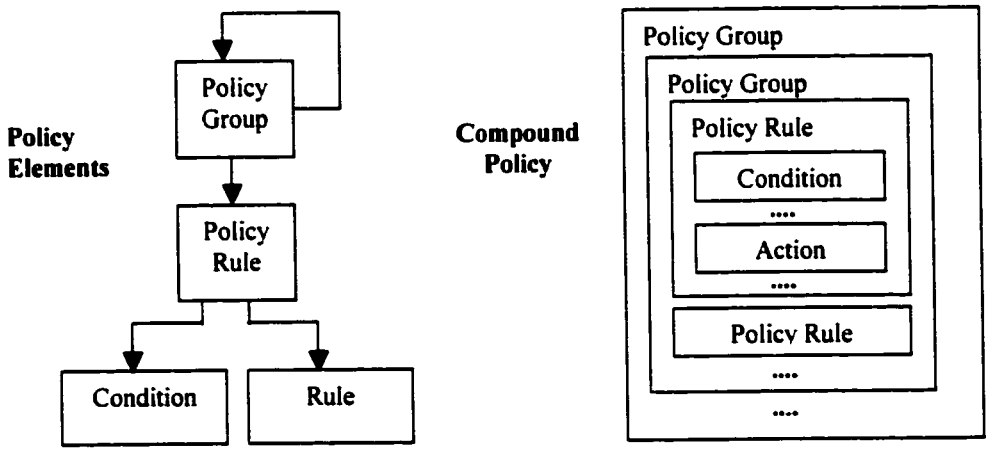


Figure 2.4: Composition of Policies [29]

Policy rule classes are structured in a hierarchical way similar to the structure of tables in SNMP's structured management format in SMI. Also, policies are specified in terms of interface functionalities, rather than, being specified explicitly for each network device.

2.3.4 Policy Architecture

The framework is comprised of two main components: the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP) [28]. The PDP retrieves policies from a policy repository using the LDAP protocol or other access protocols, detects policy conflicts, translates them in a form understood by the PEPs, and then stores them in a local Policy Information Base (PIB) [31]. It sends policy elements to the PEPs based on theirs requests, or updates of policies from external entities.

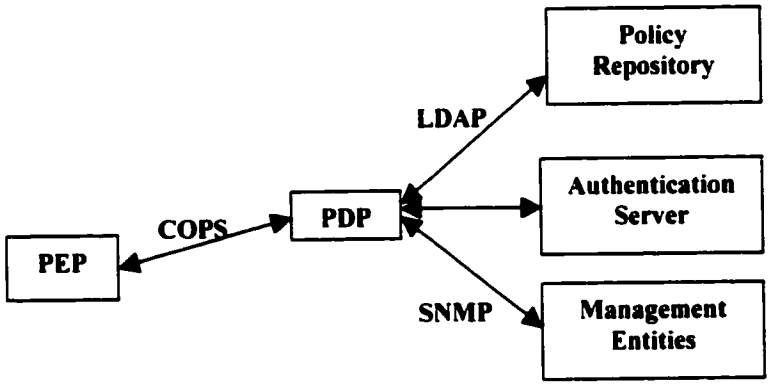


Figure 2.5: IETF Policy Framework Architecture [29]

PEP is an interface to one or more network device. It informs the Decision Point about its interfaces, and applies actions according to the PDP decisions. The separation between the PDPs and PEPs is a logical separation based on functionality. The PDP can be local to the PEP so the policy decision can be made at the level of the PEP.

PEP and PDP exchange their information using the COPS [32] protocol. The two components have to examine periodically the network elements to check whether the policies are satisfied.

2.3.5 Policy Storage & Retrieval

The framework defines a centralized repository that monitors information storage and retrieval [28]. Efficient retrieval is a key element in the performance of the decision process. For that reason, the PDP needs to retrieve, in real time, relevant policies for a specific request from a PEP. The retrieval must be done with minimum requests and has to extract the exact policy information, no more or less. The Core Policy Schema defines simple rules designed for efficient retrieval, and complex rules are designed for data efficiency, that is, allows maximum reusability of the rules. In a simple policy rule, conditions and actions are part of the rule itself, whereas, in a complex one they are part of the instances of the structural class. The policy directory can be a directory accessed using the LDAP protocol. However, LDAP cannot maintain consistency in large and dynamic systems. For this reason many policy management systems use database systems for their transactional operations.

2.4 Chapter Summary

This chapter gave an overview of the agent technology, and policy management definitions. The IETF Policy framework is an ongoing work that has the prospective to become a widely used standard in the future.

In this work, we use the assisted coordination approach to avoid the disadvantages of the direct communication approach. The Facilitator agent is a software layer on top of MicMac, which ensures communication and coordination between agents. Our policy definition and management is similar to some extent to the IETF Policy framework. We have focused on the definition of policies to manage agents rather than network entities. Policies are managed through a relational database and the storage representation doesn't follow the IETF Policy Framework LDAP Schema [29].

Chapter 3

Policy Management: Design & Concepts

3.1. Introduction

We present in this chapter an architecture of multiple agents for setting up and enforcing policies within each site of a virtual network. A policy server represents the global policies of the site and each agent manages its own policies. Policies are dynamically downloaded from the policy server into agents that carry the responsibility to enforce them. Agents propagate their policies to the policy server to detect any conflict that may rise between agents during dynamic mapping and resource reservation. A negotiation mechanism is provided to resolve such conflicts. An authorization-based mechanism is also provided such that agents must request authorization before performing any action, in response to which a ticket is delivered to the requesting agent for accountability and security reasons.

3.2. Personal Mobility Management System

3.2.1. Introduction

The development in the area of wireless data communications and mobile computers enables to a great extent mobile computing. Nowadays, the environment of mobile computing has improved considerably in terms of bandwidth, delay, computing power, and quality of display. However, mobile devices range from a very low performance equipment, e.g., personal digital assistants (PDAs), pagers, etc., up to very

high performance laptop PCs, causing a growing need for service adaptability. For example, PDAs are not designed to display high quality images, however, nomadic users will be charged for data that their equipments are unable to support. Hence, many nomadic computing platforms have been developed to address these issues and to provide the end-user with a rich set of computing and communication services in a transparent and integrated fashion [35]. Some of these systems use personal agents to represent the user's communication needs and help in the manipulation of this information to a mobile user. These personal agents provide the end user with the information about expected performance, control over the transfer operations using monitoring policies.

Such systems aim to provide an open, intelligent, secure frameworks for providing access to legacy systems, the Internet, and partners' networks that extends the enterprise network beyond existing organizational boundaries [35]. They provide manageable and secure infrastructure, along with essential applications and services, that allow nomadic users to access information across interconnected networks, while roaming from one location to another [39]. Based on the premise that the organizations manage people and their access privileges, a nomadic computing platform enables organizations to extend their network beyond traditional boundaries in a secure and manageable manner to encompass employees or associates anywhere, and build enterprises to enable mobile computing and virtual workgroups.

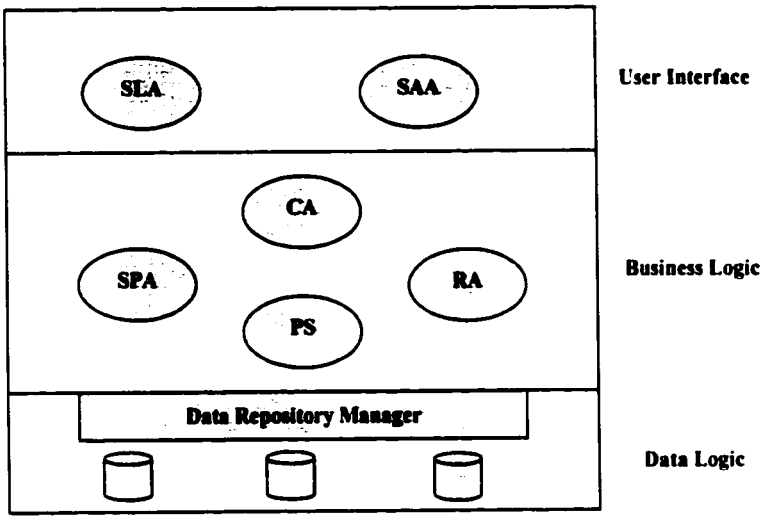
An example of a nomadic computing platform is the Personal Mobility Management System (PMMS) developed by the Multimedia and Mobile Agent Laboratory [33]. It aims to provide an integrated platform for mobile users to access different legacy systems while roaming into different locations.

3.2.2 Architecture

Personal Mobility Management System (PMMS) provides personalized services and access to resources for nomadic users over the network [33]. The application aims to offer a nomadic user a working environment similar to her home environment. In the home site, a user has a profile that describes her preferences: the services she uses, the quality of service required, the cost she is willing to pay to use some billing services, etc. A dynamic mapping between the user profile and the local policies on each site determines a set of services that a visiting site can offer to its guest. A nomadic user may be requested to pay a certain cost to use some services. In this case, agent negotiation techniques is required between the home site and the visiting site to agree on the cost or to find a service that can fit the requirements of the user with a lower cost. Negotiation is also used for resource reservation before a user starts a service on visiting site.

A prototype [33] of the system has been implemented on three sites namely: University of Ottawa, Mitel corp. and National Research Council of Canada.

As shown in Figure 3.1, the PMMS architecture is organized in three layers. The first layer is the data layer that contains all the data information of a site, such as, users' profiles, services' description, and system policies.



SLA: Site Logon Agent.
SAA: Service Access Agent.
CA: Coordinator Agent.
SPA: Site Profile Agent.
PS: Policy Server.
RA: Resource Agent.

Figure 3.1: PMMS Architecture

The second layer is comprised of four agents implementing the logic of the system. The coordinator agent manages the inter-agent communications. It's responsible to monitor agents' communication by establishing, closing, suspending, or activating communications between two agents.

The site profile agent negotiates with its remote peer (at the user's home site), the user preferences, and maps them on the local policies. The result of the mapping is a set of services that a site can offer to its visitors. Resource agent manages system resources and is responsible for the resource reservation. Moreover, the system contains a set of agents each one of them representing a service or a device. Each one of these agents reserves necessary resources for the execution of the service it represents.

The third layer defines the user interface to the system. It's comprised of two agents: User Logon Agent and Service Access Agent. The User Logon Agent is responsible for the

authentication of the user, her localization on the virtual network, and opening a new session for her. The Service Access Agent displays to the user the authorized services in the visited site and invokes the service agent when the user requests the service.

3.2.3. Agent Communication

Agents communicate using a reduced set of performatives of the Knowledge Query Manipulation Language (KQML) [11]. Messages are transported using MicMac [34], a product of Mitel Corporation. MicMac is a set of software tools, which provides a mean for communication among multiple, distributed agents. The mechanism of communication is based on the Blackboard paradigm. A tuple space, called agora in MicMac terminology, is an open area for exchanging information between agents. In PMMS, a tuple is created by the coordinator agent and can be used by any number of agents for their communication. Private agoras can also be created for private agents' communication. MicMac implements a shared memory, which stores information as tuples. Tuples are defined as unordered set of fields represented as "key-value" associations. It defines the following primitives to interact with shared memory:

- Post: Adds a new tuple to the shared memory.
- Pick: Reads and deletes a tuple from the shared memory.
- Peek: Reads a tuple from the shared memory.
- Cancel: Terminates a pending pick/peek request.

3.2.4 Policies in PMMS

A site accessed by users belonging to different organizations may be vulnerable if there is no rules to monitor their work sessions. Users always want better services, but are not always willing to pay the higher costs involved. Hence, there is a need to authenticate those that request site services and determine if they are eligible to use them or not. In [35], policies are defined as regulations that should facilitate the accountability of proper usage of communication resources, maximize the availability of communication access for users, provide security definitions for accessing a device and its corresponding services, and facilitate the creation of varied secure regions within an organization. The solution described in [35] for including policies in the system is to add certain attributes to the site related data that are stored on a Lightweight Directory Access Protocol (LDAP) [30] directory of each site. These attributes describe the services available in the site and the restricted areas that can only be assigned to privileged users. However, this approach suffers from a number of weaknesses. Firstly, no standard representation of policies is provided. Secondly, the policies are dependent on the system and co-located with the description of the system. Also, the policies are not classified according to the activities they monitor in order to facilitate their enforcement into the system. Furthermore, no mechanisms are provided to detect and resolve conflicts that are inevitable between enterprises.

The work of this thesis aims to provide a mean to define policies for each site involved in the virtual network. It provides the components to decide when they apply and how to enforce them. Policies are used to dynamically map the users' profiles and services offered by a site, to reserve resources necessary for the execution of the services

requested by the mobile users, and to monitor the actions of agents during their executions.

3.2. Policy Management

3.2.1 System Architecture

We describe in this section a policy management architecture to define and manage policies in a co-operative system, so that agent operations conform to the business goals of the organization that operates each site.

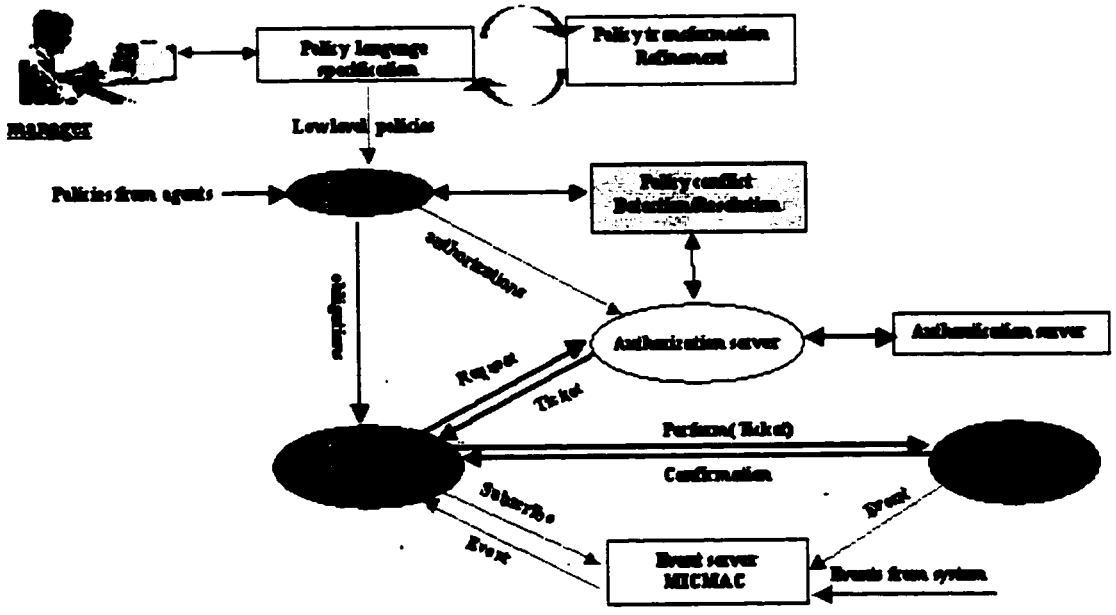


Figure 3.2: Site Policy Architecture

Agents are autonomous entities that behave according to theirs obligations with respect to the overall policies in the system. Policies are divided into two types: obligations and authorizations [18]. Obligations specify the motivations or tasks an agent is responsible to perform as part of its role. Authorizations represent the laws of the system and determine whether an agent is allowed to perform a particular action, or not.

Agents can interchangeably play two roles: requester or executor. A requester agent is an agent that requests another agent, i.e. an executor agent, to perform an action in order to fulfil one of its obligations. The executor is an agent that is designed to perform a set of actions as a part of the global role it plays in the system. For example, a printer manager agent is designed to execute printing actions, whereas, a word application agent may request the printer manager agent to print a document. Eventually, an agent could also be the requester and the executor of the same action. The division of a task into a request action and an execution action is logical division in order to ensure that no action will be performed unless it's authorized by the system policies. Hence, policies determine whether the executor should perform the action for a requester agent.

3.2.2 Policy Rules

Policy Rules are a set of actions to be initiated when a specific set of conditions are satisfied. They provide administrators the ability to manage the behavior of agents in each one of the sites involved in the virtual network. They are usually produced by the site administrators or by agents themselves and stored in the policy repository. The policy repository may be implemented as an LDAP server [30] or relational database, or any other data management system. In some cases, conflict detection and resolution may be first performed before storing the policy in the repository.

A policy rule may take one of the two following forms: authorization or an obligation.

3.2.2.1 Authorization

An authorization is a relationship between an agent, called requester, and an action of another agent called executor. Each authorization is an object with the following attributes:

Id: Uniquely identifies the authorization in the system.

Mode: The mode of an authorization can be either permission or an interdiction.

Subject: specifies the agent, also called requeter, to which the authorization applies. The subject may be specified as a role rather than an individual agent. Permission allows the subject to request that the executor agent performs an action. An interdiction prohibits the execution of the action.

Action: specifies a set of operations that the requester agent is authorized or prohibited to perform depending on the mode of the authorization. The set of operations associated with an authorization may be ordered or unordered.

Target: specifies the agent, also called executor, which executes the action. As for the subject attribute, the target attribute may specify a role rather an individual agent.

Constraint: The applicability of an authorization can be limited using a constraint. The constraint determines the conditions that must be verified in order that the authorization can be granted.

Status: A policy can have two status, enabled or disabled. Enabled policies should be enforced whenever the conditions of the constraint are satisfied. Disabled policies are not to be enforced while they are in the disabled status.

Priority: specifies the scope of the policy. A policy, with a high priority, overrides low priority policies. The priority of a policy will determine it's precedence when more than one policy is initiated by the same constraint.

Class: classifies policies according to the activity they monitor in the system. Examples of classes are security, accounting, and admission control.

System mode: A system mode represents a specific state of the system. Policies belonging to a system mode define the strategy of the system in this mode.

Creator: specifies the agent or the human manager that adds the policy to the system.

For example, to express an authorization to monitor resource reservation that may be done by a video agent providing a video service over the network, the system manager may add the following policy:

Id: A0001

Mode: Permission

Subject: Video Agent

Action: Reserve Bandwidth

Target: Resource Agent

Constraint: (Reserved Bandwidth < 10% of System Bandwidth && Cost > 12/unit)

Status: Enabled

Priority: 5

Class: Resource Reservation / Bandwidth Reservation

System Mode: Normal Resource Mode

Creator: System Manager

The Video Agent is only able to reserve 10 percent of the available bandwidth in the system and have to pay at least 12 units for each unit of the reserved bandwidth.

A label **any** can be used instead of a specific requester, action or executor. A default authorization can be set to determine the default behavior of the system. The default authorization can be:

Permission any any any: permits to any agent to request any action from any other agent in the absence of a specific interdiction.

Or

Interdiction any any any: Prohibits any agent from requesting any action from any other agent in the absence of a specific permission.

3.2.2.1 Obligations

An obligation is a duty an agent should perform once a set of conditions are satisfied. The duty can be represented as a set of ordered or unordered actions. The obligation may specify the agent that should be requested to execute the action or leave it up to the responsible agent to look for the suitable executor agent. However, the second option implies that the system has a resource discovery mechanism that allows the agents to report and knows each others capabilities. In this work, we use a facilitator agent to store all agents' actions, and acts as a broker between requesters and executors. We represent an obligation as an object with the following attributes:

Trigger: an internal or external event that triggers the obligation. The event can be a result of an action performed by another agent or a specific time, etc.

Subject: specifies the agent responsible of the execution of the obligation when it's initiated.

Action, target, constraints, class, and system mode: These attributes have the same meaning as for the authorization's attributes.

Exception: specifies an action that the subject must perform in the case it fails to accomplish the actions specified by the obligation.

Obligations and authorizations policies represent the global strategy of a site. They control behavior of the agents belonging to the site and determine how agents should react to any changes in the system. They are classified according to the activity they monitor. Four classes of policies are defined for the implementation of this work: security, admission control, user profile, and resource reservation. These classes of policies are used by the coordinator agent, site logon agent, site profile agent, and resource agent, respectively.

During its life cycle a policy may have different status depending on the requirements of the system. Hence, a policy may be enabled, that is, should be enforced whenever the conditions making its constraints are satisfied. In the contrary, a disabled policy should not be processed while it's in the disabled status.

4.2.3 Policy Editor

The policy editor is an interface to the policy system that enables the editing of a policy. Policy editing includes viewing, entering, modifying and deleting a policy rule. It may be implemented as a graphical user interface, preferred nowadays by most system administrators, or/and as a scripting interface. Policy rules may be expressed in a user friendly language. Hence, they may need to be transformed to a specific format that can be understood by the system agents. The policy translation is carried out by a policy

translation and refinement module, which can be seen as an intermediate layer between the human management and agent management. In fact, the policy translation has for purpose the mapping of the high level policy specified by human administrators to an information schema particular to each system. Thus, the refinement process may result in one or more low level policies depending on the complexity of the policy language specification. For example, an administrator who wants to secure the system may add a policy that prevents foreign users from accessing the system. However, in a big system, it's most likely that the administrator doesn't know all the specific agents carrying the responsibility to protect the system nor the actions to be executed to perform such a goal. Hence, the added policy can be expressed in a high level language such as "put system security to high". A refinement of such a policy may result in more specific policies, i.e. low-level policies, with deterministic actions that should be enforced by system agents.

3.2.4 Policy Validation & Conflict Resolution

The resulting low-level policies of the refinement process should be checked in order to not conflict with existing rules. Likely, a new entered policy stating that no user can access the system during a certain period of time, will conflict with an existing policy allowing some privileged users to access the system anytime. Hence, some mechanisms are required to detect and eventually resolve such conflicts. An obvious mechanism is to set priorities between policies, that is, a policy with a high priority will have precedence on low priority policies. A similar approach is used in [3] to resolve conflicts. In the case that two policies with the same priority conflict, precedence based on the scope of the policy may be used. For example, if a policy P1 is a subset of policy P2, i.e. P1 is always initiated whenever P2 is initiated, then policy P1 should have precedence on policy P2.

Using this conflict resolution mechanism, administrators will be able to specify a general policy denying access to all users during a certain period of time of the day, while another policy may allows some privileged users to access the system during the period prohibited by the general policy. These conflict detection rules can also be expressed as meta policies, which are policies that apply for other policies. In the case that the system cannot automatically resolve the conflict the new added policy should be rejected and a notification should be sent to the entity entering the rejected policy, whether it's a human administrator or a system agent.

However, policies may be initiated based on a specific state of the system that may not be known during the insertion of the new policy. In this case, the conflict may raise at the time of enforcement of actions specified by the policy. Hence, a dynamic conflict detection mechanism has to be implemented. Also, more elaborate conflict resolution mechanisms are needed to resolve these dynamic conflicts. Agent negotiation is one of such mechanisms that may be used, that is, agents will try to reach an agreement in order to resolve conflicts.

3.2.5 System Agents

As defined before, agents are autonomous entities behaving to fulfill their obligations. Each agent has set of defined roles in the system. For example, agents may be an interface to a device such as a router or a printer, or acts a resource broker reserving system resources for other agents. By this mean, a policy is specified for a particular role instead of an individual agent. It is a powerful mechanism that enables generation of site-wide policies independently on the existing agents in each site. Each agent should report to the policy server the roles it plays in the system. For example, if an agent is

responsible of user authentication and data encryption, then it has to report these two roles to the policy server, so the latter can distribute to this agent any obligation that applies to these two roles.

An agent can be seen as an aggregation of two different modules: a code that represents the actions the agent is designed to execute and policies that control the behavior of the agent. Thus, policies can be modified in order to change the behavior of the agent without changing the code of the agent. Furthermore, agents can be reused in different sites where different policies apply. This property is useful in the case of mobile agents, i.e. agents able to move across different network boundaries. An agent that moves from a site to another site will receive its policies from the new policy sever without need to change its code. Hence, it will continue to perform its roles according to the new site goal.

Each agent has a set of obligations it is responsible to fulfill and the associated policies will determine whether or not the agent can accomplish these obligations. When the conditions of an obligation apply, the agent responsible should perform the set of actions representing the duty of the obligation.

In PMMS system, policies represent the knowledge of the agents because they instruct them what are theirs duties, when to perform them and whether they are authorized to perform them or not.

Since agents have to collaborate to perform theirs tasks, there is a need to identify agents in the system by one or more identifiers in order facilitate communication between agents. Also, they can be assigned a group identifier depending on the roles they play in the system in order to allow for message broadcasting. For example, instead that the

policy server distributes a security policy to each agent responsible of security enforcement, it may send one message with the security identifier as receiver.

3.2.6 Event Server

A common behavior to all agents is that they are designed to report any event they internally produce, and subscribe to be notified when a new event occurs. The event server provides the mechanisms that allow agents to report and subscribe to events. It has as role to collect events from agents and eventually the system and notifies subscribed agents to these events. For example, an email forwarder agent might be interested in the email-received event. Each time an email is received, the email server will notify the event server. The latter will dispatch the event to all agents that are subscribed to it. The email forwarder agent could then accomplish one of its obligations, which consists of forwarding the received email to the current location of the receiver when she is away from the home site.

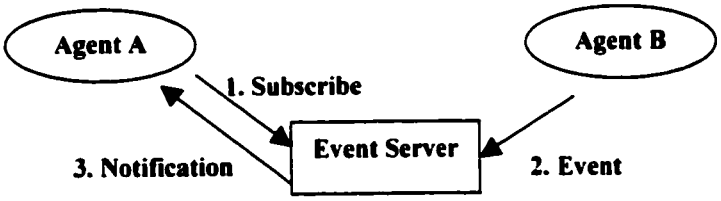


Figure 3.3: Event Notification Mechanism

Figure 3.3 shows the mechanism of events notification. In step 1, agent A subscribes to an event it needs to be notified when it occurs. The event producer, in the

figure Agent B, notifies the event server when it produces the event. The event server in step 3 dispatches it to all subscribed agents, for example agent A.

The event mechanism described above avoids the overhead generated by the polling mechanism. In a polling mechanism, agents have to periodically poll the event producers in order to track the occurrence of the event on the system.

The event server used in the implementation of this work is the product of our partner MITEL called MICMAC. MICMAC implements the blackboard paradigm for the creation of shared communication space called agora or tuple spaces. A tuple space is an open area for exchanging information [34]. It also provides the low level communication medium to the agents by enabling the creation of private and public communication channels.

3.2.7 Facilitator Agent

The Facilitator Agent is a yellow-pages that describes the services that can be provided by agents. Agents register their services with the Facilitator Agent, and can find out what services are provided by other agents on specific domains. It should keep an accurate, complete and timely list of the most current information about agents and their services in its directory. Agents need a service discovery mechanism in order to be able to request a new service from other agents. For example, if an agent would like to initiate a long distance call on behalf of the user, it needs to use the Facilitator Agent to discover the set of agents representing different carriers like Bell Canada, AT&T, Sprint Canada, etc. Hence, an agent can discover and potentially choose the cheapest, most reliable carrier able to make a particular long distance call.

3.2.8 Policy Server

The policy server is the main component of the system. It receives policies from agents or from human managers via interfaces. Obligation policies are downloaded into agents that are responsible to perform them. Authorization policies are distributed to an authorization server, which gives agents necessary authorizations to perform a set of actions. The policy server is responsible for the management of policies' status. Each policy can be, during its life cycle, enabled, disabled or deleted. Before accepting a policy, server ensures that the new policy doesn't conflict with existing policies in the system, referred as called static conflict detection. Run time conflict detection is also used to detect conflicts that may rise after a policy is enforced in the system. It's described in greater detail in chapter 4.

Policy server is notified by agents when any change in the system occurs. It reacts to these changes by enabling or disabling policies that already exist in the system, or distributing new policies. Thus, the system behavior is adapted to meet the requirements of new system states.

3.2.8.1 Notification of Agent Roles

Since agents are autonomous entities, they can change their roles without referring to the policy server. Hence, they must report to the policy server the roles they are playing on the system as part of their obligations. An agent may be responsible of several tasks and the policy server has to know about these roles in order to be able to distribute any new set of policies to the right agents that should enforce them in the system. Role notification can also be done upon request of the server as an auditing

operation in order to ensure the integrity of the system. The underlying protocol for role notification is based on two operations: Notify_Role, and Get_Roles.

- **Notify_Role(Agent_Id, Roles)** : By this operation, any agent can send to the policy server the set of roles it plays in the system. Agent_Id is a unique identifier of the agent, and Roles is the set of agent's roles. For example, an Email Agent responsible of sending and receiving users' emails may notify the server of its roles by sending the following message:

  Notify_Role("Email_Agent","Receive_Email,Send_Email")

- **Get_Role(Agent_Id)**: This operation is used by the policy server to know the roles of a specific agent identified by Agent_Id. For example, the policy server may send the following message to learn the roles of Email Agent: Get_Roles("Email_Agent"). The email agent replies by a Notify_Roles message. A broadcast message can also be sent by setting a group identifier instead of an agent identifier.

3.2.8.2 Policy Distribution

As described earlier in section 3.2.2, policies can be either an agent obligation, or a system authorization. System authorizations are automatically distributed to the authorization server that is responsible to deliver necessary authorizations to agents when they need to request an executor agent to perform a specific action. Obligation policies are downloaded to agents based on theirs roles. Once a new obligation is added to the system, policy server determines, based on the class of the policy, to which agents the policy should be sent. A policy that puts some constraints on the time when users can log on to the system is classified as an admission control policy. It should be distributed to

any agent that has as role to control the user's access to the system. Policies can also be retrieved as part of the distribution process. They may be retrieved because of many reasons such as date expiration or a change of the management requirements.

We define three operations for policy distribution:

- **Set_Policy (Agent_Id, Agent_Role, Policy)**: the policy server uses this operation to distribute any new Policy *p* to the agent identified by the Agent_Id. The role of agent to which the obligation apply is set by the Agent_Role parameter.
- **Set_Authorization (Authorization):** Used only to distribute authorizations to the authorization server.
- **Retrieve_Policy(Agent_Id, Policy_Id)**: Retrieves a policy from an agent. The policy can either be an authorization or an obligation. In the case the policy server needs to retrieve the policy from the system, an Agent Identifier referring to all agents can be specified instead of an identifier for a particular agent.

3.2.8.3 System Adaptation

The policy server is responsible to monitor the behavior of agents to meet the global system requirements. System monitoring consists of ensuring that enabled policies enforce the right behavior in each managed site. We define a set of predetermined system states, or system modes, and classify policies according to these modes. Policies belonging to a class mode are enabled when the system mode is enabled. They adapt the behavior of the system to the requirements of a new system state. For example, critical resource mode is enabled when the amount of resources available in the system is under a limit. Thus, policies belonging to this mode are enabled and will define a new resource

allocation strategy in order to prohibit agents from reserving excessive amounts of resources.

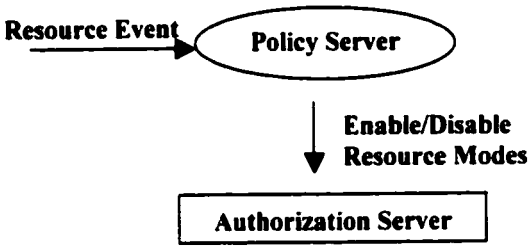


Figure 3.4: System Adaptation

Figure 3.4 illustrates the steps for the adaptation of resource modes upon the receipt of a new event from an agent responsible of the management of system resources, e.g. Resource Broker Agent. The policy server responds to this particular event by enabling policies belonging to critical resource mode and disabling policies belonging to normal resource mode.

We define two operations for enabling and disabling system modes: Enable Mode and Disable Mode.

- Enable_Mode (SM): Enables all system authorizations belonging to the system mode SM.
- Disable_Mode(SM): Disable all system authorizations belonging to the system mode SM.

We note that different system modes can be related: when a mode is enabled, other modes are automatically disabled. For example, when critical resource mode is enabled, normal resource mode is automatically disabled.

3.2.9. Authorization Server

The authorization server agent represents the laws of the system, which are expressed as authorizations. The authorization server is responsible for delivering necessary authorizations to agents requesting the execution of an action. Each agent who requests an authorization is first authenticated. Authentication is often established through third party authentication servers like Kerberos authentication server, Radius, via a certificate authority like X.509, or by sharing public and private keys. Then, the server retrieves policies that apply to the request, evaluates them and then delivers a response to the requester agent.

According to the present work, each agent must be authorized before executing any action. To illustrate how this mechanism is effected by agents requesting authorization to execute an action, an example is set forth by which a Word-Agent requests authorization to print a document using a Printer-Manager.

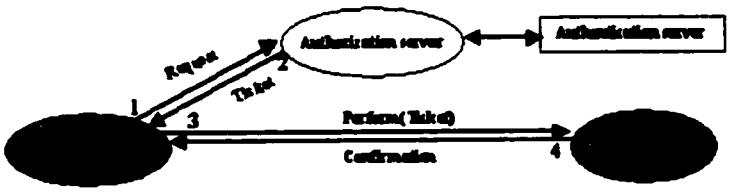


Figure 3.5: Steps of Requesting an Authorization

1- The agent requester, e.g. Word-agent, sends a request to the authorization server asking for an authorization to execute an action, e.g. print a document. The request

message includes the agent identification, the action and its parameters, and the identification of the agent executor of the action.

2- The authorization server authenticates the agent and processes the request. The authorization server then delivers a ticket that specifies the result of the processing of the request. In the case of a negative authorization, the ticket contains the reasons of failure and offers alternative choices to the requester. The requester can then modify the request to fit the conditions required by the policy server.

3- If the request is authorized, the requester sends the ticket to the agent executor of the action, e.g. printer-agent. The executor verifies the authenticity of the ticket and executes the action.

4- A confirmation is sent back to the requester agent to confirm or reports the failure of the execution of the action.

The ticket is a token that allows an agent to operate with the rights and privileges of the authorization server that granted the ticket. Naturally, it must be possible to verify that the authorization server granted the ticket. The validity of the ticket can be verified by using a public key digital signature. The ticket may include an expiration time. If a non-expiring ticket is desired, the expiration time can be set sufficiently far in the future. Furthermore, the ticket mechanism allows the agents to bill for the services they offer.
In the case the policy doesn't specify the agent executor of the action, the agent uses the Facilitator Agent to discover the set of agents providing the service. Hence the Facilitator forwards the ticket to these agents and offers will be made to the requester agent. The agents interested in this offer can then reply to the requester agent proposing to execute the action, and may specify additional constraints to execution of the action (e.g. more

cost), or provide some additional information about the execution (e.g. quality of printing). The requester chooses the best offer, or negotiates, according to the model described in next chapter, with agents that are providing the best offer.

3.3 Chapter Summary

In summary, this chapter provides a multi-agent architecture for setting up policies in a cooperative network according to different policy classes based on the activities they monitor. The advantage of this architecture is that it provides a dynamic way to adapt the behavior of agents to any changes in the system. Policies can be quickly setup to control the behavior of agents and make them behave according to the global strategy of the system. In the next chapter, we provide an example of policy definition and management in one of the sites involved in the virtual network, as well as, the negotiation model used to resolve conflicts between agents.

Chapter 4

Policy Management in PMMS

In order to better understand the concept of policy classification and the effect of system modes, this chapter describes an example of a policy management system, with reference to a working site located at the University of Ottawa. The example illustrates the classification of policies and how they control the behavior of agents belonging to the managed site.

4.1 Classification of Policies

As discussed in the previous chapter, policies dictate the global strategy of a site. Policies control the behavior of the agents and determine how they should react to any change in the system. Policies are classified according to the activity they monitor. Four classes of policies are defined for the management of the PMMS system: security, admission control, user profile, and resource reservation. These classes are used by the coordinator agent, site logon agent, site profile agent, and resource agent, respectively.

4.1.1 Default Policies

A default policy is defined with a low priority that is applies to all requests. The default policy is used to ensure that at least one policy will apply for a request. The default policy is expressed as follows:

ID	Mode	Subject	Action	Target	Constraint	Priority	System Mode
A000	Interdiction	*Any*	*Any*	*Any*		0	-

Table 4.1: Default Policy for the Site

The default policy is identified as A000, where A denotes that it is an authorization object. By default, the execution of any action in the system is forbidden. The low priority ensures that this policy will be overridden if at least one other policy applies to the request.

4.1.2 Security Policies

Inter-site communications are encrypted using different encryption algorithms such as DES and RSA [47]. Each encryption algorithm can use a different key length to encrypt data. Encryption policies specify which algorithm should be used for a specific communication between two different sites. Two modes are defined for system security: normal level and high level. Each mode has a set of policies that define the rules that the system should enforce when the mode is enabled. The following policies give an example of some security policies setup by the University of Ottawa to encrypt its communications with the other two sites.

ID	Mode	Subject	Action	Target	Constraint	Priority	System Mode
A001	Interdiction	*Any*	Encrypt	Security	Key_length<64	4	Normal security level
A002	Permission	Coordinator	Encrypt	Security	Key_length>64 and site = MITEL Or Key_length>128 and site = NRC	3	Normal security level
A003	Interdiction	Coordinator	Encrypt	Security	Key_length < 196	5	High security level

Table 4.2: Security Policies

The first policy (ID A001) applies to all requesters of an authorization to encrypt a communication. It specifies that it is forbidden for any agent to request a security agent to encrypt data with a key length less than 64 bits. This is a default security policy and sets the minimum degree of security needed for inter-site communications. The second policy (ID A002) is specific to the coordinator agent. It allows this agent to request encryption when it is establishing communication with the MITEL site using an encryption key length greater than 64 bits, or with the NRC site using a key length greater than 128 bits. University of Ottawa assumes that the connection with NRC is not secure enough. Thus, the key length used for encryption should be greater than 128 bits in length to ensure privacy of communication on top of an insecure communication channel. When the system security mode is set to high, the third policy (ID A003) sets more constraints on the key length.

4.1.3 Admission Control Policy

Admission control policies express rules, which are used to admit a new user into a visited site. They specify eligible users who are provided with access to the site and impose constraints on the login of these authorized users. These policies apply for the site login agent and apply each time a new user wishes to log onto a visited site. Some

examples of admission control policies used by the University of Ottawa site are set forth in Table 4.3, from which it will be noted that the subject and the target are the same agent.

ID	Mode	Subject	Action	Target	Constraint	Priority	System Mode
A004	Interdiction	Site logon	login	Site logon	{S} day = Sunday	4	Normal load
A005	Permission	Site logon	login	Site logon	User.Site = NRC Or (User.Site = MITEL and User.Category= Manager)	2	Normal load
A006	Interdiction	Site logon	login	Site logon		3	Heavy load

Table 4.3: Admission Control Policies

The first policy (ID A004) prohibits access to any user on Sundays. The second policy (ID A005) permits access to all NRC users but restricts access of MITEL users to MITEL managers. When the system switches to heavy load mode, the third policy (ID A006) is enabled, thereby prohibiting access to the site.

Examples of a second type of policy are shown in Table 4.4, where the ID is prefaced with an "O", which indicates an "obligation" object. The site logon agent is obliged to notify the event server (MicMac) when the number of users in the system reaches a predetermined limit. The related obligations are expressed as follow:

ID	Trigger	Mode	Subject	Action	Target	Constraint	System Mode
O001	On login	Obligation	Site logon	Notify("Heavy load")	Event server	{S} users = 7	Normal load
O002	On logout	Obligation	Site logon	Notify("Normal load")	Event server	{S} users = 4	Heavy load

Table 4.4: Site Logon Obligations

The notification of the first obligation (ID O001) is used by the policy server to switch the system mode to heavy load mode, whereas the notification of the second obligation (ID O002) is used to switch the system mode back to normal load.

4.1.4 User Profile Policies

The prototype of the PMMS presently in operation at the University of Ottawa is capable of offering three services: printing, email forwarding, and video mail service. A different quality of service (QoS) is used for each delivered service. For example, video service has three qualities of service. A high quality of service plays back an entire video for a user. Medium quality extracts video frames using a key framing application [1] and displays the resulting frames to the user. Low quality of service displays the resulting frames in black and white.

As discussed briefly in the previous chapter, the site profile agent maps the user's profile onto the local policies of a site. The following example defines some policies used by the University of Ottawa site to offer services to visiting users from MITEL and NRC sites. The example is limited to video service.

ID	Mode	Subject	Action	Target	Constraint	Priority	System Mode
A007	Permission	Site profile	Use_Video	Site Profile	QoS=1 and Cost >15 Or QoS=2 and Cost>25 Or QoS=3 and Cost>40	3	Normal resource mode
A008	Interdiction	Site profile	Use_Video	Site Profile	QoS=3 AND User.Category <> Manager	3	Normal resource mode
A009	Permission	Site profile	Use_Video	Site Profile	QoS=1 and Cost > 40 Or QoS=2 and Cost >70	4	Critical resource mode

Table 4.5: Policies for user profile mapping

The first policy (ID A007) sets a minimum cost required to use the video service. The cost depends on the quality of service to be used. For example, a high quality video service can not be used unless the user pays at least 45 units.

The second policy (ID A008) limits the use of the high quality of video service to managers from both sites.

The third policy (ID A009) is invoked during system critical resource mode. It raises the cost for using the low and medium quality of service and prohibits the use of the high quality of service.

Other policies are used to set more constraints with the use of video service, but are not described for the sake of simplicity.

4.1.5 Resource Reservation Policies

Each site offers its visitors a set of services resulting from the dynamic mapping of the user's profile and the local policies of the site. When the visitor starts a service, the agent representing the service reserves a predetermined amount of resources, CPU and bandwidth, required for the execution of the service with a specific quality, as set forth in Table 4.6:

ID	Trigger	Mode	Subject	Action	Target	Constraint	System Mode
O003	On startup	Obligation	Video agent	Reserve(CPU=20,Band=5)	Resource agent	QOS=1	Normal resource mode
O004	On startup	Obligation	Video agent	Reserve(CPU=35,Band=10)	Resource agent	QOS=2	Normal resource mode
O005	On startup	Obligation	Video agent	Reserve(CPU=75,Band=20)	Resource agent	QOS=3	Normal resource mode
O006	On startup	Obligation	Video agent	Reserve(CPU=15,Band=3)	Resource agent	QOS=1	Critical resource mode
O007	On startup	Obligation	Video agent	Reserve(CPU=25,Band=7)	Resource agent	QOS=2	Critical resource mode
O008	On startup	Obligation	Video agent	Reserve(CPU=45,Band=12)	Resource agent	QOS=3	Critical resource mode

Table 4.6: Video Agent obligations

These obligations define the amount of resources to be reserved for each quality of service in each mode. The video agent automatically adapts the requirements of the video application to the actual state of the system.

A resource reservation strategy is defined by policies that apply to controlling the reservation of resources via the resource agent, as set forth below in Table 4.7. Again, this example is limited to video service. In Table 4.7, and subsequently in this chapter, it should be noted that {S} stands for a system parameter (e.g. {S} Bandwidth is the

available bandwidth in the system whereas Bandwidth is the required bandwidth to be reserved by a resource agent).

ID	Mode	Subject	Action	Target	Constraint	Priority	System Mode
A010	Permission	Video Agent	Reserve	Resource Agent	{S} CPU > 200 and CPU < 30 Or {S} CPU > 300 and CPU < 50	5	Normal resource mode
A011	Permission	Video Agent	Reserve	Resource Agent	{S} Bandwidth > 40 and Bandwidth<5 Or {S}Bandwidth> 70 and Bandwidth<10	5	Normal resource mode
A012	Interdiction	Video Agent	Reserve	Resource Agent	CPU > 20 OR Bandwidth > 4	7	Critical resource mode

Table 4.7: Resource reservation policies

Policies that belong to normal resource mode are used to monitor the resource reservation strategy for this particular resource mode. The first and the second policies (ID A010 and ID A011) specify the maximum amount of CPU and bandwidth that the video agent can reserve depending on the available amount of CPU and Bandwidth on the site.

The third policy (ID A012) prohibits an excessive reservation of resources when the resource status is critical (the resource status can switch between two values: normal and critical). Critical status means that the available resources in the system are below a critical value.

After each reservation, the resource agent notifies the system if there has been any change in the resource status. This obligation is expressed as follow:

ID	Trigger	Mode	Subject	Action	Target	Constraint	System Mode
O03	On reservation done	Obligation	Resource agent	Notify("critical resource mode")	Event server	{S}CPU<100 OR {S} Bandwidth<40	Normal resource mode
O04	On resources released	Obligation	Resource agent	Notify("normal resource mode")	Event server	{S}CPU>200 And {S} Bandwidth>70	Critical resource mode

Table 4.8: Resource Agent Obligations

4.1.6 System Policies

Agents are designed to notify the policy server when a change in the system mode occurs. The policy server reacts to these changes by enabling and/or disabling certain policies in order to adjust the system behavior to the requirements resulting from the new state. These reactions are expressed as obligations:

ID	Trigger	Mode	Subject	Action	Target	Constraint	System Mode
O05	On heavy load mode	Obligation	Policy server	Enable("heavy load mode") Disable("normal load mode")	Policy server	-	-
O06	On critical resource mode	Obligation	Policy server	Enable("critical resource mode") Disable("normal resource mode")	Policy server	-	-

Table 4.9: Policy server obligations

The enabled policies guarantee that the system will behave to fit the requirements of the new system state. For example, the heavy load mode enables Interdiction A006, which prohibits any user from logging into the system.

4.2 Processing Policy Requests

Since policies are independent, and may be added to the system by different administrators, many policies may apply for the same request. The authorization server processes all of these policies in order to make a final decision as to which policy will prevail. First, the authorization server loads all enabled policies that apply for the request, and then it evaluates each policy to determine if the conditions specified by the policy constraint apply. The evaluation of a policy constraint involves three steps: constructing a decision tree, decorating the tree and extracting the decision related to the policy. The actual mode is then added to the attributes of the policy. The authorization server selects the applicable policies with highest priority in order to make the final decision concerning a request.

An example is provided below to illustrate the steps used in processing a request. The example relates to the reservation of system resources by a video agent, which executes a video service with a specific quality of service. The exemplary request is expressed as follow: **Video agent asks authorization to reserve (CPU=25, Bandwidth=15) by resource agent**. The following tables summarize the state of the system and the amount of resources available in the system at the time of the request is made.

Resource	Value
CPU	250
Bandwidth	100
Disc	60

Table 4.10: Available Resources

System Mode	Status
Normal resource mode policies	Enabled
Critical resource mode policies	Disabled
Heavy load policies	Disabled
Normal load policies	Enabled
High level security policies	Disabled
Low level security	Enabled

Table 4.11: System Modes

The authorization server first loads all enabled policies that apply to this request.

The following table gives the policies that apply for video agent requests.

ID	Mode	Subject	Action	Target	Constraint	Priority	Class
A000	Interdiction	Any	Any	Any		0	-
A010	Permission	Video Agent	Reserve	Resource Agent	{S} CPU > 200 and CPU < 30 Or {S} CPU > 300 and CPU < 50	5	Normal resource mode
A011	Permission	Video Agent	Reserve	Resource Agent	{S} Bandwidth > 40 and Bandwidth<5 Or {S}Bandwidth> 70 and Bandwidth<10	5	Normal resource mode
A021	Interdiction	Video Agent	Reserve	Resource Agent	Bandwidth > 12 OR CPU > 70	5	Normal resource mode

Table 4.12: Policies that apply for the video agent request

The evaluation of a policy involves an evaluation of the conditions representing the constraints of the policy. The policy A000 has no constraint; therefore, the actual mode of the policy is Interdiction. The policy A010 has a constraint to be applicable. The evaluation of the constraint determines if the policy applies or not to the request.

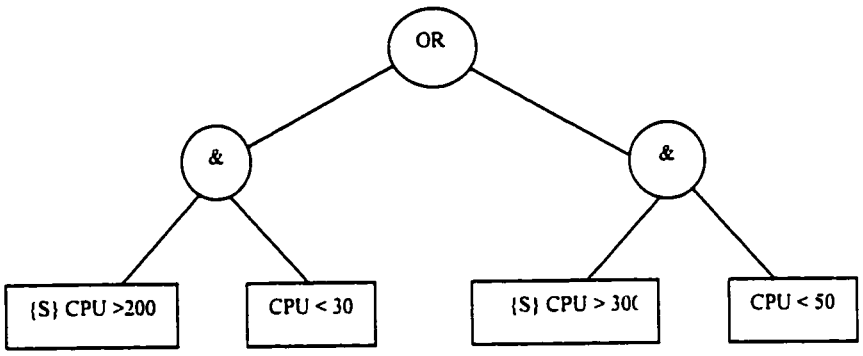


Figure 4.1: Constraint of Policy A010

The tree diagram of Figure 4.1 represents the constraint of the A010 policy. Each leaf of the tree represents a condition of the global constraints. Although the example set forth herein is characterized by a simple representation of conditions, it will be understood that more complex representations are possible. Conditions that are preceded by {S} stands for a condition that depends on the system, otherwise, the condition depends on the parameters specified by the request. The tree is decorated according to the values of parameters in the request and the actual values of system parameters (e.g. the available CPU in the system). Figure 4.2 shows the decorated tree representing the policy A010. The result of the evaluation of this policy is permission.

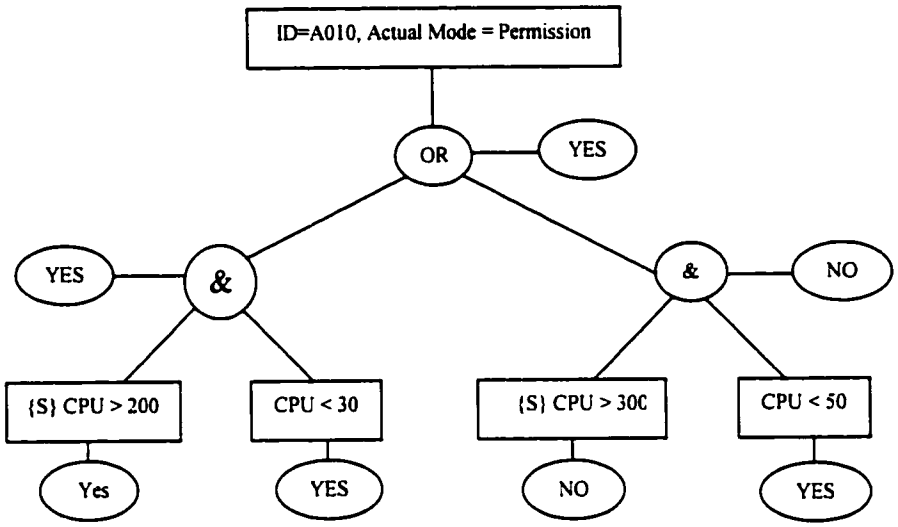


Figure 4.2: Decorated Tree of Policy A010

The policy A011 is evaluated using the same technique. The decorated tree of this policy is illustrated by Figure 4.3. This policy neither permits nor prohibits the request. Therefore, the actual mode of the policy is set to "not applicable".

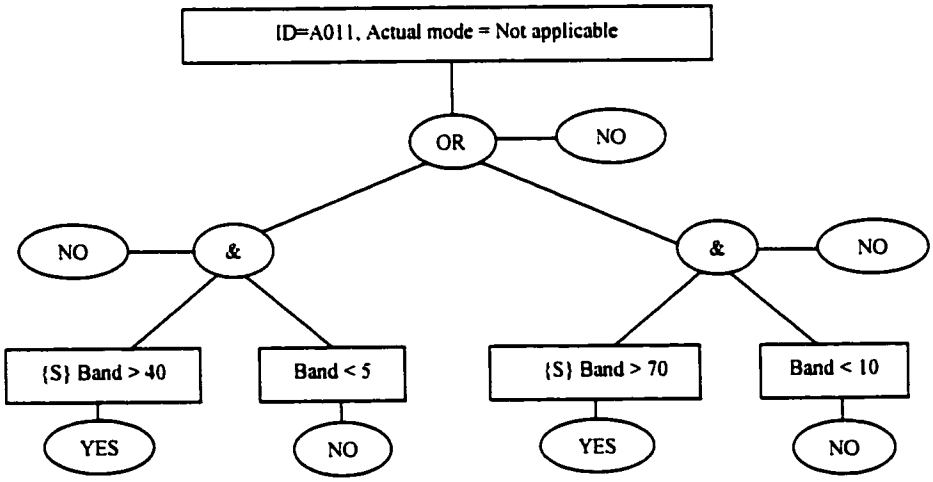


Figure 4.3: Decorated Tree Policy A011

The decorated tree of the policy A021 is illustrated by the tree diagram of Figure 4.4. The evaluation of this policy is interdiction.

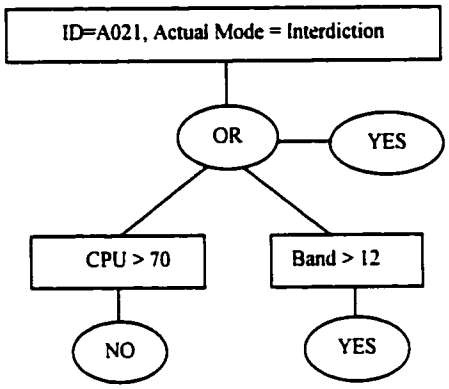


Figure 4.4: Decorated tree of policy A021

Table 4.13 summarizes the results of the evaluation of policies:

ID	Mode after evaluation	Priority
A000	Interdiction	0
A010	Permission	5
A011	Not applicable	5
A021	Interdiction	5

Table 4.13: Summary of the policy evaluation

The authorization server uses the highest applicable policies to make a decision. In this example, it uses policies A010 and A021. By default, Interdiction overrides permissions. Thus, the request is rejected.

The above example shows three different modes that result from the evaluation of policies. Another mode, conditional, is applicable if some parameters of the request are to be modified before an authorization can be granted. This type of result is used for negotiation between agents.

4.3 Negotiation Model

A request for an authorization may be refused depending on the evaluation of the authorization server due to a conflict between different policies. Therefore, the requesting agent is not able to accomplish its tasks, which may cause chaos in the system. For example, if a video agent needs to reserve an amount of bandwidth to transfer a video over the network, the request to the authorization server may be refused due to a lack of resources or insufficient cost to pay the reserved bandwidth. Hence, the conflicting agents need to start a negotiation session to resolve this conflict. The negotiation model used in this work defines a way to reach an agreement between two or more agents having different beliefs. Agreement is reached when the proposal does not violate any policy of the negotiating agents.

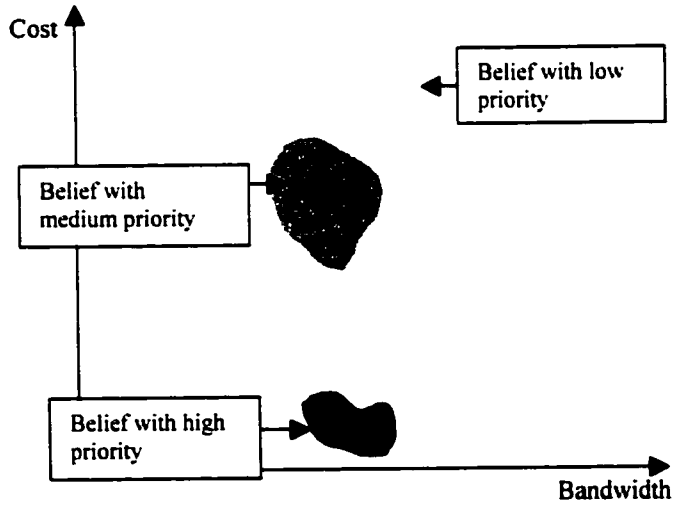


Figure 4.5: Belief of Video Agent

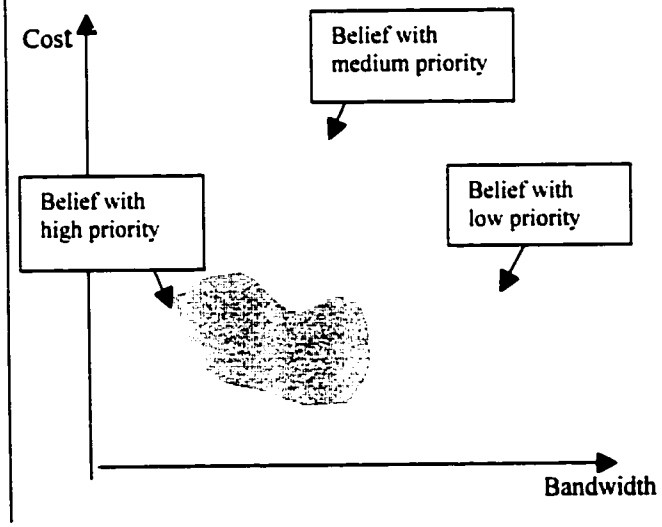


Figure 4.6: Belief of Authorization Server

We define belief of an agent as the combination of all policies monitoring that agent. Policies with the same priority define a region of belief. Depending on the priority associated with a region of belief, it can be strong, medium or weak for high, medium, and low priority respectively. As shown in Figure 4.5 and 4.6, we can represent belief of a PMMS agent as a function B that is expressed as:

$B = \sum f_p(policies)$, where each region of the graph has a priority p to express the degree of belief of the agent. For example, in Figure 4.5 the video agent strongly believes that it must reserve a certain amount of bandwidth at a small cost. On the other hand, it weakly believes (i.e. does not prefer), to reserve the same amount of bandwidth at a high cost. Also, a request, we call proposal, is represented by a point P on the graph.

A measure between a point P, i.e. a proposal, and a region of belief R_b may be defined as:

$M(P, R_b) = \min (d(P,X) + \frac{1}{p}); \quad \forall X \in R_b$; where d is the distance between two points (i.e.the Cartesian distance: $\sqrt{X^2 + Y^2}$), and p denotes the priority of the region.

In the above formula, the 1/p term expresses the tendency of an agent to prefer its strong beliefs to weaker ones.

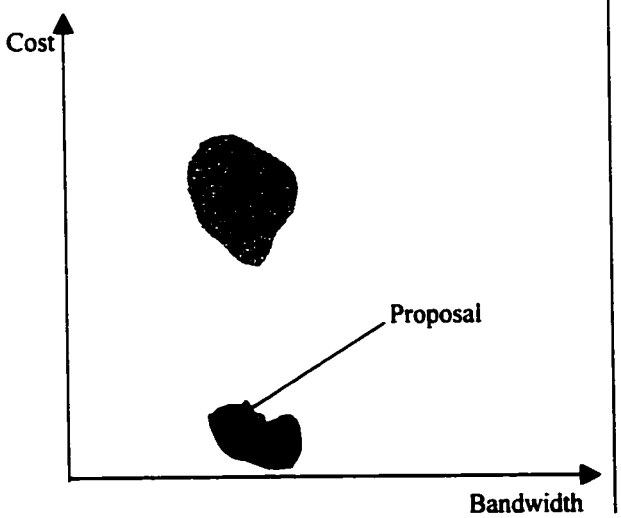


Figure 4.7: First Proposal from Video Agent

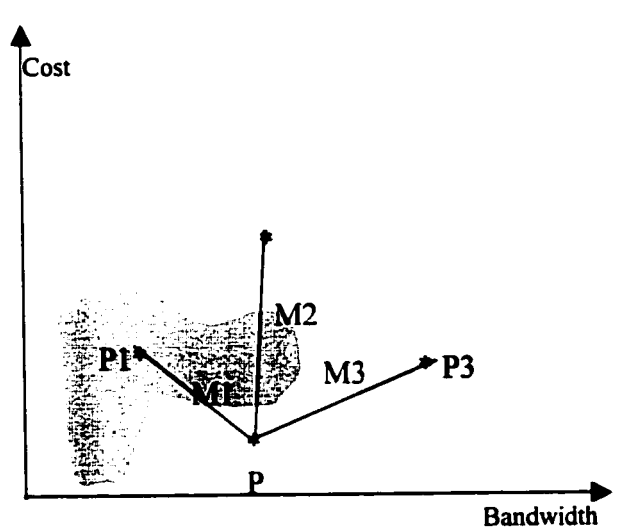


Figure 4.8: Processing the Proposal

Thus, when an agent wants to propose, or receives a proposal, it uses the formula above to make the decision. Returning to the example above, the video agent chooses a point from its strong belief region, i.e. the region with the highest priority, and sends the proposal to the authorization server as shown in Figure 4.7 and 4.8.

Using the above formula, the authorization server looks for the nearest region to the proposal. Specifically, it measures the distances M1, M2, M3 between the point P and the regions R1,R2 and R3 respectively. Since M1 is the smallest distance, i.e. R1 is the nearest region, the authorization server, chooses to guide the video agent to the first region of belief R1.

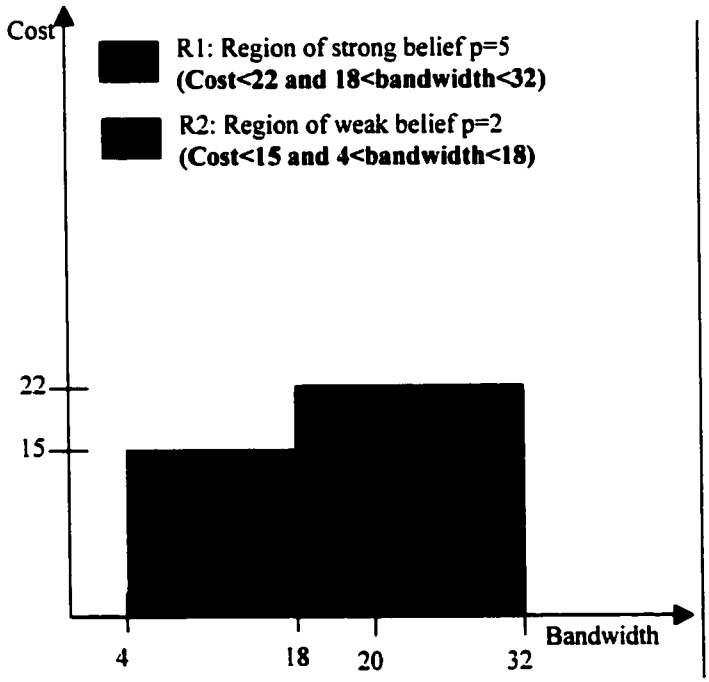


Figure 4.9: Policies of Video Mail agent

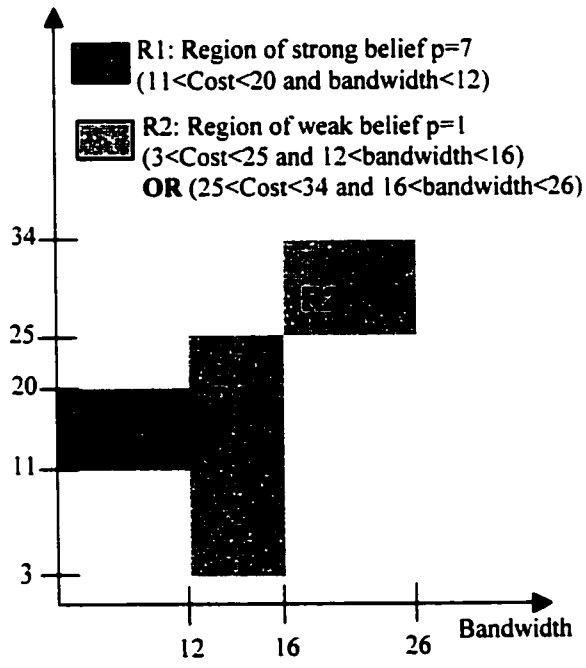


Figure 4.10: Policies of Authorization Server

Let's take a real example to show how the negotiation works. In the video agent example, the functions that represent the belief of the agents are linear, such that belief can be represented by the constraints of the policies, for example, **(cost<22 and 18<bandwidth<32 with priority p=5)** ***OR*** **(Cost<15 and 4<bandwidth<18 with priority p=2)** as belief of the video agent, and **((11<Cost<20 and bandwidth<12 with priority p=7)** ***OR*** **((3<Cost<25 and 12<bandwidth<16)** ***OR*** **(25<Cost<34 and 16<bandwidth<26) with priority=1))** as belief of the authorization server, as shown in Figures 4.9 and 4.10.

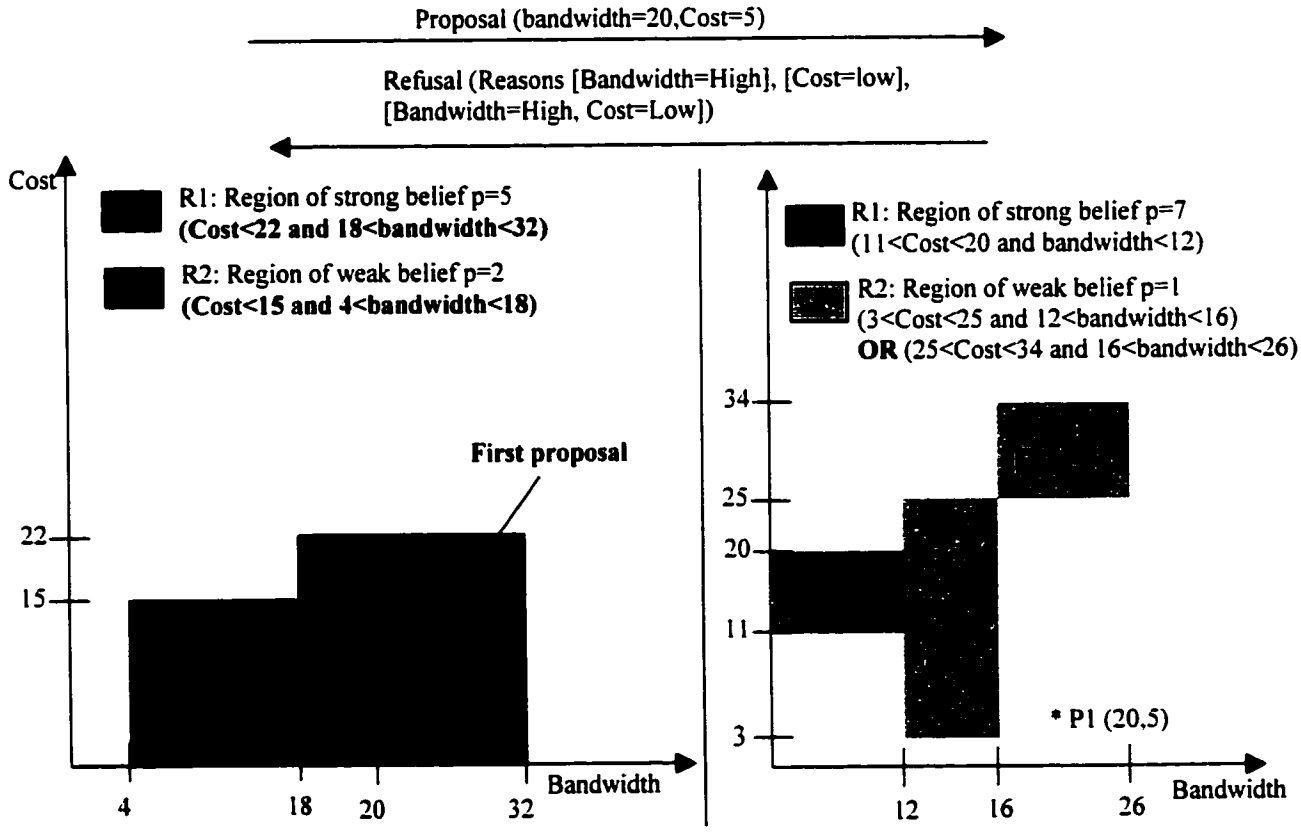


Figure 4.11: First Proposal

The belief, represented above, is used by the two agents to propose or to evaluate a proposal. The following specific formula is used in PMMS when an agent has to evaluate a proposal and deliver a counter proposal.

$$\mathbf{M(P,R)=min}\ (\sqrt{(x_p - x)^2 + (y_p - y)^2} + \frac{2}{p^3}); \quad \forall X(x,y) \in R$$

The Video Agent starts by sending a proposal to the authorization server, as discussed in greater detail below. The authorization server refuses the proposal since it violates all of its policies, however, it tries to guide the requesting agent to the nearest region (i.e. nearest in the sense of the metric M), which is R2 in the present example (Figure 4.11).

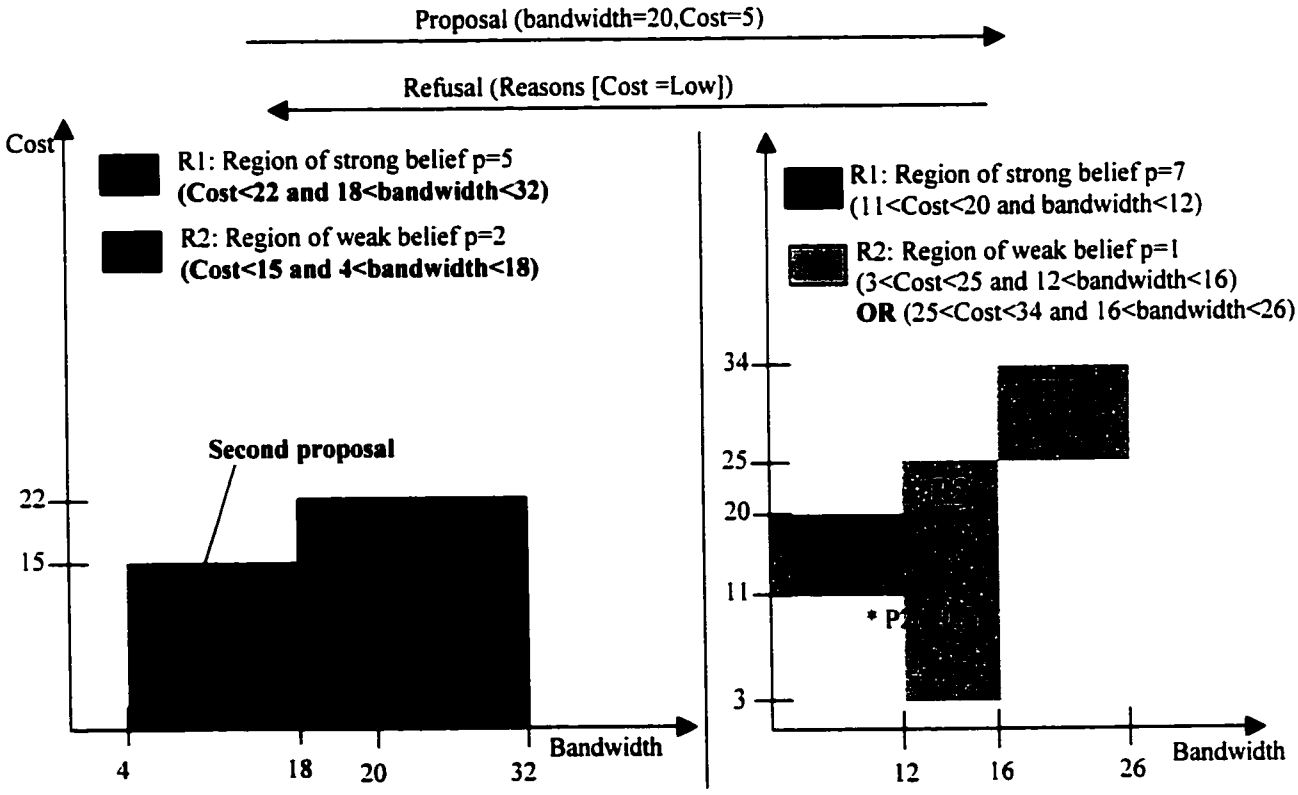


Figure 4.12: Second Proposal

Using the hints provided by the authorization server, the video agent tries to find the best proposal (according to the metric M) that is contained in its regions of belief and satisfies the conditions sent by the authorization server. The second proposal is then sent to the authorization server (Figure 4.12). Like the first proposal, the authorization server looks for the nearest region to the proposal, i.e. R1, (the use of the priority term in the formula of measurement makes the region R1 the nearest region to the proposal P2), and tries to guide it to that region. It then sends the response to the video agent.

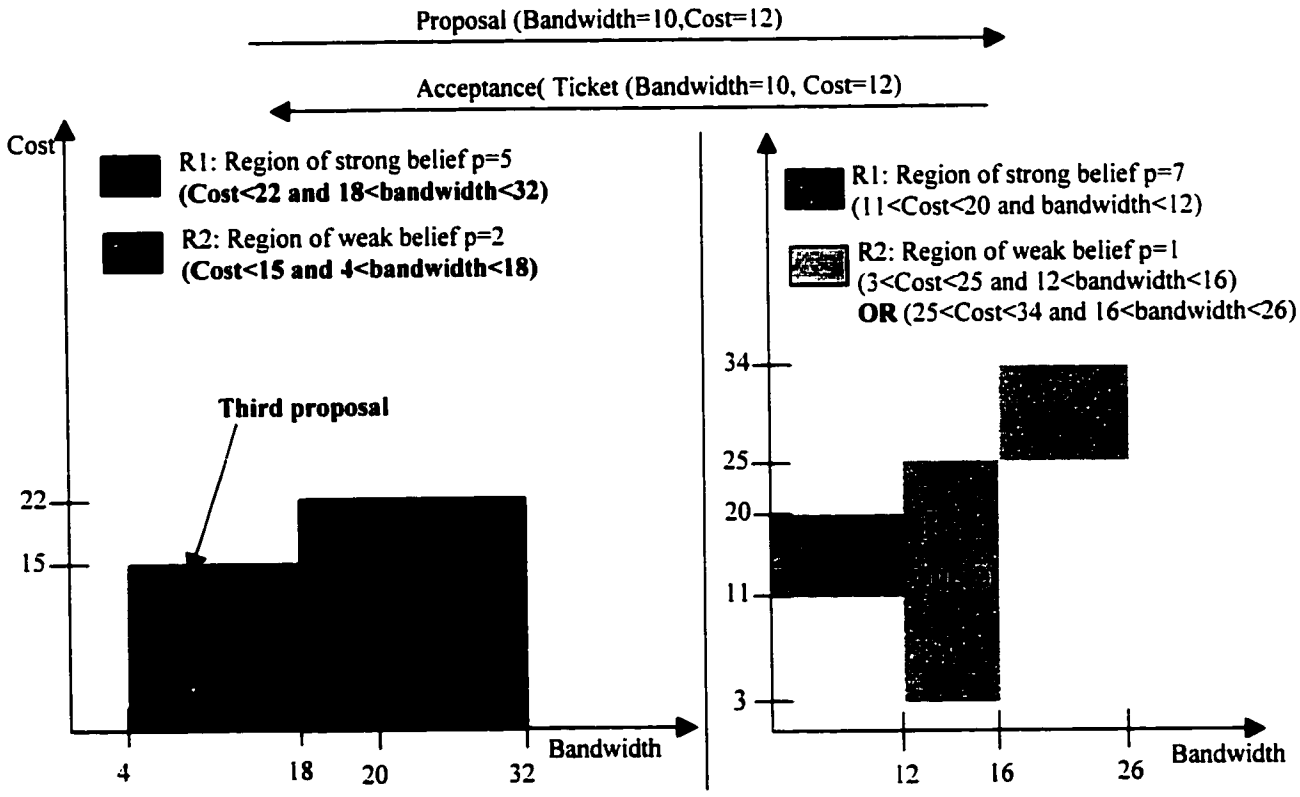


Figure 4.13: Agreement is reached

Using the last hint, the video agent looks for a third proposal (Figure 4.13). This time it sends a proposal that gets the agreement of the authorization server, since it does not violate any of its policies.

4.4 Chapter Summary

This chapter presented an example of a site management in the PMMS system. It illustrates in detail how policies are represented and classified. The advantage of this representation is that policies can be easily read and understood by system administrators. Also, we explained how policies are evaluated upon request of an authorization. The last section presented a negotiation mechanism to resolve conflicts, which are inevitable in a system fully managed by autonomous agents. Finally, we note that some of the ideas presented in chapter 3 and 4 are patent pending.

Chapter 5

Implementation

The implementation of this work consists of a multi-agent system comprised of a graphical user interface, Policy Server, Authorization Server, and an Policy Layer. All Agents in the system collaborate towards the achievement of a joint objective that is driven from the management policies. Each agent has a policy layer to analyze and take decision based on the information it receives from the policy server, event server, or other system components. Hence, the autonomy of agents is guaranteed. Cooperation between agents is strengthened by a limited communication protocol built on top of MicMac.

In what follows, we describe the representation of policies using the eXtensible Markup Language (XML) language, the communication layer built on top of MicMac, the agent development model, and the mapping of user profiles with system policies to build a session profile for a roaming user.

5.1 Technology Integration

This section briefly describes the tools used for the implementation of this system. As illustrated in figure 5.1, Java is used for the implementation of the agents, MicMac ensures the collaboration of distributed agents, KQML provides the communication layer, and XML is used for the representation of the messages content.

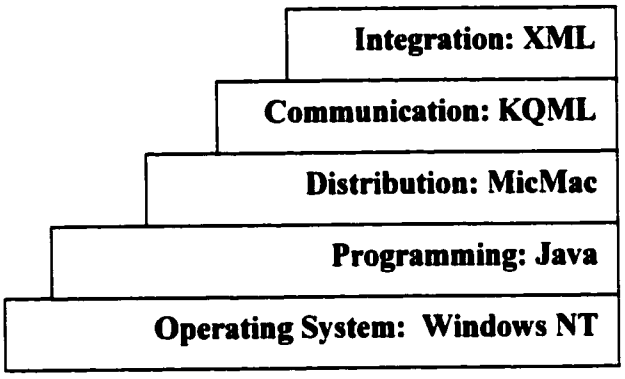


Figure 5.1: Implementation Tools

5.1.1 Java Language

The Java language has become the most used language for writing agents systems and applications. It offers the possibility to write platform independent software running on top of a Java Virtual Machine (JVM). The same software can be executed on different systems like Unix, Windows, or MAC. In addition to its platform-independent approach, Java has several advantages with respect to the development of software agents:

- Object-Oriented programming language with a syntax like C++, but cleaner code than C++.
- Classes can be dynamically loaded during execution time.
- Multi-threading is built-in to the language, with support for basic resource locking and synchronization.
- Garbage collection is handled by the Java Virtual Machine. There is no need for destructor methods to release the memory occupied by an object.

5.1.2 MicMac

MicMac is a set of software tools, which provide a common medium for the communication of multiple, distributed agents. The mechanism of communication is based on the Blackboard paradigm. A tuple space, called agora in MicMac terminology, is considered as an open area for exchanging information between agents. The communication mechanism is very simple: any agent that wants to communicate with another agent will set the receiver of the message to the latter agent and posts it in this common medium. The receiver will then pick it from the shared blackboard. Therefore, all messages are sent to the communication medium, which dispatches them to the agent listening for it. Private agoras can also be created for private agents' communication. MicMac implements a shared memory, which stores information as tuples that are unordered set of fields represented as "key-value" associations.

5.1.3 KQML

The Knowledge Query and Manipulation Language (KQML) is a language and protocol for exchanging information and knowledge [11]. It is part of the ARPA Knowledge Sharing Effort [42], which is developing methods and techniques for building large-scale knowledge bases, which can be shared and reused. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. It can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge, in support of cooperative problem solving. It focuses on an extensible set of performatives, which defines the permissible operations that agents may attempt on each other's knowledge and

goal stores. The performatives comprise a substrate on which to develop higher-level models of inter-agent interaction such as contract nets and negotiation. In addition, KQML provides a basic architecture for knowledge sharing through a special class of agent called communication facilitators, which coordinate the interactions of other agents.

5.1.4 XML

The extensible Markup Language (XML) is an emerging standard for the representation of data for sharing and retrieval of information and efficient publishing to multiple users [46]. XML has the following advantages:

XML is flexible, making it ideal for describing any block of content at any scale, from a user bookmark, to an entire database.

- It supports a wide variety of applications for authoring, browsing and so on.
- It's simple to write programs that processes XML documents, especially with the availability of different parsers provided by companies like SUN, and IBM.
- XML documents are "human" and clear, so one can view XML source code in a text-editor and understands its meaning.

5.2 XML Policy Representation

Policies are downloaded from the Policy Server to system agents running on different platforms, therefore, they should be represented in a platform independent format that can be understood by software agents developed by different programming languages. We use XML for policy representation because it becomes a standard for the exchange of structured documents between different systems.

5.2.1 Policy Document Type Definition of a policy

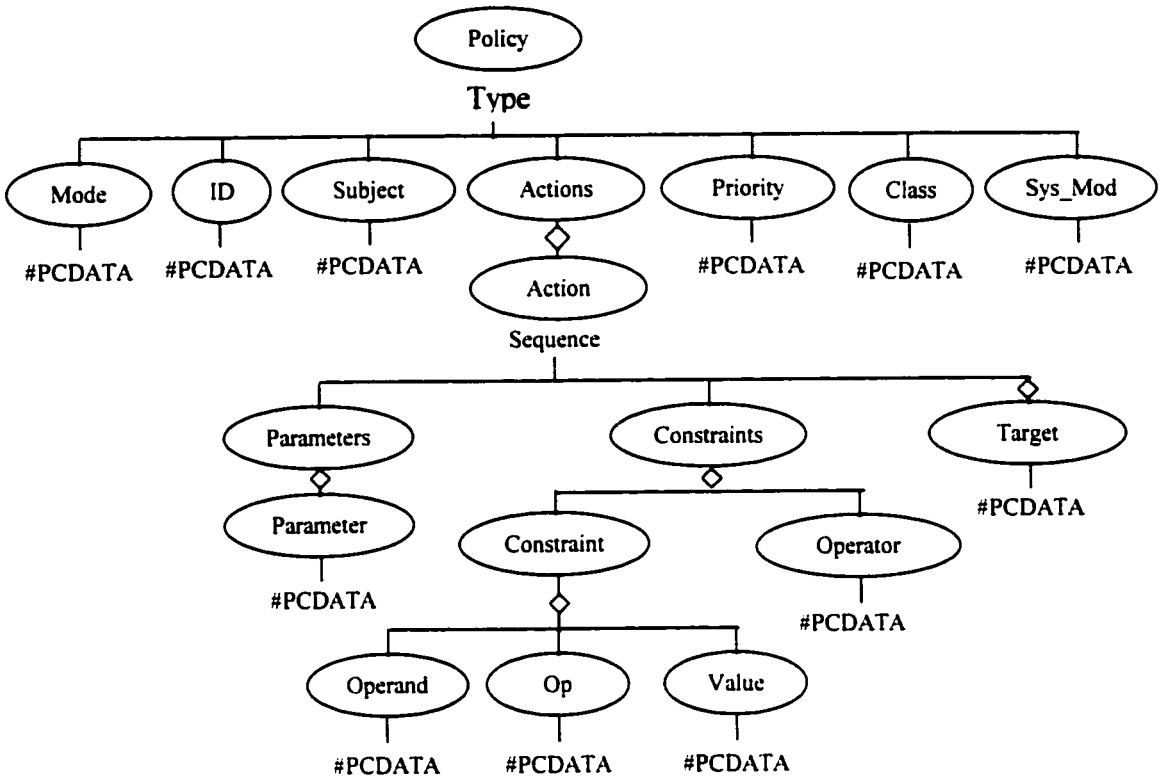


Figure 5.2: Policy Document Type Definition

As shown in Figure 5.2, each policy item has one or multiple actions associated with it. Depending on the mode of the policy, the agent is permitted or prohibited to execute these actions. Each action item has an agent executor of the action specified in the target item. Constraints of the action determine the conditions to be satisfied before the policy can be applied. Each constraint is composed of an operand, operator, and a value. An operand may be expressed using system values, like the bandwidth available at the time of the processing of the constraint; or using one of the parameters of the action. A policy may have many actions associated with it. These actions may have an order relationship expressed by the sequence item. They can be ordered, or unordered: ordered

actions have to be executed in the order specified in the policy rule; unordered actions can be executed in any order.

5.2.2 Example of an XML policy representation

Below we provide an example of a resource reservation authorization represented as an XML document.

```
<Policy type="Authorization">
<Mode> Permission </Mode>
<ID> A010 </ID>
<Subject> Video Agent </Subject>
<Actions Sequence="Ordered">
<Action>Reserve
<Parameters>
        <Parameter> CPU </Parameter>
</Parameters>
<Constraints>
<Constraint>
        <Operand type="System"> CPU <Operand>
        <Op> Greater </Operator>
        <Value> 200 </Value>
</Constraint>
<Operator>AND</Operator>
<Constraint>
        <Operand type="Parameter"> CPU <Operand>
```

<Op> Less </Operator>

<Value> 10 </Value>

</Constraint>

</Constraints>

<Target> Resource Agent </Target>

</Action>

</Actions>

<Priority> 5 </Priority>

<System_Mode> Normal Resource Mode </System_Mode>

<Class> Resource Reservation</Class>

<Creator> Administrator </Creator>

An optional element, which is not shown in the above representation, is the trigger element. The trigger is required when the type of the policy is an obligation. When triggered, the duties of the obligation have to be performed.

5.3 Graphical User Interface

The Graphical User Interface is comprised of three components: a policy editor, constraint editor, and policy validation and transformation component.

5.3.1 Policy Editor

The Policy Editor allows an administrator to perform the basic operations on policies, that is, viewing, editing, deleting, enabling and disabling policies. The policy interface has the following menus:

- View: Lists existing policies without the possibility to modify them. The viewing of policies can be restricted to a site of the virtual network, to a class of policies, or to a specific agent.
- Edit: Adds a policy to the system, or modifies an existing policy.
- Delete: Deletes an existing policy from the system.
- Enable: Puts a policy in a state where it can be processed by the authorization server, or agents when its constraints are satisfied.
- Disable: Puts a policy in a state where it is not processed even though its constraints are satisfied.
- Trigger System Mode: Puts the system in a specific state. System modes can be related, and therefore, only one system mode of related modes can be enabled at a given time.

5.3.2 Constraint Editor

Because of the complexity of the representation of the policy constraints, it was necessary to develop a constraint editor in order to ease the creation and the viewing of the constraint. The editor displays of the policy constraints as a tree. Each node of the tree represents an operator of the constraint, while leafs represents the policy operands. An administrator can also check the validity of the policy constraint by checking its syntax.

5.3.3 Policy Validation & Transformation

Once a new policy is entered, the Policy Validation and Transformation component will validate the syntax of the policy. It uses the API of the constraint viewer to determine whether the constraint is valid or not. If the policy is valid, it proceeds to the

representation of the policy using the XML syntax and then distributes it to the policy server. If the policy is not valid the editor displays a message box notifying the user of the failure of the requested operation.

5.4 Agent Development Model

Agents are implemented with Java language for the reasons given before. In our model, an agent is an aggregation of three layers: Policy, Communication, and Actions as illustrated in Figure 5.3.

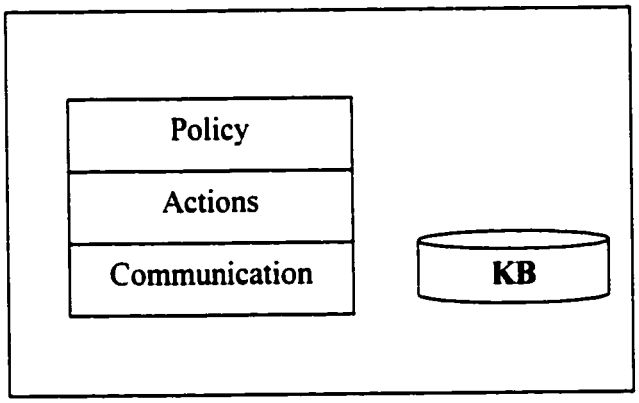


Figure 5.3: Agent Development Model

5.4.1 Policy Layer

A Policy layer is providing all the functions for receiving policies, requesting a policy, subscribing to system events, and notification policy server of its roles.

All System agents in PMMS implements the Agent Interface, which is comprised of the following methods:

- **Request**: Used by agents to request an authorization to perform an action. The method encapsulates the request in a KQML message where the content of the

message is represented using XML language, then it's sent to the authorization server via MicMac.

- **Response**: waits for the authorization to process the request and then retrieves the result from the communication channel. If the request was approved, then the response will include a ticket that gives the agent the right to execute the action or ask another agents to perform the action.
- **Ask:** Used by the agent to request another agent to perform an action. The method will include the ticket issued by the authorization server in the content of the message sent to the executor.
- **Perform:** Performs an action requested by an agent. The method checks first the validity of the ticket sent by the requester agent, such as the non expiration of the ticket, and performs the action. A confirmation of execution is sent back to the requester informing him of the success, or eventually the failure of the execution of the action.
- **Get-Policies:** Listens to the communication channel and gets any policy sent by the policy server to the agent. The trigger of the policy are then subscribed to the event server using the subscribe event method.
- **Subscribe Event:** Subscribe an event to the event server. If the event occurs, the agent will be automatically informed.
- **Unsubscribe Event:** Unsubscribe an event from the event server in case the agent is no more interested in the occurrence of this event. This can be the case of triggers of deleted or disabled policies.

- **Notify_Event**: Informs the event server of the occurrence of an event that is produced by the agent. For example, upon reception of an email, the email server will notify the event server by sending email_received event.
- **Notify_Role**: allows an agent to send to the policy server the set of roles it plays in the system.

5.4.2 Communication Layer

This layer is comprised of the MicMac client Application Programming Interface (API), which implements the following methods:

- Post: Adds a new tuple to the shared memory.
- Pick: Reads and deletes a tuple from the shared memory.
- Peek: Reads a tuple from the shared memory.
- Cancel: Terminates a pending pick/peek request.

5.4.3 Actions Layer

This layer is the code of the agent and implements its behavior. PMMS has a set of agents providing the necessary services to support user mobility.

- Site Logon Agent: authenticates users and opens new sessions.
- Coordinator Agent: manages the inter-agent communications. Its central role is to establish communication among two or more agents and control their activities when exchanging information.
- Site Profile Agent negotiates with its remote peer in the user's home site, the user preferences, and maps them to the local policies of the system. The result of the mapping is a set of services that the site can offer to its visitors.

- Resource agent manages system resources and is requested by other agents for resource reservation.
- Moreover, the system contains a set of agents each one of them representing a service or a device. Each one of them reserves necessary resources for the execution of the service it represents.

5.4.4 Knowledge Base

Upon receipt of a policy, the agent stores the element of policy in a local knowledge base. The set of actions and constrained of policies are the knowledge of the agent. This knowledge is used to perform an action upon reception of a new event from the event server, and to formulate a proposal to the authorization server when asking for an authorization.

5.5. Policy Agents

5.5.1 Policy Server

The policy server is a java standalone program that is responsible of the distribution, retrieval of policies. It maps user profiles to local policies of the system and monitors system modes, so that enabled policies meet with the requirements of the current system state.

- **Set_Policy (Agent_Id, Agent_Role, Policy)**: the policy server uses this operation to distribute any new Policy to the agent identified by the Agent_Id. The role of agent to which the obligation applies is set by the Agent_Role parameter.
- **Set_Authorization (Authorization):** Used only to distribute authorizations to the authorization server.

- **Retrieve_Policy(Agent_Id, Policy_Id)**: Retrieves a policy from an agent. The policy can either be an authorization or an obligation. In the case the policy server needs to retrieve the policy from the system, an Agent Identifier referring to all agents can be specified instead of an identifier for a particular agent.
- **Get_Role(Agent_Id)**: This operation is used by the policy server to learn the roles of a specific agent identified by Agent_Id.
- **Map_Profile:** Negotiate with Profile Agent, a session profile for a mobile user. The session profile is comprised of services the user can access during its work session at the remote site, and the quality of services associated with them.

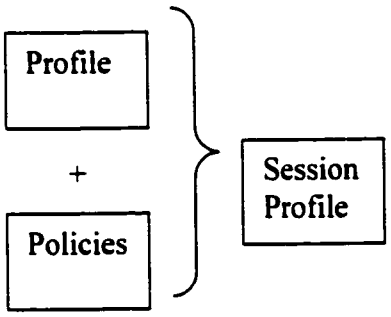


Figure 5.4: Mapping User Profile to Local Policies

- **Enable_Mode:** Puts the system in a new mode where new policies apply. PMMS supports six system modes as illustrated in table 5.1.

System Mode	Description	Category
Normal Resource Mode	Resources are available in the system	1
Critical Resource Mode	Lack of resources	1
Heavy Load Mode	Number of users is greater than a limit	2
Normal Load Mode	Number of users is under a limit	2
High Level Security Mode	System is vulnerable to external attacks	3
Low Level Security Mode	System is well protected.	3

Table 5.1: Description of System Modes

All system modes belonging to the same category are related, and therefore, one system mode of each category can be enabled at each time.

5.5.2 Authorization Server

The Authorization Server implements the engine for processing agent requests. In addition to the communication and agent interfaces, the authorization server implements the following method:

- **Process_Request:** Processes an agent request for an authorization. The server first loads all policies applicable to the request then makes a decision whether the agent shall be allowed to execute the action or not. In case of authorization, a ticket is delivered to the requester. The ticket has attributes to determine the authority that issued it, the time of its expiration, the cost of the operation, and some other useful information to be used for security and accounting purposes.

Chapter 6

Conclusion

6.1 Summary

This thesis has proposed a multi-agent architecture for enforcing policies in a cooperative network according to different classes based on the activities they monitor. The policy server is responsible of the management of all system policies (distribution, enabling, disabling, changing system modes, etc.). As part of its functions, it distributes the obligations to the agents, and the authorizations to a global authorization server that will deliver necessary authorizations to agents. The advantage of this architecture is that it provides a dynamic way to adapt the behavior of agents to any changes in the system. Policies can be quickly setup to control the behavior of agents and make them behave according to the global strategy of the system. Conflicts between agents are detected and resolved through negotiation.

A possible extension to the system is to make each agent contain it's own private authorization server that can receive policies from the global policy server and deliver necessary authorization to the agent. This approach will remove the bottleneck of a centralized common authorization server. By this way, each agent will manage its own policies and takes decision without referring to a central authority. More security mechanisms should be setup to prevent malicious agents from generating false permissions to execute some actions prohibited by the system policies.

6.2 Future Work

As future work of this research, we suggest the definition of a policy language specification to introduce policies in the system. Managers will be able to express policies in a high level language. A policy transformation and refinement agent will be responsible for the transformation of these high level policies into a set of low level policies that will be used to accomplish the goal specified by managers. The policy server will distribute the resulting low-level policies to agents that are specified as subjects of these policies. Another interesting field of research that can be explored as an extension to the current work is the usage of the Java Expert Shell System (JESS) [45] for the knowledge representation and agent reasoning. This approach has the advantage of the usage of an intelligent system widely used in the AI field, for policy storage and processing agents' requests. In this case an algorithm is needed to do the conversion of policies to JESS rules. Last, since the mobile agent technology is an emerging technology that has the potential to replace traditional client server technology, we suggest to apply the current policy management system to monitor the behavior of mobile agents.

References

[1] M. Ahmed and A. Karmouch, "Improving Video Processing Performance using Temporal Reasoning," SPIE-Applications of Digital Image Processing XXII, Denver Co, USA, July 20-23 1999.

[2] A. Karmouch, "Mobile Software Agents for Telecommunications," IEEE Communications, July 1998.

[3] P. Maes, "Intelligent Software", In Proceedings of Intelligent User Interfaces 1997, Orlando, Florida.

[4] C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages," Artificial Intelligence, 8, No. 3, pp 323-364 (1977).

[5] J. Russell, and P. Norvig, "Artificial Intelligence: A Modern Approach," Englewood Cliffs, NJ: Prentice Hall, 1995.

[6] P. Maes, "Artificial Life Meets Entertainment: Life like Autonomous Agents," Communications of the ACM, 38, 11, 108-114, (1995).

[7] M. Wooldridge, and N. Jenning, "Agent Theories, Architectures, and Languages: a Survey," in Wooldridge and Jennings Eds., Intelligent Agents, Berlin: Springer-Verlag, 1-22, 1995.

[8] C. Brustoloni, "Autonomous Agents: Characterization and Requirements," Carnegie Mellon Technical Report CMU-CS-91-204, Pittsburgh: Carnegie Mellon University.

[9] J. Mazur, "Learning and Behavior," Ed. J. Mazur , Pub Prentice Hall 1994.

[10] P. FARJAMI, C. Gorg, and F. Bell "Advanced Service Provisioning Based on Mobile Agents," First International Workshop on Mobile Agents for Telecommunication Applications, MATA 99.

[11] Y. Labrou, J. Mayfield, T. Finin, "KQML as an Agent Communication Language," Proceedings of the third International Conference on Information and Knowledge Management (CIKM'94), ACM Press, Nov. 1994.

[12] M. Genesereth, and R. Fikes, et al., "Knowledge Interchange Format: Version 3.0 Reference Manual," Technical Report, Computer Science Department, Stanford University, 1992.

[13] P. Cohen, and H. Levesquem, "Communicative Actions for Artificial Agents," Proceedings for the First International Conference on Multi-Agents, San Francisco. Cambridge, AAAI Press, pages 65-72, (1995).

[14] D. Sadek, P. Bretier, and F. Panaget, "ARCOL agent communication language and MCP, CAP and SAP agent's cooperativeness protocols." Response to FIPA Call For Proposals, http://www.cselt.it/fipa/torino/cfp1/propos97_015.htm.

[15] G. O'Hare, and N. Jenings, "Foundations of Distributed Artificial Intelgence", John Wiley & Sons, 1996.

[16] R. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," IEEE Trans. on Computers, Vol.29, No.12, pp.1104-1113, 1980.

[17] J. E. White, Telescript Technology, The Foundation of the Electronic Marketplace, General Magic White Paper, 1994.

[18] D. Marriot, and M. Sloman, "Management Policy Service for Distributed Systems," IEEE 3rd Int. Workshop on Services in Distributed and Network Environments, Macau, June 1996.

[19] G. Hughes, and M. Creswell, "An introduction to Modal Logic," Metheun, London,1968.

[20] D. Jonsher, "Extending access control with duties: Realized by active mechanisms," Proceedings of the IFIP WG 11.3 Sixth Working Conference on Database Security. IFIP WG 11.3.

[21] K. Becker et al., "Domain and Policy Service Specification," number S-SI-07-I-2-R. IDSM Deliverable D6, SysMan Deliverable MA2V2.

[22] T. Kotch et al., "Policy Definition Language for automated management of distributed systems," Proceedings of the Second International Workshop on Systems Management, IEEE Computer Society Press.

[23] R. Wies, "Policies in Integrated Network and Systems Management: Methodologies for the Definition, Transformation, and Application of Management Policies", PhD thesis, Department of Computer Science, University of Munich, 1995.

[24] P. McBrien, et al., "A rule language to capture and model business policy specifications," Proceedings of the Advanced Information Systems Engineering, Third International Conference CaiSE'91, number 498 in LNCS, Springer-Verlag, pp. 307-318.

[25] M. Massulo, et al., "Policy Management: An architecture and approach," Proceedings of the IEEE Workshop on Systems Management.

[26] K. Marzullo, et al., "Tools for distributed application management," Technical Report TR 90-1136, Cordinell University, Ithaca, New York, 1994.

[27] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol", RFC 1157, SNMP Research, Performance Systems International, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[28] Policy Management Working Group, http://www.ietf.org/html.charters/policy-charter.html.

[29] J. Strassner, E. Ellesson, B. Moore, "Policy Framework LDAP Schema," June 1999. <draft-ietf-policy-core-schema-04.txt>, Work in Progress.

[30] Recommendation X.500 ITU-T, Information Technology -Open Systems Interconnection- The Directory: Overview of Concepts, Models, and Services, 1993.

[31] M. Fine, et al., "Quality of Service in Policy Information Base," June 1999, <draft-mfine-cops-pib-01.txt>, Work in Progress.

[32] J. Byle, R. Cohen, and D. Durham, "The COPS (Common Open Policy Service) Protocol," August 1999, <draft-ietf-rap-cops-07.txt>, Work in Progress.

[33] H. Harroud, A. Karmouch, T. Gray, S. Mankovski, "An Agent-Based Architecture For Inter-Sites Personal Mobility Management System." Mobile Agents for Telecommunication Applications Workshop. MATA'99, 6-8 October.

[34] Mitel Corporation, Mitel-CATA (The MicMac Software Testbed), Technical Report, CITI, Adaptive Information Systems 1996.

[35] A. Hooda, "Data Management for Supporting Nomadic Users Based on LDAP and Software Agents," Master's Thesis, April 1999 University of Ottawa, Ottawa, Canada.

[36] D. Marriot, and M. Sloman, "Management Policy Service for Distributed Systems," IEEE 3rd Int. Workshop on Services in Distributed and Network Environments, Macau, June 1996.

[37] M. Genesereth, and S. Ketchpel, "Software Agents", Communications of the ACM, 37, No. 7, pages 48-53, 1994.

[38] R. Wies, "Policy Definition and Classification: Aspects, Criteria and Examples," Proceeding of the IFIP, IEEE International Workshop on Distributed Systems: Operations & Management, Toulouse, France, 10-12 October 1994.

[39] J. Hartmann et al. "Agent Technology for the UMTS VHE concept," Ericsson Eurolab, Germany 1998.

[40] R. Wies, M. Mountzia, and P. Steenekamp, "A Practical Approach Towards A Distributed and Flexible Realization Of Policies Using Intelligent Agents," Proceedings of the 8th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, October 1997.

[41] J. Bradshaw (ed.), "Software Agents" American Association for Artificial Intelligence, MIT Press 1997.

[42] R. S. Patil, and al, "The DARPA Knowledge Sharing Effort: Progress Report", Proc. of the Third International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, 1992.

[43] Bond, and Gasser, "Readership in Distributed Artificial Intelligence", Morgan Koffmann, San Mateo, CA, 1988.

[44] J. Rosenchein, and G. Zlotkin, "Rules of Encounter Designing Contentions for Automated Negotiation among Computers", MIT Press 1994.

[45] E. Friedman-Hill, (1997). *Jess, the Java Expert System Shell.* Sandia National Laboratories, Livermore, CA, 1997. Available at: http://herzberg.ca.sandia.gov/.

[46] Extensible Markup Language (XML) 1.0, W3C Recommendation REC-xml-19980210, World Wide Web Consortium, 1998. Available at http://www.w3.org/TR/1998/REC-xml-19980210.html.

[47] C. Zheng, "Design and Implementation of secure communication for a distributed mobile computing system," Master's Thesis, December 1999 University of Ottawa. Ottawa, Canada.