Software Agents and Quality of Service Issues in Distributed Systems

Mamadou Tadiou Koné

School of Information Science
Japan Advanced Institute of Science and Technology

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy.

March 2000

Dedication

To my dear mother Yayé.
To my brother Ibrahim who first taught me how to read and write.
To my sister Ramatou, brother Sékou, and niece Aminata who have always been on my side.

Abstract

The idea of a static or mobile software entity called **agent** that performs tasks on behalf of its owner in a distributed environment or the Internet is now well established. However, as the agent technology evolved, several interpretations and applications brought some confusion in its current landscape. Mobile agents, multi-agent systems, intelligent agents, information agents are some concepts that has sprung in the expanding agent research and development. In this dissertation, we use mobile agents and multi-agent systems to investigate in several steps their application to Quality of Service (QoS). Our contribution articulates around four steps:

• First, we propose an architecture for building an efficient link between a user and resource providers in a distributed multimedia environment. This architecture implements the mobile agent paradigm together with the concept of "logical disc interface" by cleary separating QoS negotiation and resource management. Here mobile agents take care of the QoS negotiation process and a **virtual host** manages (allocation and de-allocation) needed resources.

• Second, we present an agent-based QoS adaptation scheme. In this approach, we introduce the concepts of **macro-adaptation** as all the coarsed grained adjustments and **micro-adaptation** as the set of fine-grained corrective actions on resources. Here, our idea is to structure in an integrated fashion an adaptation strategy spanning all components of a distributed system.

• Third, elements of a multi-agent system need to rely on one another, enlist the support of peers through a meaningful **agent communication language** (ACL) in order to solve complex tasks. ACL is a new field that aims to foster communication, cooperation, negotiation, and improve interoperability in multi-agent systems. To this end, we examine the state of the art in agent communication languages design and suggest some principles for building a generalized ACL framework. Then, we evaluate some emerging ACL models, present some current issues and perspectives.

• Last, we propose a framework for service discovery and network QoS negotiation over large scale networks by applying the multi-agent system and agent communication language concepts. In our approach, a **user** and **service agents** engage in a structured communication through the mediation of a **QoS broker** agent and a **facilitator** agent. The facilitator agent acquires information from service agents and acts as a single point of contact to supply this information to the user via the QoS broker. To illustrate our approach, we implemented a prototype multi-agent system that discovers resources and negotiate network QoS.

Keywords: Mobile Agents, Multi-agent Systems, ACL, Quality of Service.

Acknowledgments

First, I express my sincere gratitude to Associate Professor Hajime Ishihara who invited me and together with Professor Takuya Katayama made my stay possible in the first place. I especially thank Associate Professor Tatsuo Nakajima for his assistance, helpful comments, guidance, and encouragements during my Ph.D. program. I am also grateful to Professor Futatsugi Kokichi, Associate Professors Takuo Watanabe, and Ichiro Sato of Ochanomizu University for their help. Due credits should go to members of Nakajima laboratory at the Japan Advanced Institute of Science and Technology in 1998 for the supporting atmosphere they created during my short stay.
I would also like to thank all the members of the Japan Advanced Institute of Science and Technology who directly or indirectly brought their contribution to this thesis.
I owe a great deal to the whole JAIST administration and specially to the student section and research cooperation division for the kind and constant assistance they provided. Without them, I would certainly have run into much trouble. I also thank all the hard working people at the JAIST restaurant, store, and library for their kind words and help that eased somewhat the burden of isolation and loneliness in this remote area.

Although far away in Côte d'Ivoire, West Africa, the spirit, love and prayers of my family were important ingredients to my success. Whithout the presence of God the Almighty and the support of my family, I certainly would have given up long ago.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research context

In the past decade, research and development in computer science has adopted a whole new technology for analyzing, designing and implementing software systems under the label of **agent technology**. Software agents are being proposed as a means to better cope with the increasing volume and complexity of information and computing resources. As successful applications of the agent paradigm spread across a wide range of domains from e-mail filtering to large complex, mission critical systems (e.g. air-traffic control), much confusion added to the understanding of the word **agent** itself. As a result, the simple question "what is an agent ?" hardly finds a clear answer.
However, there are two well-known perspectives in defining the word **agent**: the software engineering perspective and cognitive science (AI) perspective. The first refers to a piece of software called **autonomous or mobile agent** that can migrate autonomously inside a network and accomplish tasks on behalf of its owner as illustrated in figure 2.2 whereas **multi-agent systems** (**MAS**) are static entities with capabilities and mental attitudes. While a mobile agent migrates alone from host to host to perform tasks on behalf of its owner, an element of a multi-agent system needs its peers in order to perform tasks.
Like societies of humans, there is a need for agents in a multi-agent system to rely on one another, enlist the support and cooperation of peers in order to solve complex tasks. These agents will be able to cooperate only through a meaningful **agent communication language** (**ACL**) that can bear correctly their mental states and convey precisely the content of their messages. With the emergence of the Internet and its related services came recognition that ACLs could play an important role in the design of multi-agent systems. Several MAS prototypes for enterprise integration like ADEPT [29] (business

management), COOL [2] (supply chain management) and AMBEI [57] (manufacturing integration) have been built. The majority of actual agent applications exist in distributed environments where **Quality of Service** (**QoS**) is also an important issue and an interesting application area of the agent technology.

1.2 Research scope and contribution

This thesis aims to apply the software agent technology to QoS provision in several phases. The approach we adopted here articulates around three independent steps:

• First, an architecture to help build an efficient link between a user and the resource providers is proposed. This architecture implements the mobile agent technology and the concept behind the logical disk interface introduced in [13] by clearly separating QoS negotiation and resource management.

• Second, an agent-based QoS adaptation scheme that relies on *macro-adaptation* (coarse-grained adjustements), and *micro-adaptation* (fine-grained corrective actions), is presented. This structure spans all the configurations made of components selected by a QoS manager.

• Third, we propose in this dissertation, a framework for resource discovery and QoS negotiation on the Internet. This framework uses the concepts of multi-agent systems and an agent communication language (ACL) with the Knowledge Query and Manipulation Language (KQML) [17]. To illustrate the effectiveness of this approach, we designed a simulation system where a **user** and several **service agents** engage in a structured communication through the mediation of a **QoS Broker Agent** and a **Facilitator Agent**.
The original contribution of this research stems from using the concept of multi-agent systems and agent communication languages (ACL) to deal with QoS issues in distributed systems.

1.3 Dissertation outline

After this introductory chapter, the remaining chapters are organized as follows:

• **In chapter 2**, we present important terminology, principles, and concepts for understanding of software agent technology. The nature of mobile agents, their management, interaction, and transfer between hosts as well as the security concern they raise are introduced. In addition, we present important issues of agent standardization and interoperability.

• **In chapter 3**, we present the state of the art in agent communication languages (ACL) with current issues and perspectives related to their design. Several ACLs have been designed for specific purposes, but a common well accepted standard has yet to come. This chapter looks inside the main issues that lie on the road for a universal ACL.

• **In chapter 4**, we introduce the concept of Quality of Service (QoS), its applications and issues related to QoS provision. This chapter aims to give a brief overview of the field of Quality of Service as an interesting application area of the software agent technology.

• **In chapter 5**, we deal in several phases with the applications of agent technology to QoS provision in distributed systems. We propose some agent-based architectures for QoS negotiation and adaptation.

• **In chapter 6**, we describe the design and implementation of our multi-agent systems-based QoS negotiation framework for large scale Internet applications. In contrast to previous mobile agent architectures, multi-agent systems rely on peer cooperation through ACLs to achieve their goal. Our implementation uses the Knowledge Query and Manipulation Language (KQML) as an inter-agent communication language together with the JKQML Java API.

• **In chapter 7**, we finally conclude this dissertation with a statement of the research conducted and some perspectives.

Chapter 2

Agents Terminology, Principles, and Concepts

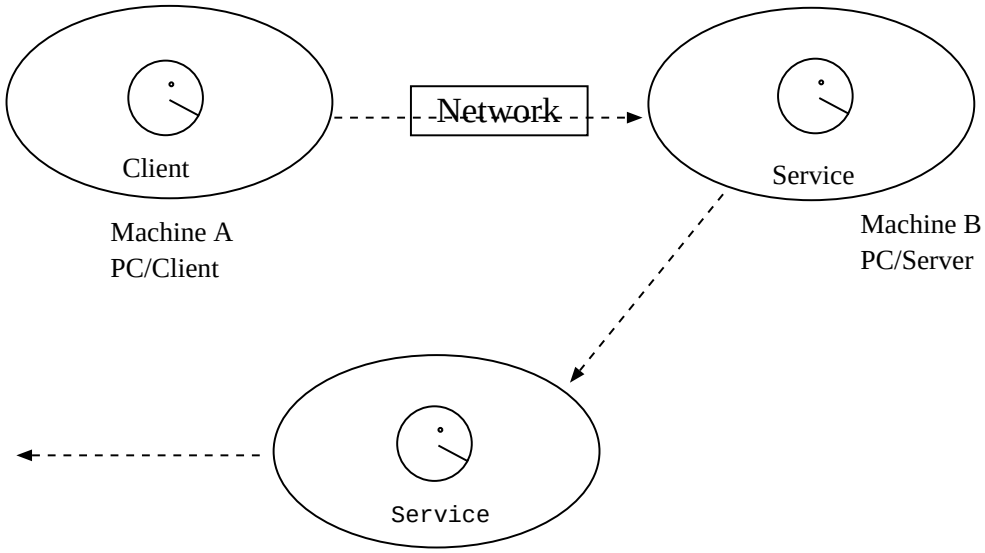
2.1 Software agents

There are two well-known perspectives in defining the word **agent**: the software engineering perspective and the cognitive science (AI) perspective. The first refers to a piece of software called **mobile agent** or **autonomous agent** that can migrate autonomously inside a network and accomplish tasks on behalf of their owners as illustrated in figure 2.2 whereas *multi-agent systems* are static entities.
Today, some applications of agent technology suggest that agents be classified as interface agents, mobile agents, co-operative agents, reactive agents, smart or intelligent agents, hybrid agents, and heterogeneous agents. In this chapter, we will focus our attention on mobile and multi-agent systems only.

2.1.1 Mobile agents

A mobile agent is a program that can migrate autonomously across a heterogeneous network. At anytime it can halt, move together with its state and data to a new machine, and resume execution at the point where it stopped. Mobile agents are a new kind of abstraction in the client/server communication world. The mobile agent technology has found many interesting applications in distributed multimedia systems, real-time systems, mobile computing, and factory automation.

Due to its unique ability to transport itself from one host to another, a mobile agent has no fixed location. A mobile agent moves to a new host where a service is available and uses it on behalf of its owner. More precisely, when an agent migrates, it enters a context within an agent system called **place**



RP (Remote Programming) paradigm with Mobile Agents

Figure 2.1: A new approach to client/server communications

where it can execute. The place of origin and the place of destination may be located inside the same or different agent systems. A mobile agent system activity is described by its:

Agents interaction

Interaction between mobile agents is achieved through simple synchronous or asynchronous message exchange. The Internet Inter Object Protocol (IIOP) and RMI are the most widely used communication protocols for agents.

Agents transfer

More pricely, when an agent migrates, it enters an agent execution environment within an agent system called a **place** where it can execute. The place of origin and the place of destination may be located inside the same agent system. A mobile agent system is characterized by its:

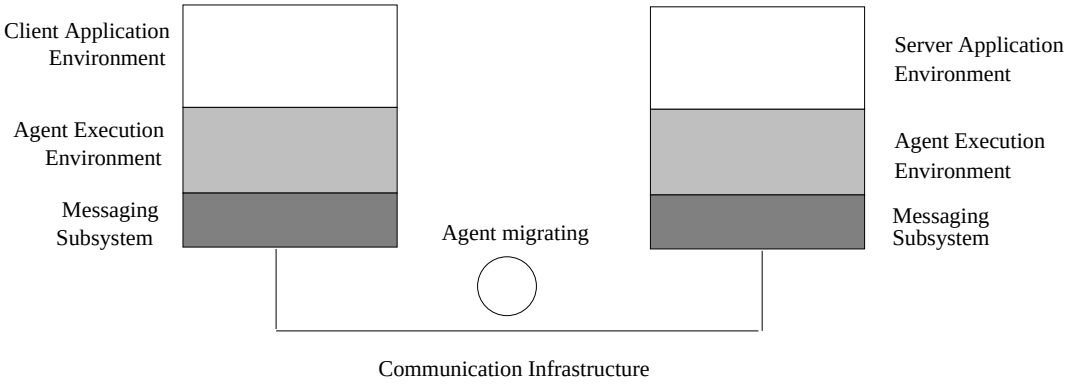


Figure 2.2: A generic agent architecture

Agent management

Agent management is concerned about the life cycle of an agent: its creation, interpretation, execution, transfer, and termination (destruction). Figure 2.2 describes a generic agent architecture. In particular, when an agent implemented in an object-oriented language intends to migrate to a known destination, the following procedure is enacted:

◇ it sends a request for transfer to the destination host system;
◇ the host system suspends the agent and
identify the elements that compose the state of the agent;
◇ the host system serializes the agent and
its state then encode it according to the current transport protocol;
◇ the agent is sent to its destination.

At the receiving end, the incoming agent is decoded, deserialized with its state restored, and its execution resumed.

Agent security

Although security and safety is not a new topic in general distributed environment, software agents bring a new kind of security and safety concern. This concern stems from some questions users ask themselves about the integrity and privacy of the message entrusted to agents as well as their authentication at the destination site. Security and safety issues are handled on two fronts: the software agent itself and the receiving host.
An agent and its content should be protected from tampering by unknown users or spoofing hosts. The fact that an agent can visit several hosts exposes its privacy and the integrity of the information gathered.

The receiving host must be protected from malicious agents because some agents have the potential of causing damage at their destination. The core principle is to isolate sensitive resources (file system, OS ...) at the host by confining the activities of incoming agents inside a carefully designed execution environment called *place*. In addition, the implementation language of the agent ought to provide some security mechanisms. The JAVA programming language, safe TCL (Ousterhout, 1996), the Telescript language by General Magic are examples of language that embeds security features that restrict access rights to agents at the host.

The inter-operability problem

The popularity and success of the agent technology led to the design of a multitude of agent systems in industry and academia. Although some of these systems may share a common implementation language, most are of different architecture and do not inter-operate at all. In order to promote interoperability and encourage diversity among agent systems designers, the Object Management Group (OMG) proposed the Mobile Agent System Interoperability Facilities (**MASIF**) standard. The OMG aims to standardize with MASIF the agent management, agent transfer, agent naming scheme, agent systems types, and agent location identification.

Example mobile agent systems

Several agent systems for commercial as well as research purposes have been built. Table 2.1 presents some agent systems.

2.1.2 Multi-agent Systems (MAS)

A Multi-agent system is a distributed computing system composed of several interacting computational entities called agents. These constituent agents have capabilities, provide services, can perceive and act on their environment. Due to an increasing number of applications, the concept of multi-agent system has expanded to a broader meaning. Nowadays, it is easily used for all types of systems composed of multiple autonomous components with the following characteristics:

- a single component is unable to solve a given problem - data is distributed across the system

hspace*.2in- communication inside the system is asynchronous

In addition, MAS provide a framework for constituents agents to interact, exchange message, and cooperate to achieve a common goal.

No.	Name	Description	Developer	Language	Applications	URL
1	Concordia	Framework for agent develop.	Mitsubishi E. I. T. USA	Java	Mobile computing data base	http://www.meitica.com/HSL
2	Aglet	Java class libraries	IBM, Tokyo	Java	Internet	http:// www.aglets.trl.ibm.co..jp
3	Agent Tcl	Transportable agent system	R. Gray U. Dartmouth	Tcl Tk	Information management	http://cs.darthmouth.edu/~agent
4	Odyssey	Set of Java class libraries	General Magic	Telescript	Electronic commerce	http://www.genmagic.com/technology/odyssey
5	OAA	Open Agent Architecture	SRI International, AI	C, Java, VB	General purpose	http://www.ai.sri.com/~oaa/main.html
6	Ara	Agent for Remote Action	U. Kaiserslautern	C/C++, TCL, Java	Partially connected c. D.D. B.	http://www/uni-kl.de/AG-Nehmer/Projekte/Ara.
7	Tacoma	Tromso and Cornel Moving Agents	Norway and Cornell U.	C, Unix-based	Client/Server issues / OS support	http://www.tacoma.cs.uit.no/
8	Voyager	Platform for distributed objects	ObjectSpace	Java	Support for agent system	http://www.ObjectSpace.com
9	AgentSpace	Agent building platform	Ichiro Sato, Japan	Java	General purpose	http://133.65.66.181/agent/index.html
10	Mole	First Java-based Agent system	Stuttgart U. , Germany	Java , UNIX-based	General purpose	http://www.informatik.uni-stuttgart.de/ipvr/vs
11	MOA	Mobile Objects and Agents	OpenGroup, UK	Java	General purpose	http://www.camb.opengroup.org/RI/java/moa
12	Kali Scheme	Distributed impl. of Scheme	NEC research Inc.	Scheme	Distributed data mining, load balancing	http://www.neci.nj.nec.com/PLS/Kali.html
13	The Tube	mobile code system	David Halls, UK	Scheme	Remote execution of Scheme	dah28@cl.cam.ac.uk
14	Ajanta	Network mobile object concept	Minoseta U.	Java	General purpose	http://www.cs.umn.edu/Ajanta
15	Knowbots	Research infrastructure of MA	CNRI , USA	Python	Distributed systems / Internet	http://www.cnri.reston.va.us/home/koe
16	AgentSpace	Mobile agent framework	Alberto Sylva, Brazil	Java	Support for dynamic application	http://www.berlin.inesc.pt/agentspace
17	Plangent	Intelligent agent system	Toshiba Corporation	Java	Intelligent tasks	http://www2.toshiba.co.jp/plangent/index.html
18	JATLite	Java agent framework dev./ KQML	Stanford U.	Java	Information retrievial, Interface agent	http://java.stanford.edu
19	Kafka	Multi-agent libraries for Java	Fujitsu lab. Japan	Java, UNIX-based	General purpose	http://www.fujitsu.co.jp/hypertext/free/kafka
20	Messengers	Autonomous messages	U. California Irvine	C (Messenger-C)	General purpose	http://www.ics.uci.edu/~bic/messengers

Table 2.1: Example of Agent systems

Agents interaction

Like societies of humans, there is a need for agents in a multi-agent system to rely on one another, enlist the support of peers in order to solve complex tasks. Agent communication languages (ACL) are at the core of agent interaction and communication. One of the main objectives of ACL design is to model a suitable framework where heterogeneous agents can interact, communicate in meaningful statements.

Example applications of MAS

To date, several industrial applications of MAS exist in manufacturing, telecommunications, and commercial applications include information management and electronic commerce. In addition, MAS are being applied in areas like games and medicine (patient monitor, health care). Several MAS prototypes for enterprise integration like ADEPT [29] (business management), COOL [2] (supply chain management) and AMBEI [57] (manufacturing integration) exist. In addition, agent-based software engineering promises to be an interesting application area. Several industrial, commercial, and medical applications of

MAS exist.
Industrial applications

- **Manufacturing**: YAMS (Yet Another Manufacturing System) is a multi-agent system that applies the Contract Net protocol to manufacturing control. A manufacturing enterprise is modeled as a hierarchy of workcells that are further grouped into flexible manufacturing systems (FMS). A collection of such FMS constitutes a factory seen as an element of an organization of a company. The goal of YAMS is to manage efficiently the production process at these plants. Factories and their components are modeled as agents that have plans representing their capabilities.

- **Process Control**: **ARCHON** is an agent-based process control application for building multi-agent systems. ARCHON has been applied to electricity transport management and particle accelerator control. This system is said to be one the world's first field-tested multi-agent systems.

- **Telecommunications**: In order to monitor and manage in real-time large distributed telecommunications networks, negotiating agents have been used by Griffeth and Velthuijsen to represent entities involved in a set up of a call. In addition, network management and control, transmission and switching.

- **Air Traffic Control and Transportation Systems** are other successful industrial applications of MAS in industry.

Commercial Applications

- **Information Management**: Our modern society is characterized by its huge amount of information that we must process in a short time. This leads us to what is known as the *information overload*. In order to alleviate users workload, direct , *information filtering* and *information management* is being carried out successfully by agents in a number of systems:
Maxims is an electronic mail filtering agent which "learns to prioritize, delete, forward, sort, and archive mail messages on behalf of a user".
WARREN is financial portfolio management multi-agent system that finds and filters information in a financial context. This system is made of agents that cooperatively self-organize to monitor and track stock quotes, financial news, and company earnings reports.

- **Electronic Commerce**: Commercial decision making can be placed in the hands of agents. Some of systems create an environment where buying and selling are conducted by agent on behalf of user. Other applications include BargainFinder and MAGMA a virtual marketplace for electronic commerce.

- **Business Process Management**: **ADEPT** for the management of business processes and **AMBEI** supply chain management have been designed. ADEPT deals with a business process as a community of negotiating and service providing agents.

There are also applications in the areas of entertainment, games, interactive theatre, and medicine.

2.2 Summary

In this chapter, we have presented mobile agents, multi-agent systems, issues related to these technologies, and some of their applications in industry. Although both technologies share the important interoperability problem, they have different approaches in dealing with it.

Chapter 3

The State of the Art in Agent Communication Languages

With the emergence of the Internet and its related services came recognition that agent communication languages (ACL) could play an important role in the design of multi-agent systems (MAS) and agent-oriented software integration.
An agent language stems from the need for better problem solving paradigms in distributed computing environments. One of the main objectives of ACL design is to model a suitable framework that allow heterogeneous agents to interact, communicate with meaningful statements that convey information about their environment or knowledge. In addition, in the process of solving a complex task, an agent may need to cooperate in a concise language with other agents able to bring their contribution. ACLs evolve around the key concept of *communicative act* (CA) from Speech Act theory [56]. A communicative act is a special type of action realized by sending a message.
Several ACLs (KQML, FIPA ACL, OAA ICL, ARCOL, Agent-0, PLACA, LOGOS) have been implemented. However, due to their specific approaches, agents from different environments cannot communicate. In order to build a consensus, the Foundation for Intelligent Physical Agents (FIPA) is designing a standard [19] that promises to bring into a common fold future ACLs in industry and academia. However several issues related to semantics, interaction protocols, and ontology remain. This chapter attempts to present the state of the art in agent communication languages (ACL) from the engineering as well as the cognitive science perspectives.

3.1 Generalized ACL Framework

3.1.1 ACL design principles

To date, the design of ACLs has evolved around several principles like heterogeneity, interoperability, transparency, extensibility, cooperation and coordination, and performance. An agent communication language could be of value to a wide number of application areas if these principles are observed. A generalized ACL framework may be characterized by:

- *the heterogeneity principle* states that agents should be able to communicate regardless of their implementation environments. The meaning of the message exchanged should be context independent and reflect a global perspective rather than the sender or receiver private perspectives;
- *the cooperation and coordination principle* states that effective cooperation to solve a complex task requires a meaningful communication language. Generally, communication is one of the means of conveying, exchanging information about an agent knowledge or environment. An ACL should give agents the means to rely on one another, enlists the support of other agents in order to achieve goals. The actual application of this principle depends on the design of an appropriate interaction or negotiation protocol for a given task. These protocols are high-level protocols different from message transport mechanisms and are intimately linked to their context. In a multi-agent system, when a complex task must be subdivided between several agents, negotiation occurs in the assignement and gathering of partial results for a complete solution. The building blocks of a negotiation mechanism as suggested in [52], are the definition of a domain of interaction, the negotiation protocol with the rules of interaction, and negotiation strategies;
- *the separation principle* states that a message content, structure, and transport mechanism are distinct entities that should be handled separately;
- *the interoperability principle* states that an ACL should provide heterogeneous agents with the means to inter-operate. For example, agent-based software integration was conceived to help heterogeneous software components modeled as agents inter-operate in an appropriate language;
- *the transparency principle* states that multi-agent systems should be shielded from the complexities of the underlying ACL specifications. An appropriate ACL API should free agents from specific details and set interactions to a higher level. With ACL transparency, the underlying transport protocol defines message handling and MAS need not embed any additional functionality;
- *the extensibility and scalability principles* states that ACL designers may add new communicative acts compatible with the existing ones. These new types may implement defined interaction protocols. In addition, the design of

an ACL should take into account any scalability issue related to the growing number of agents in a multi-agent system;

• *the performance principle* states that an ACL implementation should use efficiently system resources (CPU, memory, and bandwidth). The primitive communicative acts supplied with the language should be compatible with the underlying network technology and exhibit unicast, multicast as well as synchronous and asynchronous connection capabilities. In addition, an ACL must support reliable, safe and secure message exchange between agents.

3.1.2 ACL specifications

ACL specifications are concerned with the description of a message structure, its semantic model, and the underlying interaction protocols. These specifications define a language and supporting protocols and encompasse:

• *the message format* defines the primitive communicative acts and message parameters (sender, receiver, message-id, protocol and language) with expressions that describe actions at the content, message, and communication layers. In particular, the message content describes facts, actions, or objects in any content language. Other parameters could deal with the message meaning (ontology) and delivery. In addition, the ACL supplies the users with a finite set of primitive communicative acts;

• *an ACL semantics model* lays down the foundation for a concise and unambiguous meaning of agent messages and depends on the interactive behavior and capabilities of these agents. When agents interact or cooperate to achieve a goal, the mutual understanding of the messages exchanged depends on the semantics given to communicative actions. Approaches in providing a semantics to an ACL are based on mental concepts of belief, desire, and intention ([36], [6] and [18]) or social agency [61].

• *interaction protocols* (or *conversation policies*) are sets of well-defined patterns of interactions designed to facilitate inter-agent communication. Although protocols are optional, agent communication must be consistent with a chosen protocol. A number of protocols like the followings, are used in ACL design:

∗ *Direct communication protocol* is applied when a sender agent knows the receiving agent and its capabilities;

∗ *The Contract Net protocol*, originally designed by Davies and Smith in [10] in its generic form, sets the interaction patterns between an agent (*the manager*) who enlists the support, through *a call for proposals*, of a number of other agents (*contractors*) to perform some complex task. Contractors submit proposals to the manager who evaluates and assigns tasks under some conditions. The successful contractors commit themselves to performing the

assigned task and sending back the result to the manager;

∗ *The mediated communication protocol* uses the services of special agents (facilitators) that act as brokers between agents in need of some service and other agents that provide them. Mediation involves needy agents subscribing to services; facilitators brokering, recruiting, and recommending agents that registered their identity and capabilities.

• *Shared ontologies and content language* are prerequisites to knowledge sharing because the ability to exchange messages doesn't assume the understanding of their content. ACL designers share the same concerns in ontology development with researchers in the fields of Knowledge-Based Systems and Natural Language Processing (ontological engineering). A good ontology should display as agreed by Mahesh in [41] the following features.

- *Dependency and relevance*: An ontology must be domain-dependent and its taxonomy and relationships should show clearly their relevance to that domain. For example, ontologies for education and conference should not share elements;

- *Coverage*: For practical reasons, an ontology should have a broad coverage of its domain and must be independent of the message content language. This point is specially important when this ontology is being shared by multiple agents in several contexts;

- *Simplicity and clarity*: The organization of an ontology should be simple and easy to process by an agent and accessible to a human reader. The inheritance structure should not be tied to a particular context nor formalism (e.g. first order logic);

- *Extensibility*: A good ontology should be extensible, concise, versatile and incorporate fine-grained concepts. This extensibility will give room for improvement and allow designers to add incrementally new elements as conditions change.

• Generally, services like security, and transport for an agent system are provided by the underlying communications infrastructure. Agent communication languages should provide to interacting agents the means to specify their requirements for the quality of network communications. These requirements may include:

- *Integrity and confidentiality*: for the type of data encryption, integrity checks during network communications;

- *Authentication*: the sender agent may require authentication of the receiver before actually sending the message. Authentication services are normally provided in secure communications infrastructures.

3.2 Existing ACL Models

The design of ACLs has adopted several approaches: the declarative approach (e.g. KQML, ARCOL, FIPA, and AOP), the procedural approach (e.g. ICL and Mobile Agent Communication), and the "social agency" approach. Theses approaches, depending on their implementation environments, exhibit strengths and weaknesses. We introduce and briefly assess in the following subsections well known agent communication models.

3.2.1 The KQML Communication Model

The Knowledge Query and Manipulation Language (KQML) is a versatile, general-purpose language that supports communication between several agents with a set of reserved primitives called *performatives*. KQML described by Finin in [21] is the result of research done by the Knowledge Sharing Effort(KSE) [17], an initiative that aims to develop a foundation for software systems interaction and interoperability. Three working groups with complementary objectives compose this consortium: *the Interlingua group* designed the Knowledge Interchange Format (KIF) as a common language for describing a message content, *the Shared and Reusable Knowledge Base group* is concerned about the content of sharable knowledge bases, and the *the External Interface group* produced the KQML language and looks at interactions of system components at run time.

In KQML, an agent's mental attitudes (belief, intention, commitment, choice) are expressed in the message that represents a communicative act. These communicative acts are called performatives and define the permissible operations that an agent can conduct. The content of a KQML message is described in KIF whereas its format is expressed with suitable performatives. The complete list of KQML reserved performatives and their meaning is illustrated in table 3.1.

Table 3.1: KQML reserved performatives

Category	Name	Meaning
Discourse	ask-if	S wants to know if the :content is in R's VKB
	ask-one , ask-all	S wants one or all of R's instantiations of the :content that are true of R
	stream-all	Multiple version of ask all
	eos	End-of-stream marker to a multiple-response (stream-all)
	tell , untell	S wants to inform R about a sentence in or not in its VKB
	deny	S wants to inform R that a sentence is not in its VKB
	insert , uninsert	S asks R to add the content to its VKB , undo a previous insert action
	delete-one , delete-all , undelete	S wants R to remove one or all matching sentences from its VKB, or undo a previous delete
	achieve , unachieve	S wants R to make something true in its environment, undo a previous achieve
	advertive , unadvertise	S informs R of its ability and willingness to process inquiries in content
Intervention and Mechanics	subscribe	S wants regular updates about a performative from a facilitator
	error	S considers R's previous message as ill-formed
	sorry	S understands R's message but cannot provide further help
	standby	S wants R to announce its readiness to provide a response to the message in the content
	ready	S is ready to respond to a previous message from R
	next	S wants R's next response to a previous message it sent
	rest	S wants R' remaining reply to a previous message
	discard	S ignores any remaining response to a previous multi-response message
Facilitation and Networking	register , unregister	S announces to F (facilitator) its presence and symbolic name, reverse a previous register act
	forward	S wants F to forward a message to another agent
	broadcast	S wants F (facilitator) to broadcast a message to all agents that it knows
	transport-address	S associates its symbolic name with a new transport address
	broker-one , broker-all	S asks F to find an agent able to provide one or all responses to a <performative>
	recommend-one , recommend-all	S wants F to recommend to him an agent that is able to perform a <performative>
	recruit-one , recruit-all	S wants F to find one or all suitable agents able to respond to a <performative>

S : sender agent R : receiver agent F : facilitator VKB : agent's virtual knowledge base

A simple scenario of conversation in KQML about a call for papers for a conference where C is the conference chairman agent, R is the receiver agent, and A is one of R's acquaintances (R included), would be as follows:
The conference chairman sends (*broadcasts*) a call for papers to interested agents like R who eventually forwards (*tell*) the message to their acquaintances like A.

```
(broadcast
        :sender     C
        :receiver   R
        :reply-with id0
        :language   KQML
        :ontology   kqml-ontology
        :content    (tell
                      :sender     C
                      :receiver   A
                      :reply-with id0
                      :language   Prolog
                      :ontology   conference
                      :content    ``call(papers,conf)''
        )
  )
```

Following the reception of the forwarded call for papers to the conference, agent A submits (*tell*) his paper to the conference chairman agent C.

```
(tell
    :sender      A
    :receiver    C
    :in-reply-to id0
    :language    Prolog
    :reply-with  id1
    :ontology    conference
    :content     ``submit(paper,conf)'')
```

After reviewing submitted papers, C informs (*tell*) A of the result.

```
(tell
    :sender      C
    :receiver    A
    :language    Prolog
    :in-reply-to id1
    :ontology    conference
    :content     ``accepted(paper,conf)'' )
```

KQML is made of three layers: the communication layer (sender, receiver, and message id), the message layer (performatives and message format), and the content layer (ontology, content language, and message content). There

are mainly two types of agent communication: direct and mediated communications. In the first scheme, an agent sends directly messages to known agents able to process them. In the second scheme, an agent makes requests for services to a special agent called a **facilitator**. The role of a facilitator is to coordinate the interactions of agents involved in a particular problem solving by

- forwarding requests to appropriate agents,
- forwarding updates to agents that have subscribed to particular events,
- brokering, recruiting, and recommendingsuitable agents.
- initiating distributed problem solving between agents.
- content-based routing and smart multicasting.

All concerned agents are expected to register their identities, interests and capabilities with a facilitator depicted in figure 3.1. In a KQML communication environment, a *K-router* handles incoming messages which are further processed by the KQML Router Interface Library (KRIL) API.

A number of applications of the KQML language ranging from the engineering of hardware and software systems to database systems and technology integration experiments have been conceived. One of the most promising applications is the Agent-Based Software Integration (ABSE) approach [23] where integration and inter-operation between software components is achieved with facilitator agents.

Advantages and limitations of KQML

The main advantage of KQML is its ability to support a wide range of agent architectures with its extensible set of performatives. A message that is opaque to its content offers an attractive feature. It is with this language that the concept of agent communication language made of distinct and independent layers was first defined. As a result, KQML became the *de facto* standard for agent communication in several areas. However, the early KQML had some critics like Cohen and Levesque in [6] point out to a confusion in the usage of its performatives. Partly due to this weakness, a number of KQML dialects arose in industry and KQML-speaking agents implemented in different environments cannot communicate. Fortunately, the new unofficial KQML specification in [37] is making significant improvements to its semantics and set of performatives.

3.2.2 The ARCOL Communication Model

The *ARTIMIS* agent technology developed by France Telecom described in [54] is a generic framework for instantiating communication-enabled agents.

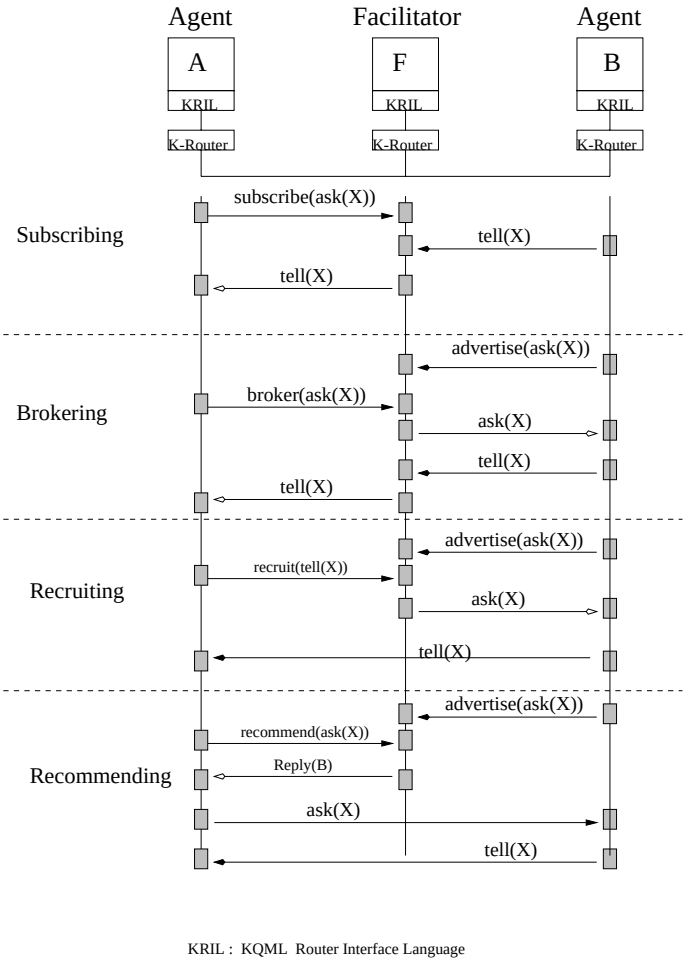


Figure 3.1: Facilitator mediation in KQML

In ARTIMIS, illustrated in figure 3.2, an agent can cooperatively interact with humans as well as with other agents. Here, agents' communicative acts are modeled as normal "rational actions". In other words, the rational unit at the heart of ARTIMIS enables agents to reason about knowledge and plan actions pertaining to their communicative acts. *ARCOL*[1] (*AR*TIMIS *CO*mmunication *L*anguage) is the inter-agent communication language used in the ARTIMIS system to define agents' communicative acts. An ARCOL expression relies on the language called *SL* (Semantic Language) for the definition of its semantics. SL in turn uses the language *SCL* (Semantic Content Language) to describe the semantics content of a communicative act. All primitives communicative

[1] http://www.drogo.cselt.stet.it/fipa/cfp1

acts of ARCOL are shown in table 3.2.

An agent i who believes a proposition p would like to inform another agent j of p is expressed as: $< i, INFORM(j, B_i\ p) >$.
ARCOL contains the following set of mutually exclusive primitives:

•*Inform*: An agent uses the assertive *Inform* to convey a message to another agent provided it believes the content of this message itself. In ARCOL, sincerity is a sine-qua-non condition for all communicative acts.

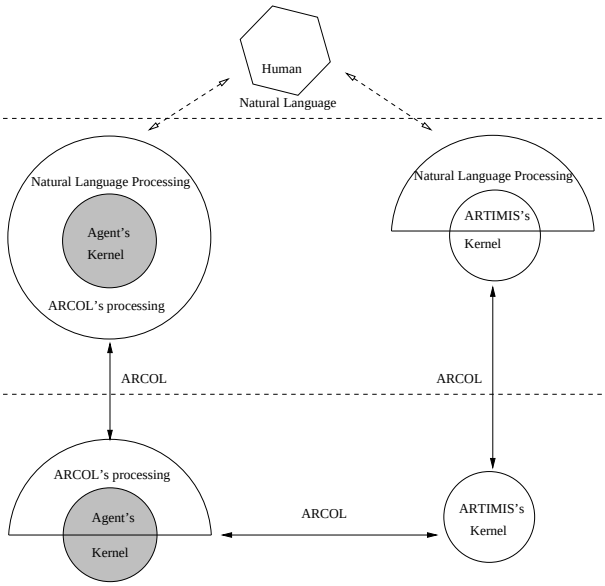


Figure 3.2: Structure of an ARTIMIS system (©1997 D. Sadek)

• *Request*: The directive *Request* enables an agent to impose an action on another agent provided it has the capabilities to perform that action.

•*Confirmation* (opposite *Disconfirmation*, like the Inform primitive, requires sincerity from an agent. When the sender believes that the receiver is uncertain about the property being transmitted, the communicative act becomes *context-relevant*.

•*Inform Referent* enables an agent to inform another agent of the value of a referent with a given description. These CA are mutually exclusive.

The goal or intention of an agent in performing a communicative act is called the *Rational Effect* (*RE*), whereas the pre-requisites to this act are known as the *Feasibility Preconditions* (*FP*) composed of the *ability preconditions* and the *context relevance preconditions*.

Table 3.2: ARCOL primitives

Primitives	Inform	Request	Confirm	Disconfirm
Qualifications or Feasibility Preconditions (FP)	A believes p A believes that B is not Uncertain about p A believes that B beleives the opposite of p	FP of a regarding A's mental attitudes A believes that B is able to accomplish p A believes that the accomplishment of a is a persistent goal for B	A believes p A believes that B is uncertain about p (context relevance)	A believes p A believes that B is uncertain about p or B believes p
Goal or Rational Effect (RE)	B believes p	Action a accomplishment	B believes p	B believes p

Agents : A (actor) and B (interlocutor) Agent mental attitudes : Belief (B) , Uncertainty (U) , Choice (C) , Goal (G) a is a communicative act p is a proposition

Advantages and limitations of ARCOL

The most important feature of the ARCOL language is its formal semantics as a reliable support for interoperability. In addition, ARCOL has a small set of primitives that can be combined. However, according to Singh in [61], ARCOL's fixed context with the sender agent required to be *sincere* is an impediment to heterogeneity. ARCOL messages not only have limited coverage but also ignore their community public perspective. Indeed, the performance conditions imposed by ARCOL restrict the scope of its practical applications.

3.2.3 The FIPA Standard ACL

The *Foundation for Intelligent Physical Agents* (FIPA) [2] is an international organization that aims to develop a set of generic agent standards with the contribution of all parties involved in agent technology. In particular, the FIPA standard for ACL attempts to identify the practical components of inter-agent communication and cooperation and define a concise formal semantics and supporting communication protocols. In fact, the main FIPA standard specification [19] is composed of 7 sub-specifications: agents management, agents communication, agents interaction, personal travel assistance, personal assistance, audio-visual entertainment and broadcasting, and network management and provisioning. The core of the FIPA specification for agents communication - communication primitives (performatives), formal model, and content (*Semantic Language SL*) - draws from the ARCOL language in [54] designed by France Télécom whereas KQML inspired the structure of its messages.
Agents who comply to the FIPA standard (FIPA ACL agents) are required

[2] http://www.fipa.org

Table 3.3: FIPA primitive communicative acts

Category	Name	Meaning
Action	Agree	Action of agreeing to perform some action, possibly in the future
	Refuse	Action of refusing to perform a request and providing reasons for this attitude
	Cancel	Action of cancelling some previously requested action with temporal extent
	Propagate	Sender requests Receiver to forward message to known agents satisfying some condition
	Proxy	Sender requests Receiver to forward message to agents that satisfy some condition
	Request	Sender requests Receiver to perform action
	Request-when	Resquest when p becomes true
	Request-whenever	Request whenever p becomes true
	Request-whomever	Request to any agent other than self to perform an action or find an agent able to perform it
Information	Inform	Sender informs Receiver that a given proposition is true
	Inform-if (macro-act)	Macro action to inform Receiver about the object that corresponds to a descriptor (name)
	Inform-ref (macro-act)	Macro action to inform Receiver about the truth value of p
	Confirm	Sender informs Receiver about the truth of p when Receiver is uncertain
	Disconfirm	Senders informs Receiver about the falsity of p when Receiver might beleive that it is true
Negotiation	Cfp	Action of calling for proposals to perform a given action
	Propose	Action of submitting a proposal for an action under some pre-conditions
	Accept-proposal	Action of accepting a previously submitted proposal
	Reject-proposal	Action of rejecting a proposal to perform an action during a negotiation
Request	Subscribe	Request of information and updates from the Receiver about value of a referenced object
	Query-if	Action of asking an agent about the truth value of p
	Query-ref	Action of asking an agent for an object refered to by an expression
Error	Not-understood	Sender informs Receiver that it doesn't understand the meaning of the previous message
	Failure	Action of informing that an action failed

S : sender agent R : receiver agent F : facilitator VKB : agent's virtual knowledge base :content and :to are keywords

to send correct, concise and unambiguous messages that follow consistently selected protocols. Communicative acts are organized in the following 5 categories in table 3.3.The previous conference scenario is expressed as follows in the FIPA ACL language:
First, agent C - chairman - sends (*cfp*) a call for papers to an agent R.

```
(cfp
 :sender      C
 :receiver    R
 :reply-with  call-proposal
 :language    sl
 :ontology    conference
 :protocol    FIPA-Contract-Net
 :content     ((action  R (submit(paper,conf)) ) true) )
```

Agent R submits his paper for review to agent C.

```
(propose
     :sender      R
     :receiver    C
     :in-reply-to call-proposal
     :reply-with  proposal-R
     :language    sl
     :ontology    conference
     :content     (action (submit(paper,conf)))
)
```

Agent C informs (*accept-proposal*) agent A that its paper has been accepted for the conference.

```
(accept-proposal
            :sender       C
            :receiver     R
            :in-reply-to  proposal-R
            :language     Prolog
            :ontology     conference
            :content      ``(accepted( R, paper ,conf))''
)
```

Advantages and limitations of FIPA ACL

FIPA-ACL is an agent communication language that involves several parties in industry and academia. As a standard, FIPA-ACL lays out clearly the practical components of inter-agent communication and cooperation and a well-defined formal semantics foundation. This feature makes it a concise reference point.

In contrast to other well-known ACL models, the FIPA-ACL suffers from a lack of practical applications that could point out its strengths or limitations.

3.2.4 The Agent Oriented Programming (AOP) Model

In 1989, Yoav Shoham introduced a new programming paradigm called *agent-oriented programming* (AOP). This paradigm described in [58], is a computational framework that draws from Artificial Intelligence (AI), Speech Act

theory, and Object Oriented Programming (OOP). According to Shoham, "*an agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments*". A complete AOP system should include three primary components:

• A concise formal language to express an agent's communicative act with its mental states.

• An interpreted language with a reserved set of performatives that include agents' mental states.

• A means to model devices into agents.

There are only three primitive communicative acts in the AOP model: *inform*, *request* and *unrequest* that can be combined to express a variety of actions.

Here agents act under the constraints of internally consistent and persistent beliefs, commitments, current and prior decisions. In addition, an agent could have commitments towards others that it cannot revoke although it can cancel its own decisions.

In the line of these principles, Torrance [66] designed the **AGENT-0** programming language to specify agents with their capabilities, beliefs, commitments together with a set of commitment rules. Agents' beliefs consist of a set of facts bound to a proposition at a particular instant in time whereas a commitment is a particular proposition. A communicative act is carried out if and only if all the three parts - message condition, mental condition, and action - of the related commitment rule are satisfied.

Advantages and limitations of AOP

The idea of ascribing mental attitudes to software agents in the AOP model is indeed unique and attractive. However, the small number (only three) of primitives communicative acts available makes the AOP model unfit for most real-world applications. According to David Parks in [16], AOP is still an underdeveloped field because this theory doesn't address the important issues of:

- formal description of the underlying mental theory,
- security in heterogeneous networks,
- consistency of beliefs and actions across implementations.

The AOP theory needs an expressive implementation language in order to become a complete programming paradigm.

3.2.5 The SRI OAA Communication Model

SRI International [3] designed the **Open Agent Architecture** (OAA) described in [42] as a means for integrating a community of software agents in a

[3] http//www.ai.sri.com/∼ oaa

distributed environment. To support this architecture, SRI conceived a logic-based declarative language called **InterAgent Communication Language** (ICL). This language is designed to fit a broad concept of agent in OAA where software agents and human users - privileged agent members - interact, share services in a dynamic community via facilitators. In this architecture, the term communication covers inter-agent as well as human-agent interactions.
Information exchange, activities, and requests of constituent agents are processed as events represented in the OAA by *ICL expressions*. For example, a request expressed in natural language as "Inform all about the Agent Communication Language workshop" is translated in an ICL expression as *send_message(email, 'all', [subject(ACL workshop)])*. An event has a type, includes parameters and a content. In the OAA framework, ICL is best described as a middleware which helps implement cooperation mechanisms between service providers (servers), service requesters (clients), and facilitator agents. It handles temporal constraints and supports only three speech act types: *Solve*, *Do*, and *Post*. The ICL language is structured around two layers:
- the conversational protocol layer describes the event types and related parameter lists;
- the content layer which is an extension of the PROLOG programming language, describes the content (goals, triggers, and data elements) of events of a conversation.
Many applications like the Automated Office, the Multimodal Map, MVIEWS, the Agent Development Tools, and Language Tutoring have been build with the OAA architecture and ICL.

Advantages and limitations of OAA ICL

The OAA framework has adopted the procedural approach in designing its inter-agent communication language ICL. In addition to being efficiently executable, this language is simple and exhibits powerful features like the ability to handle parallel solving of a list of persistent and distributed goals. Its wide perspective makes it an excellent communication medium between sofware agents on one hand and human users and software agents on the other hand. Unfortunately, the ICL language is tied to a particular agent architecture (OAA) and its procedural nature does not allow a bi-directional conversation. In addition, there is no mechanism for the variety of agents (software and human) to share a common ontology or merge vocabulary and knowledge on a large scale. Although the idea of incorporating natural language may be appealing, the development of the OAA architecture may have to face all the issues known in Natural Language Processing.

3.2.6 The Mobile Agent Communication (MAC) model

A mobile agent is a program that can migrate autonomously across a heterogeneous network. At anytime it can halt, move together with its state and data to a new machine, and resume execution at the point where it stopped. Mobile agents are a new kind of abstraction in the client/server communication world. The most common communication mechanism for mobile agents is message passing (requests, queries). In addition, some alternatives like remote procedural calls (RPC), remote method invocation (RMI), and the Commmon Object Request Broker (CORBA) in specific cases for distributed applications. The mobile agent technology has found many interesting applications in distributed multimedia systems, real-time systems, mobile computing, factory automation and mission-critical systems.

Unlike AI, software engineering makes no assumption about mental states nor social structure of mobile agents. Here, agents are only processes that are able to move autonomously and perform tasks on behalf of a user at remote locations. This property leads to a difference in the way agent communication is defined as illustrated by the following example.

The Aglet communication model

Aglets (**Ag**ile app**lets**) are Java objects designed at IBM Tokyo Research Center [4]. They can migrate together with their data, code, and states autonomously inside a computer network and execute work on behalf of their owner. Agents communicate with *message passing* (local host) or *remote message passing* (remote host) like in figure 3.3. These messages are objects (*message objects*) that are serialized, marshaled, sent, then stored at the receiving end in a message queue. The programming principles of Java mobile agents are explained by the designers of this technology in [39]. They can be a *now-type* (synchronous message), a *future-type* asynchronous message, an asynchronous *one-way type* message.

Advantages and limitations of MAC

Agents that are able to migrate certainly offer an attractive alternative to common RPC communication style between hosts. Inter-agent communication is however limited to simple requests. Due to the characteristics of mobile agents, there is neither exchange of information nor sharing of domain knowledge. The aim of communication here is not to seek the cooperation of other agents in solving a task but rather to use services provided at the destination host. As

[4] http://www.trl.ibm.co.jp/aglets

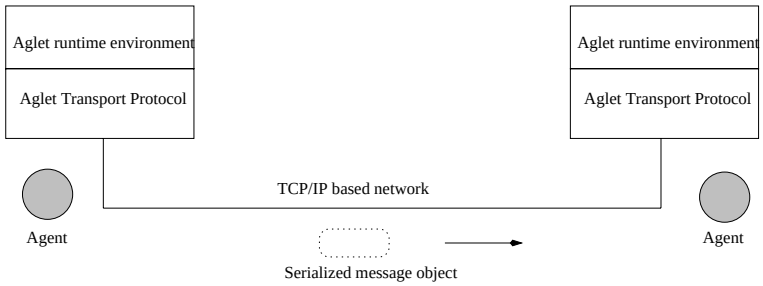


Figure 3.3: The Aglet architecture and communication model

there is no concept of community of agents, one may wonder if mobile agents *really* communicate.

3.2.7 The "Social Agency" Communication Model

In conventional multi-agent systems based on mental attitudes like belief, intention, and commitment, agent communication suffers from the lack of a concise and universally accepted formal semantics. As a result, agent communication is confined in the realm of restricted environments and heterogeneous agents do not interact. As agents are commonly defined in terms of autonomy, interoperability and their ability to cooperate, coordinate actions towards a common goal, mental attitudes alone may not help define a concise formal semantics. A number of attractive solutions to this drawback has been proposed recently by Singh in [61], Colombetti in [8] and Moulin in [43]. Singh advocates a formal semantics model that emphasizes *social agency* whereas Colombetti proposes modal logic as a basis for the definition of the semantics of agent communication language.

Singh's "social agency" model

Singh proposes a theory that provides a concise foundation for the design of multi-agent systems. In this theory, he shows that the success of multi-agent systems depends on how well the underlying ACL supports interaction among agents in a social setting. In [61], he suggests that many elements depicted in figure 3.4 contribute to the meaning of communication between agents:

◇ *perspective* could be private (sender's) or public (agent society's): the message sent contains knowledge and attitudes about senders only or some shared knowledge of the multi-agent system.

◇ *type of meaning* is individual or conventional (group of agents),

◇ *basis* deals with semantics or pragmatics,
◇ *context* should be flexible to make agent communication more meaningful,
◇ *coverage* of communicative acts should be wide (including all categories of Speech Act theory) in order to improve interactions in a multi-agent system. In addition, Singh introduces in [62] a social semantics for ACL based on social commitments within temporal logic.

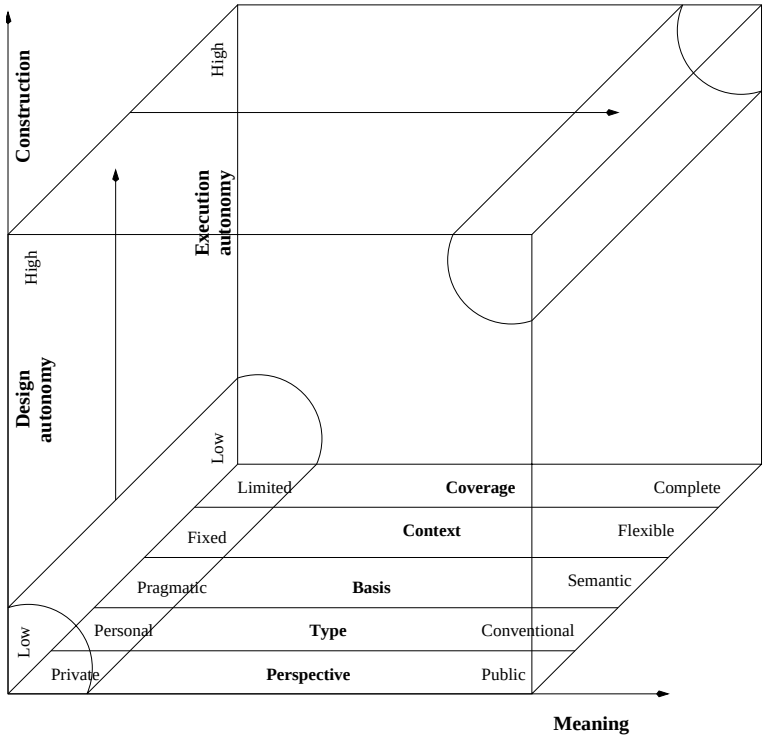


Figure 3.4: The design space of agent communication languages (©1998 IEEE): *The region in the left represents existing ACLs, which follows a mental agency model. The region in the upper right represent the desired goals, which dictate a social agency model: high design and execution autonomy, high coverage (include all significant categories of communicative acts), flexible context, semantic basis for meaning, conventional meaning type, and a public perspective.*

Colombetti's modal-logic-based approach

Colombetti proposes in [8] the idea of agents with *"social mental states"*. With modal logic, he defines concepts related to social interactions of agents. These

interactions could create common beliefs expressed in different ways as illustrated in figure 3.5. In order to establish common beliefs in a group of agents, he suggests that one or several of the following mechanisms should be considered:

- ◇ *Deduction*: knowledge can be inferred from another knowledge,
- ◇ *Displayed information* is perceived by all agent when an event takes place,
- ◇ *Mutual observation* between agents is a way to acquire new knowledge,
- ◇ *Intentional communication* is public and implies common beliefs.

In addition, he analyzes in [7] the social commitment entailed by the performance of communicative acts.

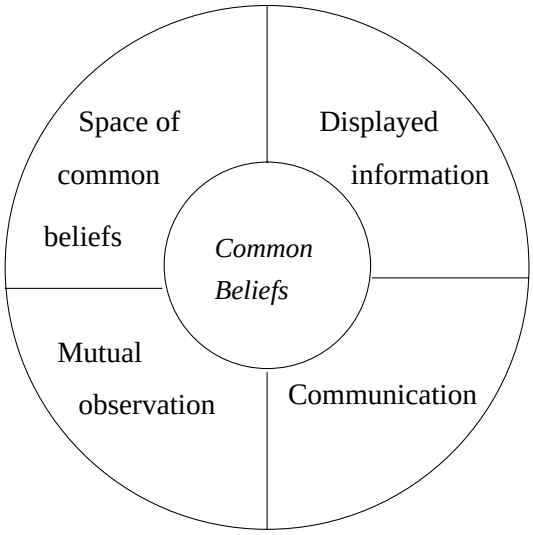


Figure 3.5: Common beliefs in agent communication.

Advantages and limitations of the social agency model

The social dimension of multi-agent systems is a valuable asset for the design and management of a wide range of these systems. The theoretical foundation of the social agency model laid down by Singh, Colombetti and Moulin may prove to be an important tool for the new field of agent-based software integration. This approach is a macro theory that may be difficult to implement in real world environments because computation models impose constraints that the social agency model may not accommodate.

3.2.8 Other Communication Models

The *COOL* language: COOL in [2] is a language for multi-agent systems designed by the Interprise Integration Laboratory at the University of Toronto and the Technical University of Berlin. It is based on WinterP (made of Lisp) and addresses the issues of agent interaction, representation, reasoning and legacy software integration.
The *LOGOS* language: LOGOS is an agent communication language designed at the NASA. It is written as a set of two Java classes describing its structure and content and contains four message types: `COMMAND`, `REQUEST`, `REPLY`, and `INFORM`. Several primitives communicative acts specific to air craft control are provided.
Agent-0 derived languages: *AgentK*: Davies and Edwards in [9] conceived an interesting integration of AGENT-0 and KQML called *Agent-K* . Like *AGENT-0*, *AGENT-K* has no planing features. To correct this, Thomas in [65] designed *PLACA*, a language that enables agents with intentions and planing capabilities.
The *APRIL* and *MAIL* languages: These two languages stem from the ESPRIT project IMAGINE. APRIL is a general purpose language for the design of multi-agent systems with multitasking, communication, and pattern matching properties whereas MAIL is a system that provides plans for agents and was intended to use APRIL as its implementation language.

3.3 Models Comparison and Current Issues

3.3.1 ACL models comparison

The lack of a universally accepted formal semantics and message format for existing ACL models makes it difficult to make a comparison in terms of performance. Nevertheless, given table 3.4, we can assess languages described in this survey along the principles listed in section 2.1.

Heterogeneity: Unlike the "social agency" approach, several declarative languages like KQML, ARCOL, FIPA ACL do not explicitly embed any feature related to global perspective of an agent environment. This shortcoming, elegantly pointed out by Singh in [61], may come from the Belief-Desire-Intention perspective of these languages. In particular, the many dialects of KQML in real world applications demonstrate its low heterogeneity. In addition, ARCOL and FIPA ACL . On the other hand, the ability of all OAA agent systems, regardless of their implementation language and platform to communicate through ICL demonstrates gives an excellent exemple of heterogeneity.

While *separation* and *transparency* are common underlying principles of

existing ACLs, the *interoperability* issue is addressed from different point of views. For example, the KSE group supports interoperability at the levels of communication language, ontology, and knowledge base with respectively *KQML*, *Ontolingua* and, the *Knowledge Interchange Format* (KIF). Unfortunately, with real world practical applications of KQML, several dialects arose and different agents are unable to interoperate. On the other hand, FIPA and ARCOL propose predefined interaction protocols (conversation policies) in addition to a formal semantics of individual messages to support interoperability. In fact, these approaches need to include a global semantics for agents conversations (compositional semantics) and fine-grained constraints on their ACLs.

Although *extensibility* and *scalability* are originally parts of the design strategies of KQML, ARCOL, FIPA ACL, and OAA ICL, their underlying semantics and conversation policies do not explicitly show how extensions should be integrated in harmony with existing communicative acts. For example, in a domain-specific implementation of an ACL, a user may create new performatives (communicative acts) which do not necessarily fit in any predefined conversation policy. As a result, an incoherent interpretation of messages may affect the common goal of participating agent in a conversation.

The actual *performance* of existing ACLs is generally evaluated at the transport protocol level with their corresponding API. For example, a number of KQML implementations (Lockheed KQML, Loral/UMBC KQML) has efficient communications layers and various enhancements significantly reduced communication costs. How the choice of an ACL contributes to the performance of a multi-agent system may not be easy to evaluate. However, some experiences suggest that the lack of reliability and safety properties in most ACLs could be an impediment to performance in multi-agent systems.

The descriptions in the previous sections show that there are three perspectives in ACL design: the declarative approach, the procedural approach, and the approach based on the concept of "social agency".

On one hand, the declarative languages FIPA ACL and KQML have similar syntax, identical message format and share several parameters. In KQML, primitive CAs must bear the burden of describing the actual mental attitudes (intention, belief, commitment, and choice) of the message sender whereas FIPA ACL embeds some of these properties in the expression of the message content. In addition, the structure of a message in KQML must exhibit explicitly features pertaining to agent management, communication management (transport and facilitation), and multiple response queries. In contrast, FIPA offers equivalent primitives embedded in a specific *request* primitive and has no facilitation features outside of a directory facilitator (DF). Some critics like Cohen and Levesque argued in [6] that KQML semantics is ill defined and its

Table 3.4: Comparison of ACL models.

ACL models \ ACL elements	Message Format	Content Syntax / Language	Content Semantics	Support for Interaction Protocol
KQML	Layered structure Seperation between content, message, and communication	Knowledge Interchange Format No predefined content language	Informal semantics	No predifined negotiation protocol
ARCOL	Structure based on a first order modal language with Feasibility Precondition (FP) and Rational Effect (RE)	SL language	Formal semantics	Minimal Cooperativeness Protocol Corrective-Answer Protocol Suggestive Answer Protocol
FIPA	Layered structure Seperation between content, message, and communication	SL language	Formal semantics based on modal logic	FIPA-Contract Net FIPA-query, FIPA-request FIPA-Auction-Dutch/Dutch
ICL (OAA)	Procedural approach ICL expressions	PROLOG-like	No semantics	No predefined protocol
AOP	mental attitudes and commitment rules Modal logic as a basis	No defined language	No semantics	No predefined protocol
Social Agency	Social structure Social agency or Social community	No predifined language	Formal semantics with public perspective and common beliefs	No predefined protocol

"performatives" (communication primitives) are ambiguous and do not cover completely all interactions in agents conversation. Over time, KQML designers improved its semantics and provided a better foundation for the many applications using this language. Although, the FIPA standard ACL is arousing much interest, indeed, as Labrou and Finin stated in [20]: "*KQML has played a 'pioneering' role in defining what an ACL is and what issues are when it comes to integrating communication into agent systems.*"
On the other hand, in mobile agent systems, communication is intimately tied to the underlying transport protocol and is mainly in the form of simple requests for information. No assumption is made about the mental state of a mobile agent nor is any form of knowledge exchanged.
Then, models based on the social dimension of agent interactions like Singh's "social agency" in [61] and Colombetti's "space of common beliefs" in [8] seem complementary. The second approach could be used as a practical application of the first and include agents mental attitudes.

3.3.2 Issues and perspectives

Research in ACL is concerned with issues of design approach (procedural versus declarative), semantics, interaction or negotiation protocols (conversation policies), ontologies, and supporting communication infrastructures. Although semantics has attracted much attention, the question of designing a satisfac-

tory ontology shared by all parties involved in a conversation still remains an important issue. In addition, the idea of structuring a conversation around established conversation patterns known as interaction protocols or conversation policies (e.g. Contract-net), and the choice of such protocols, adds a new dimension to the ACL implementation debate. Agent communication languages will be of long term value to industry and academia only if these pragmatic issues are resolved.

• *Design approach*: ACL designers adopt either the procedural approach or the declarative approach. In the first approach (e.g. OAA ICL), ACL expressions (messages) are modeled as procedural statements that inform the recipient agent about the requirements of the sender and how these requirements should be carried out. In the second approach (e.g. KQML, FIPA ACL), messages are modeled as declarative statements that specify a primitive communicative act, an ontology, a content language, an interaction protocol, and a message content. Generally, the choice of an approach depends on the architecture of the current multi-agent system and the nature of the information exchanged. Although the declarative approach is used in the KQML and FIPA ACL de facto standards, ACLs implemented with procedural statements like the OAA ICL from SRI International make excellent applications.

• *Consensus on Semantics* appears to be the most important and challenging problem faced by the ACL research community. Although several attempts to define a good semantics has been done like in [18] and [36], there are still some important aspects (Singh in [61] and Moulin in [43]) that must be taken into account. A common ACL semantics is needed to guarantee that concepts used in a message are correctly interpreted at the receiving end with no regard to context. To this end, a semantics based solely on mental state of agents may not be appropriate for all domains. Ultimately, the standard semantics for ACL will include a complete coverage of its domain, have the means to handle a flexible context, a conventional type, and a public perspective. In this line of thought, Moulin in [43] argues that the notion of role, decision power, social network and communicative conventions have an impact on agent communications.

• *Interaction protocol, naming, registration, and facilitation services* are important elements of an ACL design. Compliance with interaction protocols and conformance testing are essential for ACLs conceived according to some standard because sending a message doesn't guarantee its correct interpretation at the destination. These rules will ensure that different implementations of an ACL will let agents communicate efficiently. To date, these issues have received little attention.

Due to their practical importance and interest of industry in this technology, different multi-agent systems may need to interact. Independently developed

systems may be reused as components or constituent agents in a new integrated multi-agent system. For this reason, interoperability between different systems becomes a necessity that can be achieved only if some aspects of agent communication is standardized. The Foundation for Intelligent Physical Agent proposed a comprehensive set of MAS management standards including an ACL standard that could be a solution to the interoperability issue. However, a single, unified ACL may not suit all application domains. For example, the procedural approach in the design of the ICL language in the OAA system [42] prevents any kind of interoperation with agents using a declarative ACL in their native system. Instead of a one size fits all approach to the design of ACLs, practical applications of ACLs in industry will ultimately suggest that only some aspects of an ACL like semantics and ontology need to be standardized leaving room and the freedom to a given system to conceive its message format, select its content language, and communicative acts.

3.4 Summary

In this chapter, we suggested a general ACL framework motivated by a number of design principles and specifications. In addition, we described elements of the structure of an agent communication language. Then we presented some emerging ACL models together with similarities and differences between these models. Multi-agent systems and related ACLs are promising areas of research and new perspectives like social interaction of agents are paving the way for a better foundation of agent communication.
Last, we presented some of the current issues in ACL design. Although semantics seems to attract a lot of attention, we showed that issues related to ontology design and agent interaction protocols deserve as much attention if agent communication languages must be of value to industry and academia.

Chapter 4

Quality of Service (QoS)

A number of applications like Video-on-Demand (VoD), Desktop Video Conferencing, Distributed Collaborative Environments, Distant Learning are taking advantage of the new capabilities offered by current networks and end-systems. However, these applications will not be able to provide satisfactory level of service if a suitable QoS provision scheme is not designed to sustain their activities.

Originally, the principles of Quality of Service (QoS) were applied to computer network communications. As QoS is gaining wide acceptance, its mechanisms are being applied to end-systems in distributed multimedia systems (DMS) as well. The concept of QoS is best described through the levels of service, a given provider can offer in order to sustain an activity.

4.1 QoS concepts and management

In a distributed multimedia environment, any activity that contributes to the production, delivery and consumption of some multimedia data can be characterized by the quality of its service. The followings are some well accepted phases of Quality of Service provision borrowed from the network communications area that are used in distributed multimedia environments:

QoS Specification: A service user must specify to the provider with objective or subjective QoS parameters the level of service that it expects.

QoS Negotiation and Mapping: In this process, the system must identify the set of components that are able to provide and sustain completely the required level of service. Then, with adequate mapping, the required QoS are translated from one system layer to another.

Resource Allocation and Admission Control: In conjunction with the negotiation phase, necessary resources are reserved for an activity. The required resources availability could be an issue unless the admission control

functions compares beforehand the activity needs with the existing resources. Admission is then granted when enough resources exist.
Monitoring: Once an activity has started, a given system layer constantly monitors the lower layer to check for alteration in the value of the performance parameters.
Maintenance and Adaptation: These processes aim to counteract the percieved degradation or violation of the contracted QoS. Adaptation is achieved through change in the topology of the components selected by the negotiation process, a coarsed or fine grained resource adjustment or an explicit change of service by the user.
Re-negotiation: When a degradation of the QoS level has reached a critical level and nothing further can be done to recover or adapt to a lower level of service, a re-negotiation is initiated by the system. This scheme proposes an entirely new QoS provision process. A change of service preferences by a user also can triger a re-negotiation. In addition, the *QOS contract* binds the entities (network provider, network and end-system resources ...) involved in the QoS provision. We define it in generic terms as follows:

```
typedef struct{
  service-spec_t       : service-spec;
  committment_t       : committment;
  monitoring_t       :  monitoring;
  adaptation_t      :  adaptability;
  dependability_t  : dependability;
  service-contract_t }
```

The contract is composed of the *service specification*, the class of *commitment* the resources are able to provide to sustain a given activity, the *monitoring* process, the *adaptation* process, and a subjective parameter we call *dependability*.
A user states with objective and subjective parameters in the service specification the performance expected. For instance, in dealing with timeliness parameters like loss, delay, jitter and throughput could be considered.

The commitment clauses are known as **Deterministic**, **Statistical** and **Best effort**. The first guarantees that resources are exclusively dedicated to an activity. The second is a loose version of the first where a service tolarates some fluctuations because resources are shared and eventually pre-empted. The best effort commitment does not guarantee any level of service nor does it permit control over any resource. These clauses are qualitative expressions that helps precise the nature of the service required.

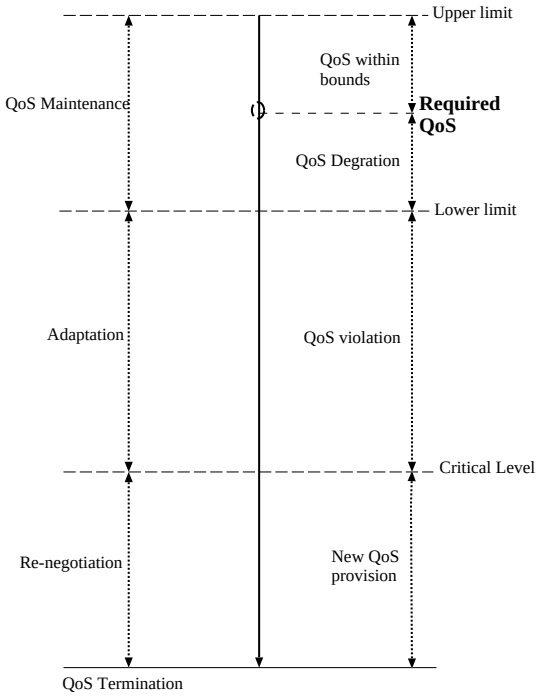


Figure 4.1: A simple view of QoS Management

The scope of the usual service contract could be extended with the **dependabiliy** parameter. Dependability, with the values High, Average and Low, is a subjective QoS requirement that states the level of trust and confidence a user is putting in the system. The dependability parameter puts an emphasis on the user, whereas the commitment clauses (deterministic, statistical, and best effort) relate to the way the system manages its resources. For example, a user working on a low-end system might concede a low dependance. However, when a multimedia file is being transfered from a remote server, the deterministic clause is required for loss.

4.2 Example of QoS architectures

Several research projects have proposed QoS architectures that have emerged in the literature. We propose here a brief overview of these architectures as described in [1]

- *The Tenet QoS model* [15]: The *Tenet Architecture* by the Tenet Group at the University of California at Berkeley, is a family of protocols including a Real

Time Chanel Administration Protocol (RCAP) for generic connection establishment, resource reservation and signaling. The other element, Continuous Media Transport Protocol (CMTP) deals with the transport and network layer for resource reservation and flow setup.
- *The QoS-A model* [5]: The Quality of Service Architecture (QoS-A) is an architecture composed of several layers and planes with specific functions. The distributed systems platform, the orchestration layer, the transport layer, the network layer, the data link layer, and the physical layer. In addition, QoS management is achieved in three vertical planes:
The protocol plane for media control,
The QoS maintenance plane for monitoring and fine-grained maintenance purposes,
The flow management plane for flow establishment, QoS mapping, and QoS scaling (filtering and QoS coarse-grained adaptation).
- *The OSI QoS model* [28]: The OSI QoS Framework defines concepts and is dedicated to QoS support for OSI communications. It helps identify objects for QoS management in open systems standards. This framework includes:
  * *QoS requirements* with the management and maintenance functions,
  * *QoS characteristics* which describe the elements and their measures,
  * *QoS categories* that group related QoS dimensions,
  * *QoS management functions* are applied to meet QoS requirements.
- *The Int-serv QoS model* [3]: Inter-serv architecture by the Internet Engineering Task Force (IETF) is dedicated to QoS control for multi*media applications over an integrated services inter*network. Although initially reserved to the network, Inter*Serv applies also to end*systems. It includes four components:
  * a packet scheduler
  * a classifier for mapping incoming packets to QoS classes,
  * an admission controller for flow admission,
  * a reservation setup protocol for flow specific state in the routers along the path of the flow.

4.3 Summary

In this chapter, we examined the concept of Quality of Service, its requirements and elements involved in its provision and management. QoS specifications, QoS negotiation and mapping, resource allocation and admission control, maintenance and adaptation, and negotiation are the necessary steps in providing QoS for any system in need of a sound service. A number of architectures like Tenet, QoS*A, and OMEGA, conceived in academia illustrate different design approaches.

Chapter 5

Mobile Agent Support for QoS in Distributed Systems

5.1 Introduction

In this chapter, we propose two different architectures for supporting QoS in distributed systems:
In the first architecture, a mobile agent is an object characterized by its attributes and methods, which performs operations. A meta-object like in the Apertos Operating System described in [75] is an abstraction for the set of methods that operate on appropriate resource managers when satisfying several QoS. The foundation of our architecture lies on three concepts: An *agent* is an object that negotiates QoS on behalf of a user. Here a *virtual host* is a meta-object that spans several resource managers and a *place* is a dynamic virtual location inside a virtual host where mobile agents use some services.
The second architecture presents a QoS adaptation scheme based on the mobile agent technology. Then we introduce the idea of *macro-adaptation* as the coarse-grained adjustments, and *micro-adaptation* for fine-grained corrective actions on resources.

5.2 An architecture for a QoS-based mobile agent system

In the design of the proposed architecture, four entities come into play: a virtual host, a virtual host interface, a place and a resource manager. A place, located inside a virtual host, is responsible for providing the adequate computation resources for the execution of an agent and grants it the desired QoS. A place confines the actions of the agent to a restricted environment

for portability and security reasons. Each place is configured specially by an interface according to the requests submitted by the mobile agent. This interface is the virtual host interface, which is also in charge of translating the subjective user defined QoS parameters into system compliant parameters. The virtual host interface separates the responsibilities of QoS negotiation and

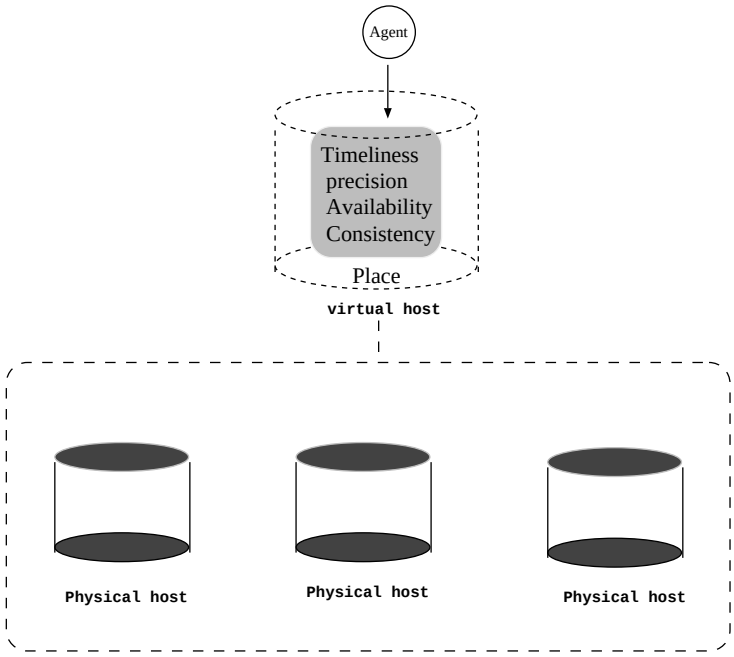


Figure 5.1: A virtual host spanning multiple machines

resources management.That is to say: the mobile agent system takes care of QoS negotiation at the user level while the virtual host deals with resource management at the machine level. In fact, we propose in this paper a protocol for QoS control that relies on the movements of mobile agents inside and between places. In order to satisfy its owner's QoS request, the mobile agent in our system implement an effective policy of QoS control inside places created on demand. The virtual host interface receives mobile agents and translates their data (subjective QoS) into system defined QoS . With this data, it configures an adequate place inside a virtual host. Then, the mobile agent enters that place and use the services provided there.

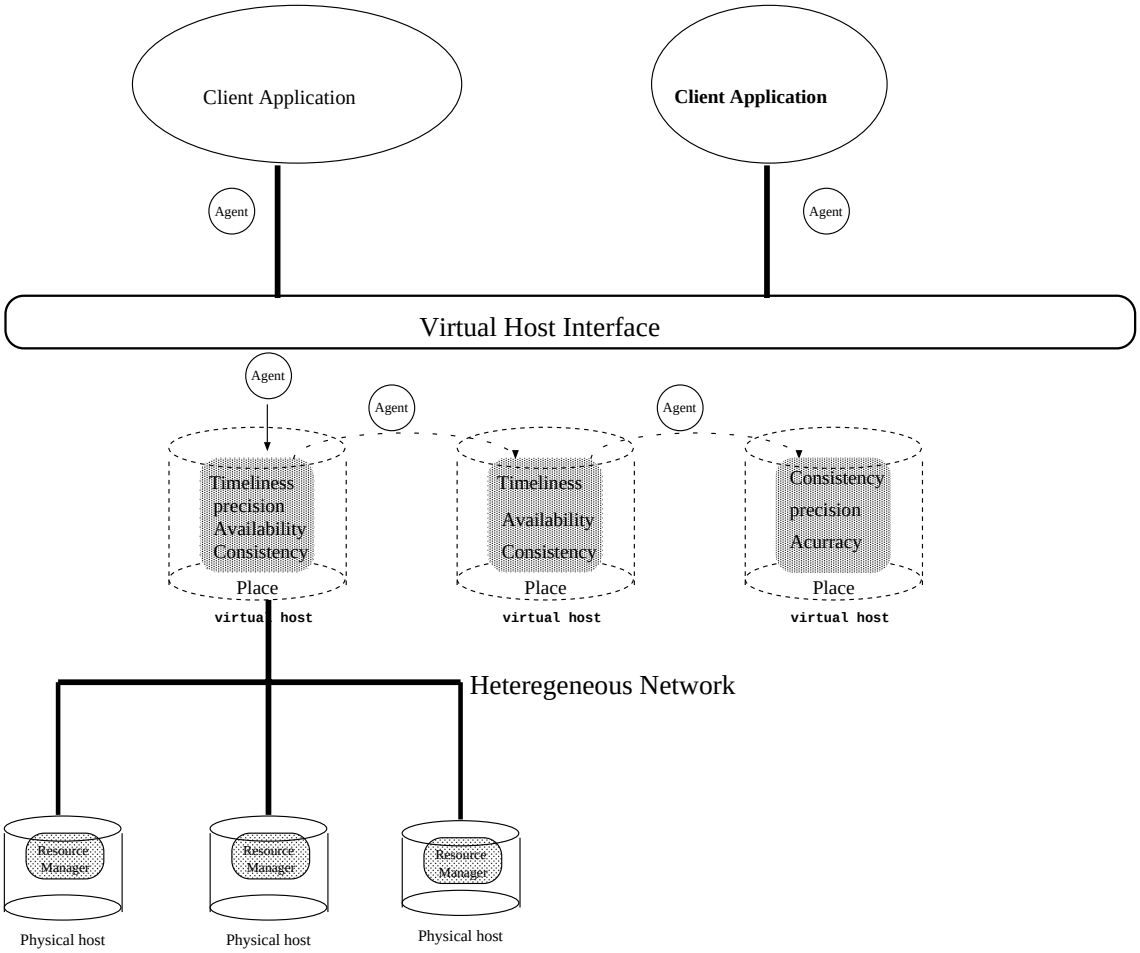


Figure 5.2: A client view of the mobile agent system

5.2.1 QoS assumption

In this model,we assume that a client application targets a fixed number of QoS: timeliness for systems with time constraints, accuracy, precision, availability and consistency. We assume also that a number of host machines scattered across the network have the necessary resources to satisfy the user needs. Each place inside a virtual host in conjunction with the adequate resource managers provides several types of QoS and not necessarily the entire set of QoS. In addition, in our context a resource reservation scheme exists at the level of each physical host machine.

5.2.2 Agents movement

A client application with multiple QoS requests expressed as timeliness, accuracy, precision, availability and consistency may need several resources in order to be satisfied. In this case, the mobile agents in charge migrate with these requests to an appropriate virtual host. Before any operation, the subjective QoS is translated by the virtual host interface into system defined QoS. With this data, the interface configures a suitable place within the virtual host. Each virtual host spans, over the network, a number of resource managers and host machines containing the needed resources. When a place is built, the mobile agent moves into it and tries to secure the available QoS and migrate - if necessary - to other places to satisfy the complete set of requested QoS. In

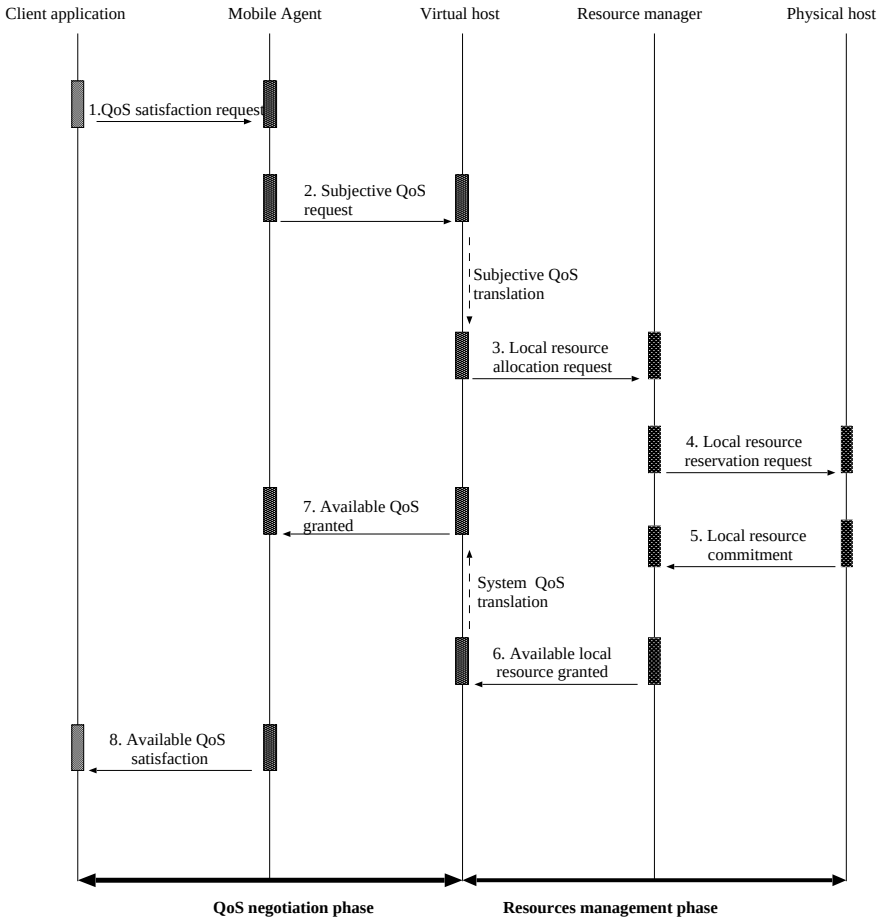


Figure 5.3: The QoS negotiation process

particular, when a time constraint is set by the client application, the mobile

agent must set a priority to the timeliness parameter and accept lesser values for other QoS. Assuming that the execution environment is reliable, the agent migrates throughout the entire network, from virtual host to virtual host in order satisfy all QoS requests.

5.2.3 Adaptation

On one hand, during the execution of an agent at a place, resources at physical hosts can be added, removed or simply fail unexpectedly. In such cases, the virtual host interface reconfigures a new place inside the related virtual host and reassigns the agent to this new place or the agent simply travels to another place. Here the only impact on the mobile agent movement is the change of path taking in account the new event. This simple and straigthforward strategy helps rearrange available resources and maintain user satisfaction. On the other hand, as mobile agents are bound to perform many operations and use the services provided by multiple hosts, some concerns naturally arise about the reliability of the agent itself, its execution environment, the host machine and the network it uses. To deal with this issue we propose a simple scheme for failure recovery. At the time it leaves a place, an agent creates a checkpoint on a persistent store. This way, when a host crashes together with an agent, the last host visited can completely recreate this agent with its state.

5.2.4 Inter-agent communication

As many agents migrate across the network on behalf of their owner to satisfy different QoS requests, they need to cooperate for optimization purposes. Agents communicate with one another by passing messages about new environmental conditions at their respective place. This way, an agent can have an impact on the computation of other agents. For example, when an agent, on behalf of the real time system user, plans to migrate to a place where timeliness (CPU time) used to be available; another agent staying at the targeted place knows that a failure of the local resource makes this QoS no longer satisfiable at the virtual host managing it. So the latter agent, upon request, sends a message to the first agent which then changes its path. Also, the level of QoS available locally may not be sufficient and the agent in need looks elsewhere for better service. The inter-agent communication is achieved through asynchronous messages rather than Remote Procedure calls (RPC) for flexibility purposes. In this scheme, an agent calls a communication primitive. The supplied arguments here are the identity of the receiving agent and the message.

5.3 Mobile agents and QoS adaptation in DMS

In a distributed multimedia environment, unstability, fluctuations in the level of service or temporary failures are inherent parts of the activity of any system. As a result, service maintenance or adaptation are required if user confidence is important. In our context some particular motivations are the following:
1.In distributed systems environments, shared resources due to system policies (resources management, priotities, ...) may not be available on request.
2.Congestions in the network or end-sytems often occur.
3.Different clients QoS requirements, depending on preferences, may create conflicts.
4. Granted resources may be pre-empted and not be guaranteed over time.

In our approach to adaptation, we define the concept of **macro-adaptation** as all the coarse-grained adjustments relying on agents movement with change in components topology; whereas **micro-adaptation** describes all the fine-grained corrective actions done at the lower level resources along agents hierarchy. In addition, in this agent-centric design, we assume that QoS monitoring and resource reservation are done at the component level. Our idea is to organize an adaptation strategy inside mobile agents themselves: computation is adaptated to the available services. The interactions between mobile agents and component interfaces incorporate the adaptation mechanisms.

5.3.1 Agent-based adaptation strategy

Macro-adaptation and micro-adaptation are done in an integrated and hierarchichal fashion.

An integrated approach:
The system we propose is integrated in two aspects:
First, all components spanned by related mobile agents, are integrated in a single end-to-end QoS provision process.
Traditionally, the negotiation process identify a finite set of components that are able to sustain an activity and assign to each, in a rigid configuration, its share in the overall end-to-end QoS provision. In contrast, with the introduction of mobile agents, several components assembled in a loose configuration, are connected dynamically to provide the required QoS. Such an approach is described in [34]. In our scheme, system agents coordinate the operations of system components and communicate with network agents responsible of network transactions at the transport layer of the network.

Second, all adaptation mechanisms in research related to QoS provision have been implemented as isolated processes activated when a degradation or violation actually occurs. However, unstability, degradation are unavoidable events

in heterogeneous environments. We beleive that the best way to deal with these events is to predict them long before they happen. The QoS negotiation process, the resource allocation and admission control should incorporate strategies that prevent as much as possible violations of QoS requirements.

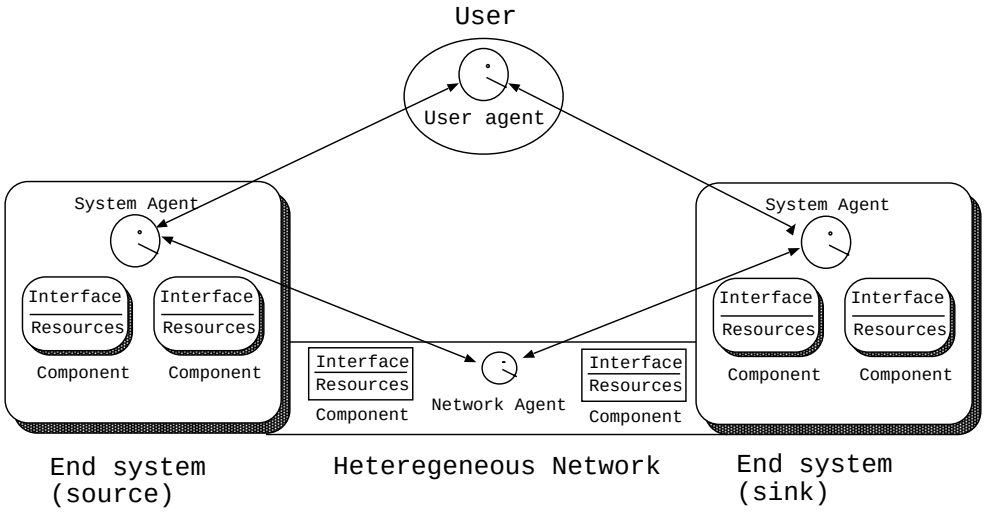


Figure 5.4: Agents structure in end-to-end QoS provision

A hierarchical approach:
When a degradation or violation is detected, maintenance or adaptation is carried out along the following steps:
1. The system identifies the component that fails to satisfy the local QoS.
2. The related agent (ex. end-system agent) attempts to solve the problem locally by:
2.1. moving the computation to a peer component (ex. end-system component).
2.2. sending a message to a sub-component agent to make the necessary corrections down into the hierarchy or interact with the resource manager.
2.3. sending a message to another agent located elsewhere and the previous steps are performed there.
3. If local adaptation fails, the adaptation process is moved to another part of the system (ex. network). The same strategy as in (2.) is re-enacted.
4. When resources are scarce or the service requires a strict amount of resources (deterministic clause), mobile agents may transfer immediatly the computation into another part of the system.
Horizontally, computation is moved with the QoS access point from the current component to a peer component.

Vertically, a message is sent to an agent at a sub-component down in the hierarchy. The target agent can further move the computation to a peer sub-component if necessary. Should the QoS maintenance or adaptation to a lower QoS fail, the end-system agent sends a message to the network agent to carry out a similar strategy.

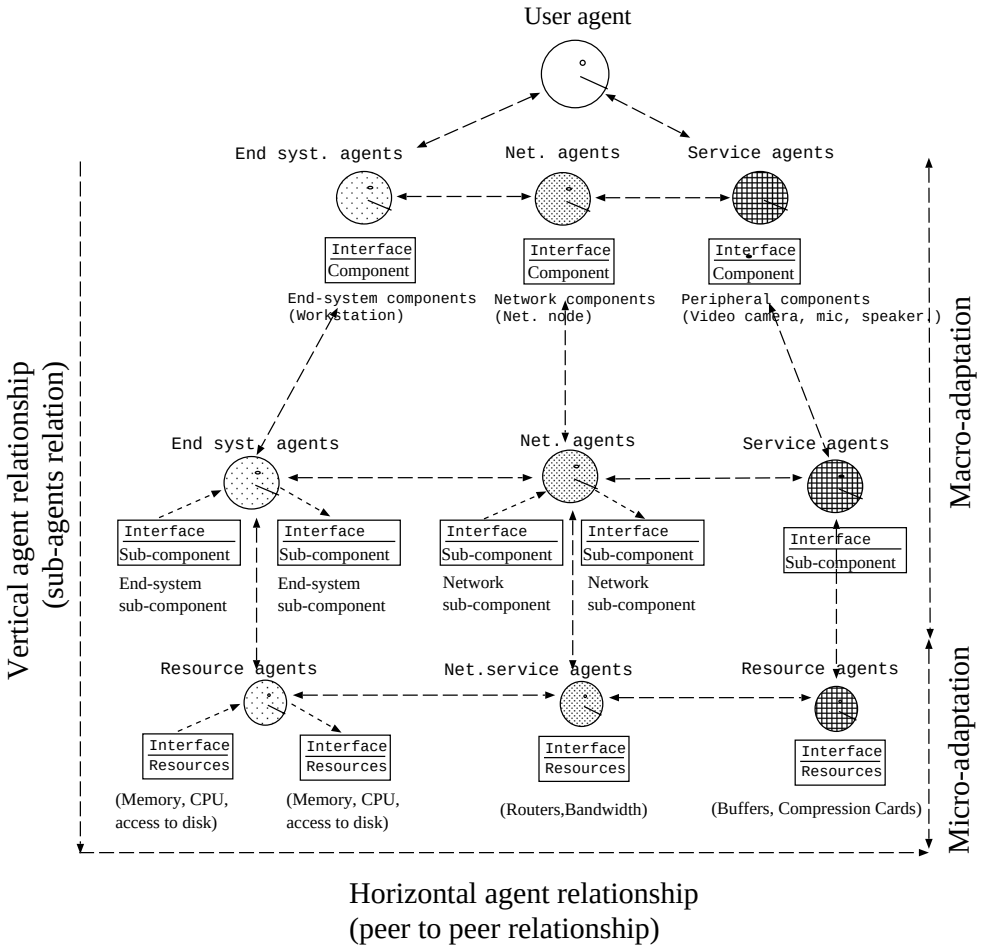


Figure 5.5: Agent-based hierarchical adaptation

5.3.2 Agent-based adaptation mechanisms

Agents interacting with components

The main purpose of our system is to make existing services at a component available on request to agents. In normal situation, the level of the service

offered matches the request. Otherwise, a maintenance or adaptation to a lower level is proposed. Upon receiving from the user agent the QoS contract with the specified QoS parameters, an end-system agent moves to the *agent execution environment* of a component as depicted in figure 5. In this environment, the agent makes the necessary computations, calls the *Application Programmer Interface (API)* in order to use the services.

An example of interface is the Java Database Connectivity Layer and the Java Advanced Windowing Toolkit. Here we assume that during the previous Resource Allocation and Admission Control, resources necessary to sustain the computation have been reserved. When there is a degradation and the level of service is not satisfactory and all attempts to maintain it has failed, the agent has the option to adapt its computation to the available resources. That is, the mobile agent alters a QoS level to a lower level.

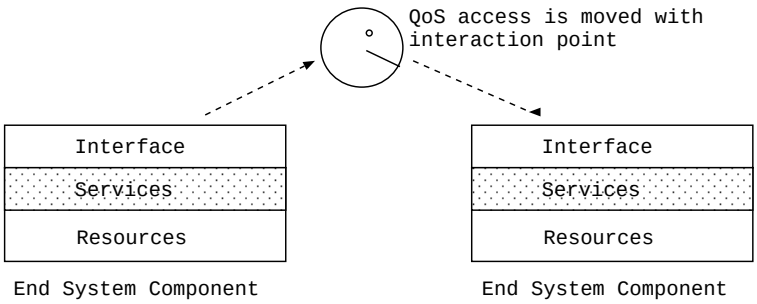


Figure 5.6: Agent-based Adaptation Model

Agents migration:
Agents mobility is supported by the object serialization functions included in the Java language. For the three types of agents (user agent, end-system agent and network agent) migration happens inside their respective domain (end system or network). An agent migrates between peer components of the same domain. A network agent assigned to nodes can migrate only between nodes and so do end-system agents. Often, the operation of a component depends on other sub-components. These sub-components are visited by related agents down in the hierarchy. All computations are identical at this level as at the upper level. At the bottom of the hierarchy, in connection with the resource manager, resource agents participate in fine grained adjustments called micro-adaptation by travelling between locations.

Inter-agents interactions:
Along the hierarchy, agents send messages in a serialized form horizontally to peer agents located in another domain or vertically to agents related to sub-

components or resources. The two types of communication are at the heart of agents cooperation for maintenance or adaptation.

QoS Adaptation Strategy	Context	Agent Migration	Agent interaction
Macro-adaptation	End-system	WS ----> WS NSAP ----> NSAP	Component API Message
	Network	Node ----> Node Router ----> Router	Component API Message
	Peripheral (Video camera, Mic.)	Video Cards --->Video Cards	Component API Message
Micro-adaptation	End-system	CPU/Memory ----> CPU/ Memory	Resource Manager Message
	Network	Buffer ----> Buffer	Resource Manager Message
	Peripheral	Codec ----> Codec	Resource Manager Message

Figure 5.7: Agent-based adaptation scenario

5.3.3 An application area

Let's consider a system that captures images and stores them in a remote multimedia database storage (MMDBS). In addition, this data in the remote server is accessible to other users ready to be displayed locally. That is, the whole system can be used in two modes. When a user captures data and stores it remotely. The source is a camera together with a set of workstations linked in a subnetwork and the sink is the multimedia database server. Conversely, a user in need of some multimedia data could fetch it from the database server and display it locally.

In the first mode, a QoS manager in its negotiation phase would simply select a configuration made of the following components: the camera, a workstation with a video capture card, a path of nodes and routers in the network and the database storage.
When all the resources allocation and admission tests complete, the service starts. We assume all the workstations in the subnetwork have the same capabilities and resources. The camera physically linked to a given workstation

capture images which are compressed, sent accross the network and stored in the database.

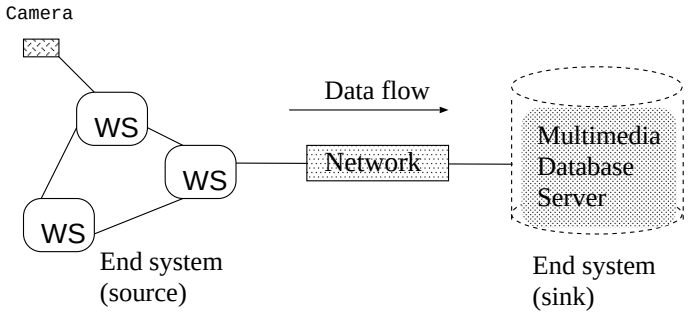


Figure 5.8: Example system: storing data in a multimedia database

In the second mode where data is fetched from the remote server, the QoS manager selects a configuration made of the following components: the database server (the source), a path of nodes and routers in the network and a workstation (display sink). In fact, during the QoS negotiation phase, several potential configurations of components and resources were identified and recorded by the QoS manager.

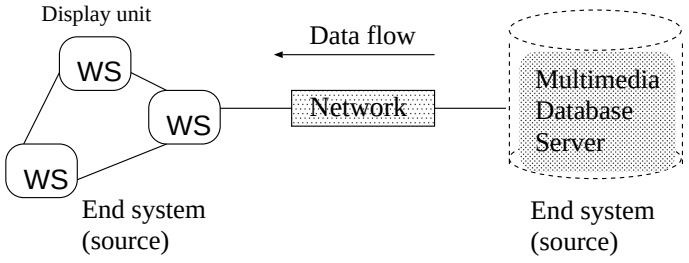


Figure 5.9: Example system: fetching data from a multimedia database

Here the user agent, system agents and network agents are located respectively in the user application, the end-system and the network. When a degration or violation occurs, the QoS manager tries to locate and identifies the failing component (source workstation, network node, router or database server). The mobile agent located at the failing component has two alternatives. On one hand, it can send a message to other agents located at subcomponents like the operating system, the hard disk, the memory or video capture card. On the other hand, the mobile agent can move the necessary data

to another workstation (macro-adaptation) and restart computation where it stopped. As a result, the intial QoS level is maintained or adapted to a lower level. In case violation is detection at network node or router, the corresponding agent migrates with all the data to another node or router.
Although the service started with a given configuration of system components, the mobile agents give us the possibility to move computation from one component to another. The configuration changes dynamically to cope with violations.

5.4 Related Work

A number of works have addressed the problem of QoS adaptation with various techniques but few use the agent technology to improve QOS provision. For example, the system described in [47] use the interaction of classes of mobile and fixed agents, the concepts of contract and QoS agency. Unfortunately, the ideas of several classes of agents, which communicate, create necessarily congestion on the network. In addition, in [47] neither the issue of reliability nor changes in the environment are addressed. Although our basic architecture is similar in principle to the Telescript model in [24], the concept of virtual hosts in our system separating the responsibilities of QoS negotiation and resource management is new.
On the other hand, In [25], a QoS manager conducts QoS adaptation by interacting with static QoS agents bound to components. The QoS adaptation strategy relies on three schemes are. The first called Component Reconfiguration Scheme (CRS) performs adaptation by altering the components topology and initiating a smooth and transparent transition. The second named Resource Reconfiguration Scheme (RRS), redistributes the level of QoS supported by each component in a given configuration to cope with violation and keep the overall end-to-end QoS.
In [72], the concept of binding as abstraction of communications between components is used to sustain the required QoS through monitoring, adaptation and reconfiguration. Here binding objects, organized in a hierarchy, apply adaptation policies dynamically.

In [69], a hierarchical QoS adaptation based on the organization of binding objects is proposed. In this paper, adaptation is carried out along a hierarchy of bindings. From the top with coarse-grained actions to the bottom with fine-grained atomic resource control, the adaptation mechanisms are embedded in the binding components which are conceived as communications abstractions. In addition, these bindings are responsible for the end-to-end QoS maintenance.

In [12], an agent-based approach is introduced to carry out corrective QoS adaptation in a cloud of non-RSVP-capable network routers. Static agents monitor tunnel properties and interact with RSVP routers to deliver the required QoS.

In the above systems, when agents are used, their functions are limited to monitoring the level of QoS and eventually notifying the system of QoS violations at the host component. Our approach that consists of moving the QoS access point from one component to a peer component with mobile agents is new. The initial configuration need not be altered unless no component is able to compensate the degradation. Adaptation is performed smoothly by mobile agents horizontally or vertically in an integrated fashion. System agents - at the source or sink - and network agents cooperatively manage the end-to-end QoS transparently.

5.5 Summary

This chapter presented two architectures for agent support of QoS provision: The first approach offers significant advantages over classical methods in the QoS negotiation process. We borrowed the concept of Logical Disk from the operating system field and applied it together with the mobile agent technology in the QoS negotiation and management area. The mobile agent technology is well suited to applications which execute on a network with high latency or applications which operate on a partially connected basis.

The second approach is a description of QoS adaptation that relies on the flexibility of mobile agents. In general QoS provision, a static configuration of system components is provided. To this static configuration, the mobile agent technology allowed us to substitute a dynamic one. The adaptation strategy evolves around the interactions of agents and components interfaces on one hand, and the cooperation between agents themselves on the other hand. To achieve this, we have introduced macro-adaptation and micro-adaptation to describe the operations of agents that cooperate along a hierarchy. This second model - due to its integrated and hierarchical aspect - could improve the overall QoS provision significantly provided the entire system is reliable.

Chapter 6

Multi-Agent System Support for QoS Negotiation

6.1 Introduction

The tremendous growth of the Internet in the past few years sparked a whole new range of applications and services based on its technologies. Users will be able to take full advantage of these new capabilities only if there is an appropriate configuration to deal with the scalability and heterogeneity problems inherent to the Internet. In this line, resource discovery on the network and Quality of Service (QoS) assurance are important subjects that are drawing attention. In particular, the Service Location Protocol (SLP) [50] designed by the Internet Engineering Task Force (IETF) aims to enable network-based applications to automatically discover the location of services they need. However, SLP was designed for use in networks where the Dynamic Host Configuration Protocol (DHCP) [11] is available or multicast is supported at the network layer. Neither DHCP nor multicasting extends to the entire Internet because these protocols must be administered and configured. As a result, SLP does not scale to the Internet.

Our objective in this chapter is to present a new QoS negotiation scheme and deal with two important limitations in resource management for large-scale applications: scalability and communications costs. We propose here a framework that relies on the concept of multi-agent systems and the Knowledge Query and Manipulation Language (KQML) [17]. In this framework, a user agent, a QoS manager agent, one or several facilitator agents, and service agents (application agent, system agent, network agent, and resource agent) engage in a mediated communication through the exchange of structured KQML messages.

6.2 System Framework

6.2.1 The problem

In standard QoS provision schemes for application running on small or local area networks, a QoS manager determines all configurations that can sustain an activity by:

- identifying necessary system component and building potential configurations.
- classifying these configurations
- selecting the most suitable configuration.

This approach assumes that the QoS manager has knowledge of potential service providers, system components and resources that exist in its environment and can communicate directly with them. As long as the number of entities involved in this service is small, this scheme is feasible and communication costs are acceptable. However, in a heterogeneous setting like the Internet with millions of computers, this approach shows two clear limitations:

- *First*: During negotiation, the QoS manager alone must bear all the bur-

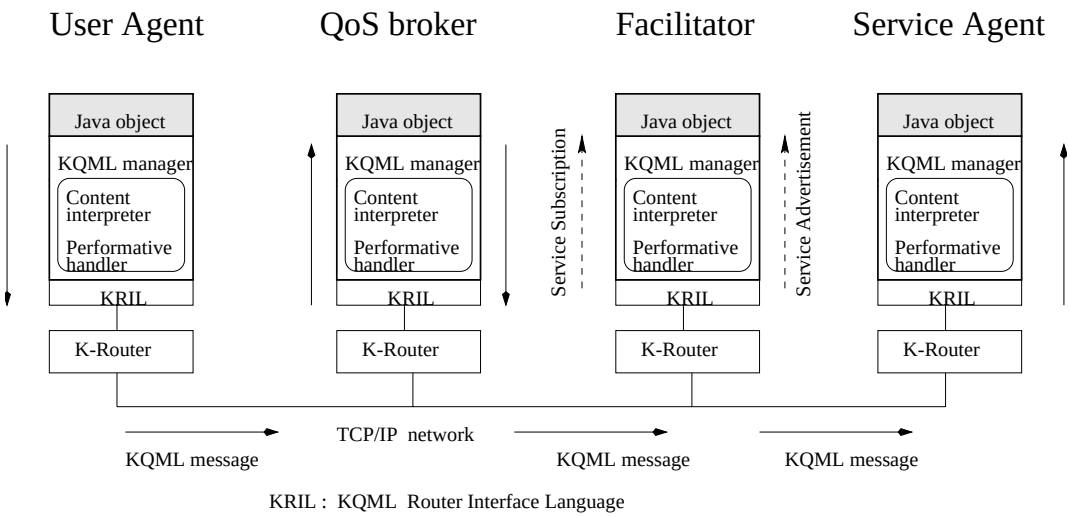


Figure 6.1: System architecture

den of identifying and selecting appropriate resources on a large scale network like the Internet. This situation adds a substantial overload on the operation of the QoS manager. In addition, services and resource may not be guaranteed consistently.

- *Second*: When the number of entities involved in a direct communica-

tion with the QoS manager is small, communication costs remain reasonable. However, in the Internet, these costs become prohibitive even with auxiliary local QoS managers.
To address these scalability and communication cost issues we propose a framework for QoS negotiation illustrated in figure 6.1 where clients applications and service providers engage in a mediated communication. The mediators called **facilitators** and **QoS brokers** are supplied with information about identities and capabilities of service providers by the providers themselves. These entities are modeled as software agents with attributes, capabilities and mental attitudes as in AI. At the core of our framework lies the concept of multi-agent system composed of a user agent, a QoS manager agent, facilitator agent, and service agents (network agents) communicating in KQML.

6.2.2 The concepts

Prior to starting a service, a user specifies and supplies the QoS manager with a level of service expressed in QoS parameters. Then, the QoS manager must identify the set of components that can sustain this service. This process uses the following concepts:

• An ontology provides a vocabulary for representing and conveying knowledge about a topic (e.g. QoS) and a set of relationships that hold among the terms in that vocabulary.

6.2.3 Example Conference Ontology

Types used in this ontology: STRING, NUMBER, and DATE.

Taxonomy:

```
Persons

    Organizers
        General chair
        Program chair
        Registration chair
        Tutorial chair
        Program committee
```

```
Elements
    Themes
         Agent Architectures
         Agent Communication Languages (ACL)
         Mobile Agents
         Network Agents
         Agent-based Software Engineering
    Tutorials
         Background  statement
         Target audience
         Presenter's resume
         Outline and description
    Submission
         Deadline
         Material format
         Destination
         Notification
    Software demo.
         Technical content
         Requirements (hardware and software)
         Demo. storyboard
```

Relationships:

Relation	Domain	Range
submissionAuthor	Submission	Person
submissionDate	Submission	DATE
demoAuthor	Software Demo.	STRING
demoDate	Software Demo.	DATE
TutorialAuthor	Tutorial	STRING
notificationSender	Program Chair	Notification
member	Organizers	Person
relevance	Submission	Themes
relevance	Tutorial	Themes
relevance	Sofware demo.	Themes

Our architecture uses four ontologies:

∗ a *yellow page* ontology for service advertisement by service agents. Yellow pages allows agents to locate other agents, given their names, addresses, functions, and capabilities. It is a service oriented general-purpose search

mechanism. This facility competency relies on the willingness of agents to register and update their identities, services, and locations to the local facilitator;

* a *white page* ontology only for locating an agent given its name.
* a *general QoS* ontology for the current domain knowledge, and a
* *QoS broker* ontology for asking network options.

- A *KQML manager* encompasses :
  * *Conversations* that group messages with a common thread identified by the ":reply-with" and :in-reply-to"parameters;
  * *content interpreters* that handle incoming response messages according to the ACL, content language and ontology associated to these messages. Originally, a KQML message is opaque to its content. A message content is processed by a content interpreter that gives a meaning to message received;
  * *performative handlers* that process a ACL message performative, content language and ontology.
  * *content-based routing* is achieved by the local KQML Router (**K-Router**). Messages sent by an agent can also be routed according to the interests of other agents. When relevant information is advertised by an agent, it is forwarded by a facilitator to the requester via the K-Router. The K-Router in conjunction with the facilitator initiate a mediated communication between agents.

6.2.4 The negotiation protocol

In our framework, four types of agents communicate in KQML according to the following protocol:

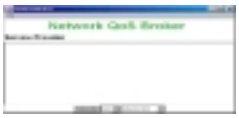
- The user informs its agent via an interface of a level of service.
- The user agent sends to the QoS manager agent a KQML message containing the required level of service. This interaction is described in figure 6.2.
- The QoS manager needs to identify all components necessary to build a configuration that can sustain an activity. Our simulation system considers only network parameters although a complete QoS configuration requires all application, system and network parameters. For this purpose, the network agent sends a KQML message to the facilitator agent and can ask its cooperation in four different ways (subscription, brokering, recruiting, and recommendation) in discovering all the appropriate resources. A structure of this KQML message and agent interaction is shown in figure 6.3.

QoS spec.

(**tell**
:sender **User**
:receiver **QoS-manager** (Network QoS Broker)
:language anACL
:reply-with id1
:ontology QoS
:content (*:throughput* 10 *:delay* 5 *:jitter* .2 *:loss* 5))

Corresponding KQML message

Figure 6.2: User and QoS broker interaction



Service request

(**broker-all**
:sender **QoS-manager**
:receiver **Facilitator**
:reply-with id0
:language **KQML**
:ontology kqml-ontology
:content
(**ask-all**
:sender **QoS-manager**
:reply-with id1
:language anACL
:ontology QoS-broker
:content (*:throughput* 10 *:delay* 5 *:jitter* .2 *:loss* 5)))

Corresponding KQML message

Figure 6.3: QoS manager and Facilitator interaction

- The facilitator agent acts as a resource broker that
  - recruits, recommends appropriate service agents (application agents, system agents, network agents) to the QoS manager;
  - forwards the QoS manager messages (brokering and recruiting) to suitable service agents; and
  - informs (on subscription) or recommend to the QoS manager service agents fulfill its requirements.
- All service agents (Network agents) advertise their capabilities to the the facilitator agent like in as in figure 6.4 and figure 6.5. Upon request from the QoS broker, the facilitator supplies the identities and locations of necessary network resources. At last, the user may view on an appropriate interface in figure 6.6 the available resources.

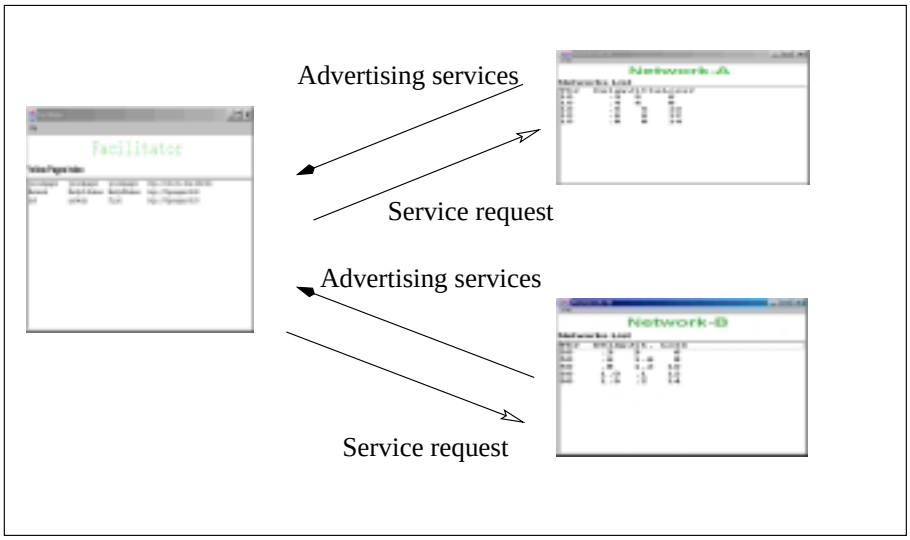


Figure 6.4: Service advertisement and request

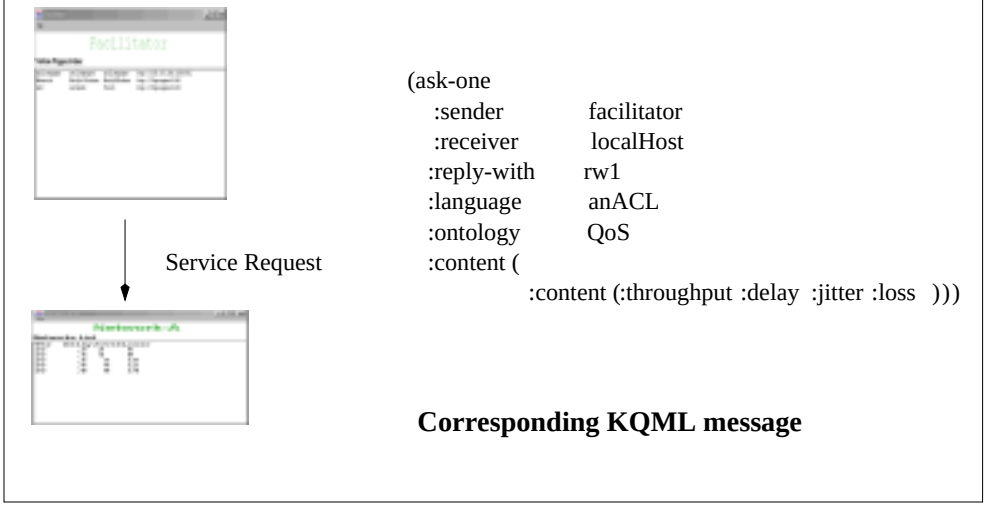


Figure 6.5: Network agents and Facilitator interaction

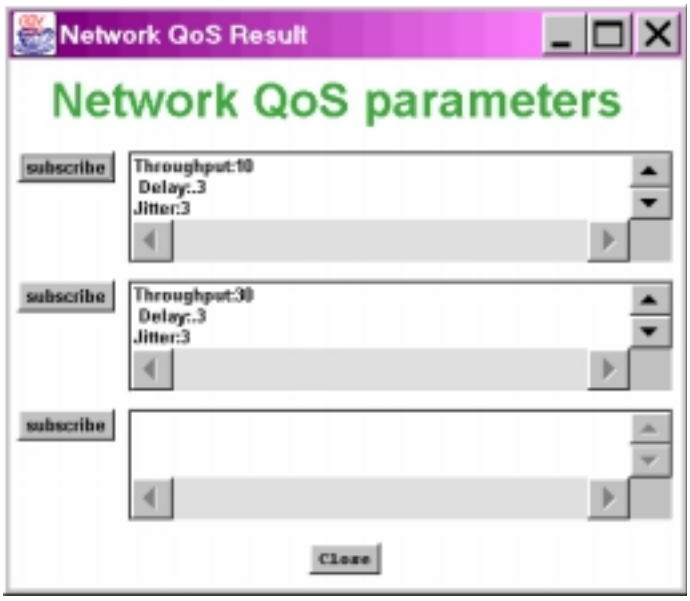


Figure 6.6: Request results

This QoS negotiation model for large scale Internet applications is applied in two ways: locally or remotely. When the required resources are available locally and registered at the local facilitator, negotiation is done at the current host as illustrated in figure 6.7.
On the other hand, when some resources are unavailable on site, the local facilitator reaches out to other facilitators at different locations as illustrated in figure 6.8. The local facilitator forwards requests (*broker-all*) to remote facilitators which in turn conduct a local inquiry. In fact, this approach to agents interaction is already used in the field of *agent-based software engineering* where application programs are modeled as software agents and interaction is supported by an appropriate ACL. In this approach, agents are organized in a *federated system* with messages relayed by facilitators between hosts.

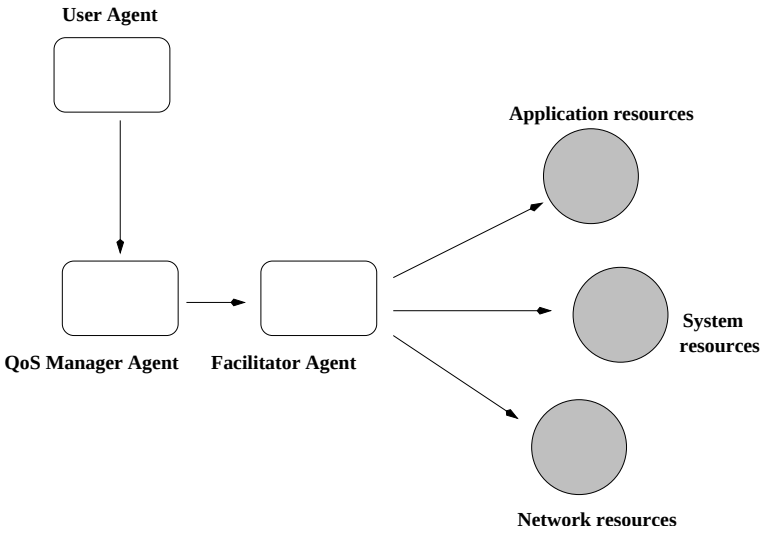


Figure 6.7: Local QoS negotiation

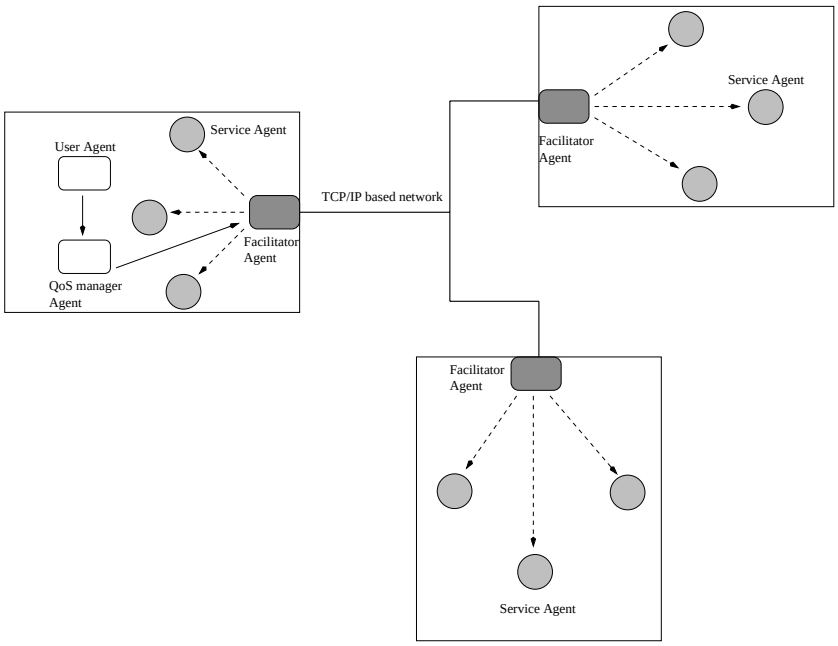


Figure 6.8: Large scale QoS negotiation

6.3 System Implementation

In experimenting with this model of resource discovery and QoS negotiation, we implemented a system prototype in the JAVA language that simulates QoS negotiation between several agents at the network level. That is to say, to illustrate our approach, a user agent and network agents communicate via a QoS broker and a facilitator in terms of network parameters only. First, local negotiation is considered, then it is extended to remote locations on the Internet.
The implementation of our prototype includes the following tools:
- The Java-based KQML API called **JKQML** in
[26]. The JKQML API with its structure in figure 6.9 adapted from [26] provides a platform for designing KQML-enabled agents. JKQML is written completely in the JAVA language and provides interoperability to software that need to exchange information and services. The whole interaction pro-

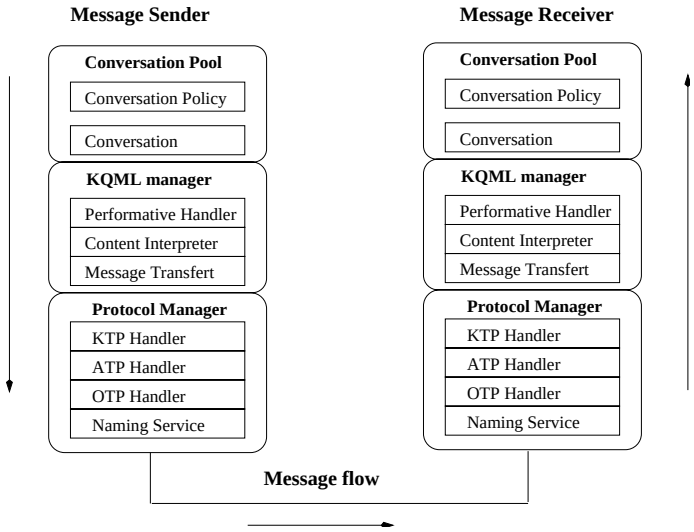


Figure 6.9: Structure of JKQML

cess between the players in our multi-agent system can be described in three steps:

6.3.1 Step 1: Registration phase

All agents register to the facilitator with the "lifecycle" ontology by sending a KQML message object in the following piece of code:

```
KQML kqml = new KQML();  (a new KQML message object is created)
String rw = new String(kManager.getInitialID()); (reply identification)

kqml.setPerformative(``register'');
kqml.setParameter(``:name'', agentName);
kqml.setParameter(``:sender'', localHost);
kqml.setParameter(``:receiver'', facilitator);
kqml.setParameter(``:reply-with'', rw);
kqml.setParameter(``:language'', ``anACL'');
kqml.setParameter(``:ontology'', ``lifecycle'');
kqml.setContent(``(:class Facilitator)'');
```

6.3.2 Step 2: Advertisement phase

In the advertisement phase depicted in figure 6.11, service agents advertise their capabilities at the facilitator with the "yellow pages" ontology by sending a KQML message object in the following Java method:

```
KQML kqml = new KQML();
String rw = new String(kManager.getInitialID());

kqml.setPerformative(``insert'');
kqml.setParameter(``:name'', agentName);
kqml.setParameter(``:sender'', localHost);
kqml.setParameter(``:receiver'', receiverAddr);
kqml.setParameter(``:reply-with'', rw);
kqml.setParameter(``:language'', ``anACL'');
kqml.setParameter(``:ontology'', ``yellowpages'');
String content = new String(
                     ``(:ProviderName "Network A" ''
                     `` :category QoS''
                     `` :subCategory network''
                     `` :class anacl_NetQoS''
                     `` :name Network-A''
                     `` :language anACL''
                     `` :ontology  QoS)''
                       );
```

6.3.3 Step 3: Brokering phase

In the brokering phase depicted in figure 6.12, the user agent, the QoS manager, and facilitator negotiate through the following code:

```
KQML kqml = new KQML();
String rw          = new String(kManager.getInitialID());
String rw2         = new String(kManager.getInitialID());

kqml.setPerformative(``advertise'');
kqml.setParameter(``:name'', agentName);
kqml.setParameter(``:sender'', localHost);
kqml.setParameter(``:receiver'', facilitator);
kqml.setParameter(``:reply-with'', rw);
kqml.setParameter(``:language'', ``KQML'');
kqml.setParameter(``:ontology'', ``kqml-ontology'');

KQML kqml_i = new KQML();
kqml_i.setPerformative(``broker-all'');
kqml_i.setParameter(``:sender'', facilitator);
kqml_i.setParameter(``:receiver'', localHost);
kqml_i.setParameter(``:in-reply-to'', rw);
kqml_i.setParameter(``:language'', ``KQML'');
kqml_i.setParameter(``:ontology'', ``kqml-ontology'');

KQML kqml_ii = new KQML();
kqml_ii.setPerformative(``{\bf ask-all}'');
kqml_ii.setParameter(``:sender'', facilitator);
kqml_ii.setParameter(``:receiver'', localHost);
kqml_ii.setParameter(``:reply-with'', rw2);
kqml_ii.setParameter(``:language'', ``anACL'');
kqml_ii.setParameter(``:ontology'', ``QoS'');
String  content = new String(
     ``(:throughput''
     `` :delay''
     `` :jitter''
     `` :loss )''
     );

kqml_ii.setContent(content);
kqml_i.setContent(kqml_ii);
kqml.setContent(kqml_i);
```

The corresponding UML representations of these phases are:

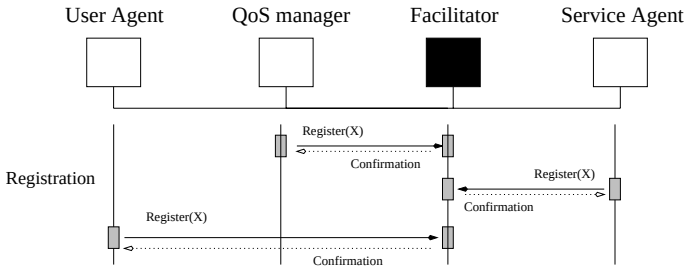


Figure 6.10: Registration phase

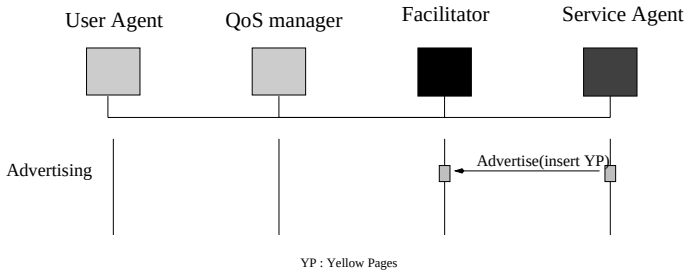


Figure 6.11: Advertisement phase

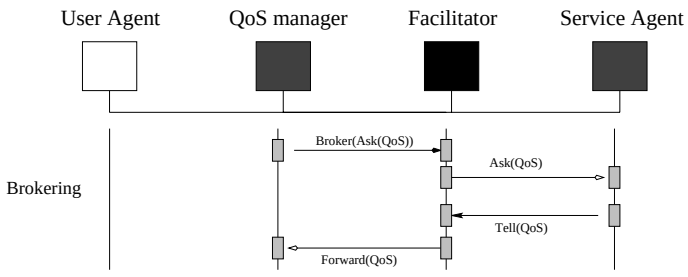


Figure 6.12: Brokering phase

6.4 Related work

A number of service discovery protocols have been implemented for different platforms but few of them deal directly with the issue of QoS. Some of these are the CORBA architecture [48] with its **Trader** and **Naming Services**, Service Location Protocol (SLP) designed by the Internet Engineering Task Force, the QoS Broker designed by Klara Nahrstedt in [44], and recently Sun Microsystem Jini.

6.4.1 The CORBA Trader and Naming Services

CORBA is a middleware that enables a *client application* to request information from an *object implementation* at the server side. The ORB core provides CORBA with the basic communications services necessary to transfer requests from clients to object implementations.
In addition, CORBA can advertise available objects and services on behalf of object implementations via a *Common Object Services Specifications* (COSS) service called **the trader Service**. With the CORBA trader service, new object implementation application servers can be added and be available at anytime and anywhere.
Services are registered with the **Naming Service** by specifying its name and object reference. A client who wishes to access the service specifies the name of the service, which the Naming Service uses to retrieve the corresponding object reference.
Services are registered with the Trader Service by specifying its service type, properties and object reference. A client who wishes to access the service, specifies the type of the service and constraints. Therefore, the Trader Service can be viewed as a yellow pages phone book. In spite of the similarities in both approaches, it is important to note that the main difference between our system and CORBA services is that we are dealing with messages which bear meaning and are organized in conversations. The players in our system are agents that are engaged in structured conversations while CORBA enables applications to exchange only objects, data structures, and propositions.

6.4.2 The Service Location Protocol

The idea of using multiple agents for the discovery of services accross a local area network has already been used by the SLP. In this model, the following agents play an important role:
- A user agent (UA) acts on behalf of a user or client in need of a service;
- A service agent (SA) declares its services to a directory agent previously

discovered;
• A directory agent (DA) accepts requests and registrations from a UA or a SA.

There are two fundamental differences between the SLP scheme and our approach: SLP uses multicast and DHCP protocols to initialize its scalable service discovery framework. However, as DHCP cannot extend to the entire Internet, SLP is unable to scale to the entire Internet. A user agent itself must send its queries to a remote DA when a service is not available locally. In contrast, our approach considers a federation of services as illustrated in figure 6.8 with several facilitators. Only facilitators may forward requests from one region to another. In addition, we use KQML messages to convey these requests accross the Internet.

6.4.3 The QoS broker

The QoS broker in [44] is an architecture for resource and QoS management which (1) orchestrates resources needed for tasks in the application and transport subsystems, (2) negotiates with network resources management, and (3) negotiate with remote QoS Broker. In this model, some protocol entities on the networked multimedia system called *broker-buyers* are resource "buyers" and others at remote sites called *broker-sellers* "sell" these resources. In addition, communications between the broker-buyer and the broker-seller are split in two entities: the *broker-sender* and the *broker-receiver*. Here, the buyer or the seller could alternatively play the role of sender or receiver.
The main difference between the QoS broker and our approach is that the QoS broker does not include any mediator between parties. QoS negotiation services like the negotiation of application QoS and the negotiation of network QoS are processed on a peer to peer, peer to group, group to peer basis through direct communication. Obviously, when the number of resources involved in the system grows, scalability becomes a big concern.

6.4.4 Jini

Jini is a network operating system for a broad range of electronic devices and software services designed at Sun Microsystems. Jini allows you to create a "federation" of devices and software components in a single distributed computing space. Although the components work together to serve a common goal, they're still identified as separate components on a network. The Jini discovery architecture is similar to that of SLP. Jini agents discover the existence of a Jini Look Up Server which collects service advertisements like the facilitators in our system. Jini agents then request services on behalf of client

software by contacting the Look Up Server. Jini uses attributes modeled as java objects to find services that match a given client requirements. However, the fact that an installation of a Java Virtual Machine needs memory and resources, Jini may not be able to run on devices with limited resources.

6.5 Summary

In this chapter, we have presented a framework for quality of service negotiation with resource discovery over the internet. The main novel contribution of this framework is the introduction of the concept of multi-agents together with KQML an agent communication language. In contrast to existing automatic resource discovery protocols like the SLP, our scheme scales to the entire Internet. To illustrate its effectiveness, we designed a prototype simulation system based on the IBM Java KQML API with several agents. The participating agents: the user agent, the QoS broker agent, the facilitator agent, and network agents interact in the KQML language. Although this approach to QoS negotiation may look attractive, the important number of facilitator could eventually be a concern. An alternative to this drawback could be to let facilitators move from host to host with necessary information.

Chapter 7

Conclusions

7.1 Research contribution

We have proposed in this thesis two different perspectives of software agent technology: mobile agents and multi-agents systems. The first stems from software engineering while the second is rooted in artificial intelligence. As we realized in the research conducted that mobile agents had limited applications in complex problem solving, we turned to multi-agent systems as a new tool for the mechanisms of QoS provision. We showed that successful multi-agent systems depend heavily on a good communication language characterized by its format, semantics, and interaction protocol. Then, we proposed a general framework and some design principles of ACLs, described their structure, issues related to their design, and some perspectives that promise to pave the way for the ideal ACL. Last, we argued that the concept of multi-agent systems together with agent communication languages could be valuable tools for QoS provision in distributed systems. To illustrate our approach, we designed a prototype multi-agent system for QoS negotiation on large scale networks. The original contribution of our approach lies in the fact that we use an ACL, namely KQML, together with middle agents called QoS manager agent and facilitator agent in resource discovery and relay of meaningful statements between users and these resource providers. In contrast to models like SLP in [50] and the QoS broker in [44] users do not need to know the location of services nor do they have to initiate any direct communication. As facilitators dwell in a given host and orchestrate services in that host, we acknowledge that in the case of the Internet with a huge number of hosts, scalability may be a serious concern.

7.2 Research perspectives

In the QoS-based MAS prototype developed, we intend to split the facilitator function into an information gatherer part and an information fetcher part. The first should act as a resident facilitator and the other should behave as a mobile agent that migrates from host to host and interacts with resident facilitators. Such an architecture will undoubtely raise the problem of communication protocol. In Search for the ideal ACL, questions related to interoperability, consensus on semantics, interaction protocols, and ontology design need to be tackled.
Several standardization efforts are being conducted in mobile agent research (OMG MASIF) as well as in multi-agent systems research (FIPA and ARPA) in order to make different systems interoperate. In industry, mobile agents-based applications and multi-agent systems-based applications may need to interact as parts of a larger system. For this reason,we are interested in bringing together Mobile Agents and multi-agent systems. However several questions need answers: should multi-agent systems make use of mobile agents ability to migrate. How should we coordinate different communication protocols ? Incorporation ACL capabilities into mobile agents could open interesting perspectives. We are looking forward to tackling these issues in future research.

In addition, we are specially interested in applying the software agent technology to the field of **Active Networks** in distributed systems. Active Networks provide a flexible framework for the initiation of user specific adaptation procedures within a network. Many active network architectures use the mobile agents technology but few have implemented the multi-agent system paradigm together with agent communication languages. The concepts of mobile agents and multi-agent systems are indeed parallel to the concepts of active packets, active nodes, and active networks. It could be interesting to consider nodes in an active network as elements of a multi-agent systems with the active packets being carried around by mobile agents. Although this approach may sound attractive many security and protocol issues must be addressed in order to have a good system. We are looking forward to do substantial research in this direction in the future.

Bibliography

[1] Aurrecoechea C. , A. Campbell, and L.Hauw. A Survey of QoS Architectures. In: *ACM Multimedia Systems Journal*, Special Issue on QoS Architectures, 1997.

[2] Barbuceanu, M. and Fox, M. S. COOL: A Language for Describing Coordination in Multi-Agent Systems, In V. Lesser (ed.), Proceedings of the First Intl. Conference on Multi-Agent Systems, AAA Press/The MIT Press, 1995, pp. 17-25.

[3] Braden, S. Clark Integrated Services in the Internet Architecture: An Overview, Request for Comment RFC-1633

[4] Bradshaw, Jeffrey M. KAoS: Towards an Industrial-Strength Open Agent Architecture, In: J. M. Bradshaw (Ed.), *Software Agents*, AAAI press/The MIT press, June 1995.

[5] Campbell, Andrew. Integrated Quality of Service for Multimedia Commumications, In: *Proceedings of IEEE INFOCOM'93*, San Francisco, USA, 1993.

[6] Cohen, Philip R. and Levesque, Hector. Communicative Actions for Artificial Agents, In: J. M. Bradshaw (Ed.), *Software Agents*, AAAI press /The MIT press, June 1995, pp 419-436.

[7] Colombetti, Marco. Semantic, normative, and practical aspects of agent communication. In: Thomas Dean (ed.), *Proceedings of IJCAI'99 Workshop on Agent Communication Languages*, Stockholm, Morgan Kaufman Publishers, San Francisco, CA, August 1999.

[8] Colombetti, Marco. Different ways to have something In: common, In: H. Christiansen T. Andreasen and H.L. Larsen (eds.), *Proceedings of the Third International Conference on Flexible Query Answering Systems (FQAS'98)*, LNAI 1495, Springer Verlag: Berlin, 1998, pp 95-109.

[9] Davies, W. H. E. and Edwards, P. Agent-K: An integration of AOP and KQML, In: *Proceedings of The CIKM '94 Workshop on Intelligent Agents*, Gaithersburg, Maryland, December 1994.

[10] Davis, Randal and Smith. Negotiation as a metaphor for distributed problem solving, *Artificial Intelligence* **20**, 63-109, 1983.

[11] Droms, R. rfc1541. *Technical Report* rfc1541, IETF, Network Working Group, October 1993.

[12] de Meer, Hermann; J-P. Richer, A. Puliafito and O. Tomarchio. Tunnel Agents for Enhanced Internet QoS, *IEEE Concurrency*, April-June,1998. pp. 30-39.

[13] de Jonge, Wiebren; M. Frans Kaashoek and Wilson C. Hsieh. The Logical Disk: A New Approach to Improving File Systems, In: *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, December, 1993, pp 15-28.

[14] Dignum, Frank. Social interactions of autonomous agents; private and global views on communication, In: *Proceedings of The 4th ModelAge workshop*, January 1997.

[15] Ferrari, D. The Tenet Experience and the Design of Protocols for Integrated Services Internetworks, *Multimedia Syst. J.*, May, 1998.

[16] Parks, David. Agent-oriented programming : A practical evaluation, Technical Report 94720, University of California, Berkeley, 1997

[17] Patil, Ramesh S. , Fikes, Richard E., Schneider, Peter F. Patel, Don Mckay. The DARPA knowledge sharing Effort: Progress Report, In: Michael Huhns and Munindar P. Singh (Ed.), *Readings In: Agents*, Morgan Kaufmann, 1998, pp 243-254.

[18] Smith, Ira A. and Cohen, Philip R. Toward a semantics for an agent communications language based on speech acts, In: *Proceedings of the Thirteen National Conference on Artificial Intelligence*. AAAI press /The MIT press, August, 1996.

[19] FIPA, Foundation for Intelligent Physical Agents. Fipa spec 2 agent communication language, *Technical Report Draft*, version 0.1, 1999.

[20] Finin, Tim; Labrou, Yannis; and Peng, Yun. The current landscape of agent communication languages, *Intelligent Systems*, **14**(2), March-April 1999.

[21] Finin, Tim; Labrou, Yannis; and Mayfield, James. KQML as an Agent Communication Language, In: J. M. Bradshaw (Ed.), *Software Agents*, AAAI press/The MIT press, June 1995, pp 291-315.

[22] Fischer S. , A. Hafid,G., Bochmann,and H. de Meer. Cooperative QoS Management for Multimedia Applications, In: *Proceeding of the 4th IEEE Int. Conf. on Multimedia Computing and Systems*, Ottawa, Canada, June 1997, pp. 303-310.

[23] Genesereth, Michael R.and Ketchpel, Steven P. Software Agents, *Communications of the ACM*, July 1994, pp 37-48.

[24] General Magic. Telescript Technology and Mobile Agents.

[25] Hafid H. and G. V. Bochmann, Quality of Service Adaptation in Distributed Multimedia Applications, *ACM Multimedia Systems Journal, Vol.6*, 1997.

[26] Hajime Tsuchitani, Osamu Furusawa. Jkqml. *AlphaWorks, IBM*, 1998.

[27] Hutchison D., Coulson G. , Campbell A. , Blair G. Quality of Service Management in Ditributed Systems, In: Sloman M.(ed), *Network and Distributed Systems Management*, Chapter 11, Addison Wesley. Reading, Mass.

[28] International Standard Organization. ISO Quality of Service Framework, ISO/IEC JTC1/SC21/WG1 N9680

[29] Jennings, N. R. et al. Managing business processes using intelligent agents, In: *Proceedings of BCS Expert Systems Conference (ISIP track)*, Cambridge, UK, 1996.

[30] Jenings, N. R. ; Sycara, Katia; Wooldridge, M. A roadmap for agent research and development, *Autonomous agents and multiagent systems*, 1, p 7-38, Kluwer, Boston, 1998.

[31] Jorg, P. Muller. The Design of Intelligent Agents - A Layered Approach, LNAI 1177, Springer Verlag, 1996.

[32] Kearney, Paul. Proposal for standard inter-agent communication language, *Technical report* SLE/DIS/97-01, Sharp Laboratories of Europe Ltd, UK, January 1997.

[33] Kone Mamadou Tadiou and Shimazu Akira. The state of the art in agent communication languages. submitted to *Knowledge and Information Systems*, 1999.

[34] Kone Mamadou Tadiou and Tatsuo Nakajima. An Architecture for a QoS-based Mobile Agent System, In: *Proceedings of 5th Int. Conf. on Real-Time Computing Systems and Applications*, Hiroshima, Japan, Sept. 1998.

[35] Labrou, Yannis; Finin, Tim; and Yun Peng. The interoperability problem: bringing together mobile agents and agent communication languages, In: *Proceedings of 32nd Hawaii International Conference on System Sciences, mini-track on Software Agents (HICSS-99)*, Maui, HI, January 1999.

[36] Labrou, Yannis and Finin, Tim. Semantics and conversation for an agent communication language, In: Michael Huhns and Munindar P. Singh (Ed.), *Readings In: Agents*, Morgan Kaufmann, 1998, pp 235-242.

[37] Labrou, Yannis and Finin, Tim. A proposal for a new kqml specification, *Technical Report* TR-CS-97-03, University of Maryland, Baltimore County (UMBC),1997.

[38] Labrou, Yannis and Finin, Tim. Semantics for an agent communication language, In: *The Fourth International Workshop on Agent Theories, Architectures, and Languages*, Providence, Rhode Island, USA, 1997.

[39] Lange, Danny and Mitsuru, Oshima. *Programming and Deploying Java Mobile Agents with Aglets*, Addison Wesley, August 1998.

[40] Linder B. Communicating rational agents, In: *A Common Formal MODEL of Cooperating Intelligent AGEnts*, *http://www.cs.uu.nl/people/bernd/Model.*

[41] Mahesh, K. Ontology development for machine translation: Ideology and Methodology, *Technical Report MCCS-96-292*, Computing Research Laboratory, New Mexico State University, Las Cruces, NM.

[42] Martin, David L. ; Cheyer, Adam J. ; and Moran, Douglas B. The Open Agent Architecture: A Framework for Building Distributed Software Systems, *Applied Artificial Intelligence* **13** (1/2), 91:128, January-March 1999.

[43] Moulin, Bernard. The Social Dimension of Interactions In: Multiagent Systems, In: Wayne Wobcke and Chengqi Zhang (Eds.), *Agent and Multi-agent Systems*, LNAI 1441, Springer Verlag: Berlin, 1997.

[44] Nahrsted, Klara and Jonathan M Smith. "The QoS Broker," *IEEE Multimedia*, pp. 53-67, Spring 1995.

[45] Nakajima, Tatsuo and Hiroyuki Aizu. Environment Server: A System Support for Adaptive distributed Applications, In: *World Wide Computing and its Applications WWWCA '98, Proceedings of the Second International Conference*,Tsukuba, Japan, March 1998, pp.142-157.

[46] Nwana, Hyacinth S. Software Agents. In: Hyacinth S. Nwana and Nader Azarmi (Eds.), *Software Agents and Soft Computing, Towards Enhancing Machine Intelligence; Concepts and Applications*, LNAI 1198, Springer Verlag: Berlin, December 1996.

[47] Oliveira, Luiz A.G. , Paulo C. Oliveira and Eleri Cardozo. An Agent-Based Approach for Quality of Service Negotiation and Management in distributed systems, In: *Mobile Agents, Proceedings of the First International Workshop,MA '97*, Berlin, Germany, April 1997, pp 1-12.

[48] Randy Otte, Paul Patrick, Mark Roy. *Understanding Corba, The Common Object Request Broker Architecture*, Prentice Hall, 1996.

[49] Peng, Yun; Finin, Tim; Labrou, Yannis, Long, Tolone, Agent-based Approach for Manufacturing Integration: The CIIMPLEX Experience, *Applied Artificial Intelligence* **13**(1/2), 39-64, 1999.

[50] Perkins C. Slp white paper, Technical report, Sun Microsystems, 1998.

[51] Radouniklis, Nikolaos; Baumann, Joachi; and F. Hohl. Kurth Rothermel and Radu Popescu-ZeletIn: (Eds.), *Communication Concepts for Mobile Agent Systems*, LNCS 1219, Springer Verlag: Berlin, April 1997, pp 123-135.

[52] Rosenschein, Jeffrey S. Consenting Agents: Negotiation Mechanism for Multi-Agent Systems, In: Ruzena Bajcsy (ed.), *Proceedings of the Thirteenth International Joint Conference on AI, IJCAI'93*, Chambery, France, Morgan Kaufman Publishers, San Francisco, 1993, pp 792-799.

[53] Russel, Stuart and Norvig, Peter. *Artificial Intelligence, A Modern Approach*, Prentice Hall, 1995.

[54] Sadek, M. D.; Bretier, P.; and Panaget, F. ARTIMIS: Natural Dialogue Meets Rational Agency. In: *Proceedings of the 1997 International Joint Conference on Artificial Intelligence (IJCAI'97), Palo Alto*, Morgan Kaufmann Publishers, 1997.

[55] Sadek, M. D.; Bretier, P.; Cadoret, V.; and Cozannet, A. A cooperative spoken dialogue system based on a rational agent model: A first implementation on the AGS application. In: *Proceedings of the ESCA/ETR Workshop on Spoken Dialogue System: Theories and Applications*, Vigso, Denmark, 1995.

[56] Searle, J. R. *Speech Acts: An essay in the philosophy of language*, Cambridge University Press, Cambridge, 1969.

[57] Shen, W. and Xue, D. Agent-based manufacturing enterprise infrastructure for distributed integrated intelligent manufacturing systems, In: *Proceedings of the Practical Application of Intelligent Agents and Multi-agent Systems*, London, UK, 1998, pp. 533-550.

[58] Shoham, Yoav. Agent-oriented programming, *Artificial Intelligence* **60**(1), 51-92, 1993.

[59] Singh, Munindar P. Towards a formal theory of communication for multi-agent systems, In: John Mylopoulos and Ray Reiter (eds.), *The Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, Sidney, Australia, 1991, pp 69-74.

[60] Singh, Munindar P. Multiagent Systems - A Theoritical Framework for Intentions, Know-How and Communications, LNAI 799, Springer Verlag, 1994.

[61] Singh, Munindar P. Agent communication languages: Rethinking the principles, *IEEE Computer*, 31(12), 40-47, December 1998.

[62] Singh, Munindar P. A Social Semantics for Agent Communication Languages, In: Thomas Dean (ed.), *Proceedings of the ACL Workshop, IJCAI'99*, Stockholm, Sweden, Morgan Kaufman Publishers, San Francisco, CA, August 1999.

[63] Stanford KSL Network Services. Ontolingua, ontologies editor. *KSL Home Page*, 1995.

[64] Tassel, Jérôme. Quality of Service(QoS) adaptation using Reflective Java. British Telecom Technical Report.

[65] Thomas, S. Rebecca. The PLACA Agent Programming Language, In: *Intelligent Agents, ECAI-94*, Proceedings of the Workshop on Agent Theories, Architectures, and Languages, 1995, pp. 355-370.

[66] Shoham, Yoav. AGENT0: a simple agent language and its interpreter, In: *Proceedings of AAAI*, Anaheim, 1991, pp. 704-709.

[67] Torrance, M. and Viola, Paul. The AGENT0 manual, In: *Stanford Technical Report CS-TR-91-1389*, 1991, pp. 704-709.

[68] Waddington D. and David Hutchison. End-to-end QoS Provisioning through Resource Adaptation, In: *Proceedings of the 8th IFIP Conference on High Performance Networking - HPN'98*, Vienna University of Technology, Vienna, Austria, September 1998.

[69] Waddington D. and David Hutchison. End-to-end QoS Provisioning through Resource Adaptation, In: *Proceedings of the 8th IFIP Conference on High Performance Networking - HPN'98*, Vienna University of Technology, Vienna, Austria, September 1998.

[70] Waddington D., Chritopher Edwards and D. Hutchison Resource Management for Distributed Multimedia Applications, In: *Proceedings of the Second European Conference on Multimedia Applications, Services and Techniques - ECMAST*, Milan, Italy, May 1997; LNCS 1242.

[71] Waddington D. , Chritopher Edwards and D. Hutchison. Resource Management for Distributed Multimedia Applications, In: *Proceedings of the Second European Conference on Multimedia Applications, Services and Techniques - ECMAST*, Milan, Italy, May 1997; LNCS 1242.

[72] Waddington D. and G. Coulson, A Distributed Multimedia Component Architecture, In: *Proceeding of the first International on Enterprise Distributed Computing*, Gold Coast, Australia, October 1997.

[73] Werner, Eric. Cooperating Agents: A Unified Theory of Communication and Social Structure, In: L. and Huhns, M. N (Eds.), *Distributed Artificial Intelligence*, volume II, Pitman/Morgan Kaufmann Publishers, July 1997.

[74] Wooldridge, Michael. Verifiable semantics for agent communication languages, In: *International Conference on Multi-Agent Systems (ICMAS'98*, August, Paris, France, 1998.

[75] Yokote Yasuhiko. The Apertos Reflective Operating System: The concept and Its Implementation, In: *ACM OOPSLA'92*, 1992, pp. 414-434.

Publications

[1] Mamadou Tadiou Kone and Tatsuo Nakajima. An Architecture for a QoS-based Mobile Agent System, *Proceedings of the Fifth International Conference on Real-Time Computing Systems and Applications (***RTCSA'98***)*, Hiroshima, Japan, October 27-29, 1998, pp.145-148.

[2] Mamadou Tadiou Kone and Tatsuo Nakajima. Mobile Agents and QoS Adaptation in Distributed Multimedia Systems, *Proceedings of the Third World Multiconference on Systemics Cybernetics*, Orlando, Florida, July 31 - August 4, 1999. Volume 1, pp. 570-576.

[3] Mamadou Tadiou Kone, Akira Shimazu, and Tatsuo Nakajima. The State of the Art in Agent Communication Languages, *Knowledge and Information Systems* (**KAIS**), an international journal, Springer Verlag, May 2000.

[4] Mamadou Tadiou Kone and Tatsuo Nakajima. A Multi-agent System Architecture for QoS Management in Distributed Multimedia Systems, *Smartnet'99*, Bangkok, Thailand, Thongchai Yongchaeron, Finn Arve Aagesen, and V. Wuwongse (Eds.) *Intelligence in Networks*, Kluwer Academic Publishers, November 2000, pp. 121 - 134.

[5] Mamadou Tadiou Kone and Tatsuo Nakajima. An Agent-based Framework for Large Scale Internet Applications, *Networking 2000*, Paris, France. Lecture Notes in Computer Science(LNCS), Springer Verlag, Berlin, May, 2000.

[6] Mamadou Tadiou Kone and Tatsuo Nakajima
A Multi-agent System Support for QoS Negotiation, *Proceedings of the JSSST 3rd Annual Workshop on Systems for Programming and Applications,* **SPA 2000**, Japan, March 2000.