An Agent-Based Framework for Large Scale Internet Applications

Mamadou Tadiou Kone[1] and Tatsuo Nakajima[2]

[1] Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Tatsunokuchi-machi, Nomi-gun, Ishikawa-ken, Japan 923-12
mamadou@jaist.ac.jp

[2] Waseda University, Shinjuku-ku, Ookubo, Tokyo 169-8555, Japan.
tatsuo@mn.waseda.ac.jp

Abstract. The idea of a software entity that performs tasks on behalf of a user across the Internet is now well established. We introduce in this paper a new approach to service discovery and QoS negotiation over the Internet. Our approach presents a framework for service discovery and QoS negotiation at the network level that rely on two concepts: multi-agent systems and agent communication languages (ACL). In this framework, a *user* and *service agents* engage in a structured communication through the m ediation of a *QoS Broker Agent* and a *Facilitator Agent*. Here, the Facilitator Agent acts on behalf of several service agents. It acquires information from these service agents and acts as a single point of contact to supply this information to the User Agent via the QoS Broker Agent. A number of service discovery protocols like the Service Location Protocol (SLP), and Sun Microsystem's Jini has been designed for restricted environments and do not scale to the entire Internet. In order to pro vide an infrastructure for large scale Internet applications, we designed a prototype multi-agent system that is able to discover resources and negotiate QoS at the network level.

Keywords: Mutli-agent systems, Agent Communication Languages (ACL), Knowledge Query and Manipulation Language (KQML), Quality of Service (QoS), Internet.

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) [4] 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) [1] is available or multicast is supported at the network layer. Neither

DHCP nor multicasting extend 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 paper is to deal with two important limitations in resource management for large scale applications: scalability and communication costs. We propose in this paper a framework that relies on the concepts of Multi-agent Systems and Agent Communication Language (ACL) (here the Knowledge Query and Manipulation Language (KQML) described in [3]). 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.

Following this introduction, we state in section 2 the problem we intend to examine. In section 3, we describe the concepts and protocols underlying our multi-agent system based QoS negotiation scheme. In addition, we give the implementation details of our framework in the same section. Some issues and perspectives are proposed in section 4 and then related works are presented in section 5. Finally, we conclude in the last section 6.

2 Agent-Based Systems

2.1 Multi-agent Systems

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. On the other hand, the second states that *multi-agent systems* are distributed computing systems composed of several interacting computational entities called agents. These constituent agents have capabilities, provide services, can perceive and act on their environment. Service components involved in a QoS provision are modeled as this type of agent.

2.2 Agent Communication Languages

An agent communication language (ACL) 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.

The Knowledge Sharing Effort group designed one example of ACL, the Knowledge Query and Manipulation Language (KQML) described in [3]. Our framework uses the KQML language, which is made of three layers (figure 1) : the communication layer, the message layer, and the content layer. A KQML message has the following structure:

Communication Layer: sender, receiver, msg id
Message Layer: performatives, msg format
Content Layer: ontology, content language

Fig. 1. KQML three layers structure

```
(tell
        :sender QoS-manager
        :receiver User
        :language Prolog
        :in-reply-to id1
        :ontology QoS-ontology
        :content "available(resource,URL)"
   )
```

Here, *tell* is called a *performative*, *:sender*, *:receiver*, *:language*, *:in-reply-to*, and *:ontology* are parameters. The QoS manager informs (*tell*) the user about the availability of a resource on the Internet by using Prolog as a content language. There are two types of agent communication: the direct communication relates a sender agent with a known receiving agent and the mediated communication

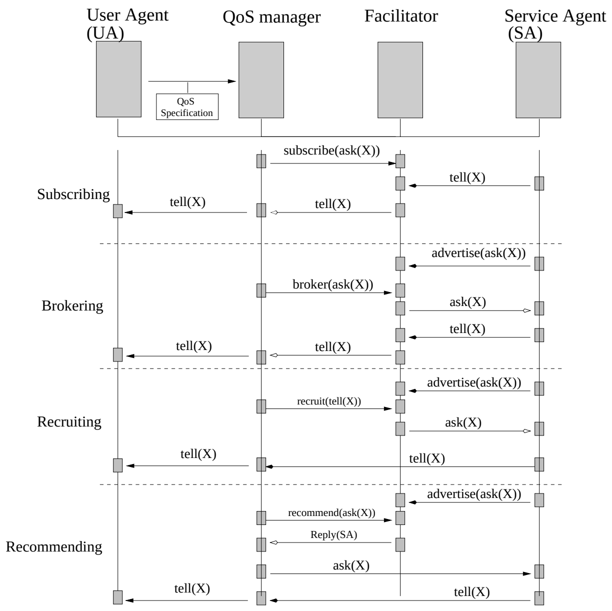


Fig. 2. Facilitator mediated QoS negotiation

illustrated in figure 2 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 on one hand, needy agents subscribing to services and on the other hand facilitators brokering, recruiting, and recommending agents that registered their identities and capabilities.

3 Multi-agent System-Based QoS Negotiation

3.1 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 components and building potential configurations,
• classifying these configurations, and
• 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 burden of identifying and selecting appropriate resources on a large scale networks like the Internet. This situation adds a substantial overload on the operation of the QoS manager. In addition, services and resources may not be guaranteed consistently.
• *Second*: When the number of entities involved in a direct communication with the QoS manager is modest, communication costs remain reasonable. However, in the Internet, these costs become prohibitive even with auxiliary local QoS managers.

To address these scalability and communication costs issues, we propose a framework for QoS negotiation illustrated in figure 3 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, a facilitator agent, and service agents (network agents) communicating in KQML.

3.2 Concepts and Framework Description

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

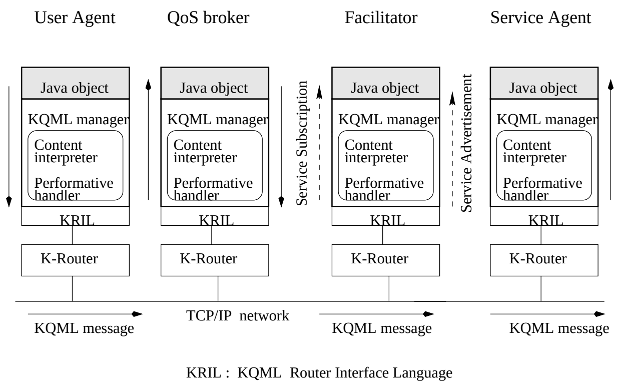


Fig. 3. System architecture

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. Our architecture uses four ontologies:
  ∗ a *yellow page* ontology for service advertisement by service agents,
  ∗ a *white page* ontology for finding the location of an agent given its name,
  ∗ a *general QoS* ontology for the current domain knowledge,
  ∗ and a *QoS broker* ontology for asking network options by the user and QoS broker.

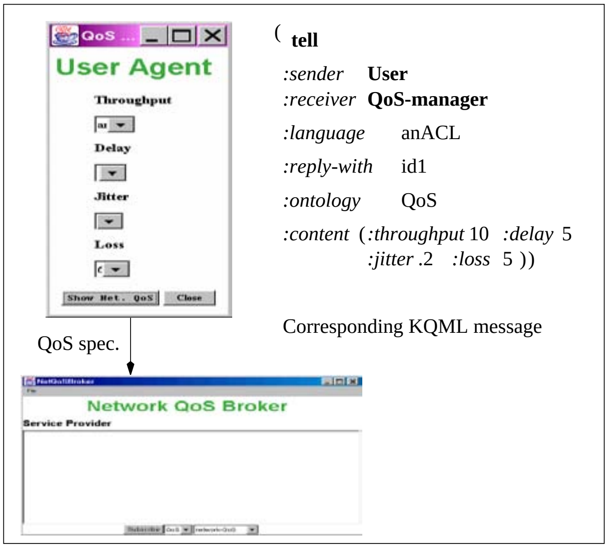
• A *KQML manager* encompasses:
  ∗ *Conversations* that group messages with a common thread identified by the ":reply-with and :in-reply-to" parameters;
  ∗ *content interpreters* that handle incoming and related response messages according to the ACL, content language and ontology associated to these messages;
  ∗ *performative handlers* that process a message performative in conjunction with its ACL, content language and ontology.

QoS 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 the required level of service.

• The user agent sends to the QoS manager agent a KQML message with required levels of service expressed in appropriate QoS parameters like in figure 4.

• The QoS manager needs to identify all components necessary to build a configuration that can sustain an activity. For this purpose, its 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



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

Fig. 4. User and QoS Broker interaction

appropriate resources. A structure of this KQML message and agent interaction is shown in figure 5.

• The facilitator agent acts as a resource broker that

  ∗ recruits, recommends appropriate service agents (application, system, and network agents) to the QoS manager;

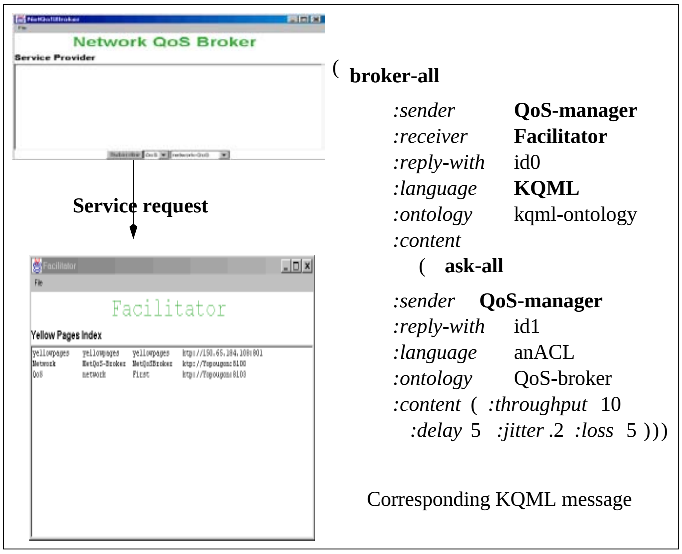
  ∗ forwards the QoS manager messages (brokering and recruiting) to suitable service agents; and

  ∗ informs (on subscription) or recommend to the QoS manager service agents that fulfill its requirements.

• All servive agents (network agents) advertise their capabilities to the the facilitator agent upon registration. Upon request from QoS broker, the facilitator supplies the identities and loccations of necessary network resources. At last, the user may view on an appropriate interface the available resources.

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.

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 7. 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.



```
yellowpages  yellowpages   yellowpages  ktp://150.65.184.108:801
Network      NetQoS-Broker NetQoSBroker ktp://Yopougon:8100
QoS          network       First        ktp://Yopougon:8103
```

```
( 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

Fig. 5. QoS Broker and Facilitator interaction

3.3 Implementation

In experimenting with this model of resource discovery and QoS negotiation, we designed a prototype in the JAVA language to simulate QoS negotiation between several agents at the network level. That is to say, to illustrate our approach, a user agent and network agent communicate via a QoS broker and a facilitator in terms of network parameters only. First, local negotiation is considered, then

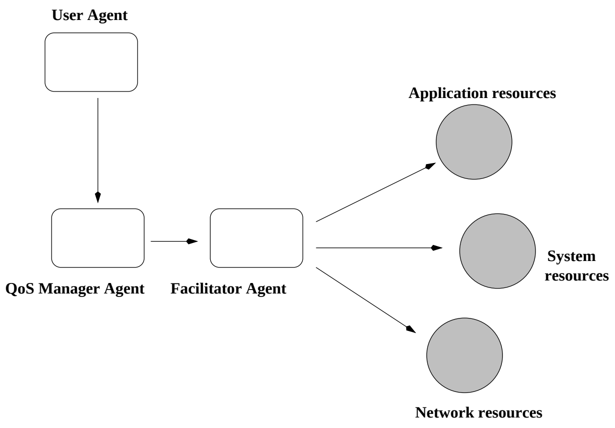


Fig. 6. Local QoS negotiation with a single facilitator dealing with resources inside a given host.

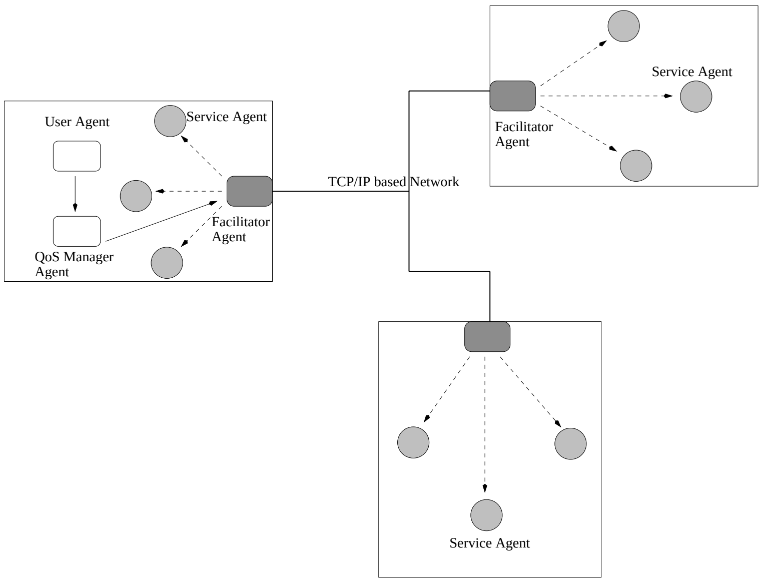


Fig. 7. Large scale QoS negotiation with several facilitators involved in the negotiation process accross the Internet.

it is extended to remote locations across the Internet.
The implementation of our prototype includes the following tools:
- The Java-based KQML API called *JKQML* in [9]. The JKQML API with its structure in figure 8 adapted from [9] provides a platform for designing KQML-enabled agents. JKQML is based on the JAVA language and provides interoperability to software that needs to exchange information and services.
Handling KQML messages involves the following steps:

1. Instantiating a KQML manager with the method:
`public KQMLManager(String agentName, String protocol, int port);`
2. Managing protocol handlers with the method:
`public void addProtocol(String protocol, int port);`
3. Managing content interpreters with the method:
`public void addContentInterpreter(String acl, String language, String ontology);`
4. Managing performative handlers with the method:
`public void addPerformativeHandler(String acl, String language, String ontology, String performative, PerformativeHandler ph);`
5. Managing conversation termination with the method:
`public void setConvCleanupHandler(ConvCleanupHandler c).`

- We used the Stanford KSL *Ontolingua* ontology editor at [8] to design both the general QoS ontology and the QoS-broker ontology used by the language interpreter. Then, we extended our simulation program to a larger TCP/IP network with facilitators at different locations communicating in KQML. A couple of

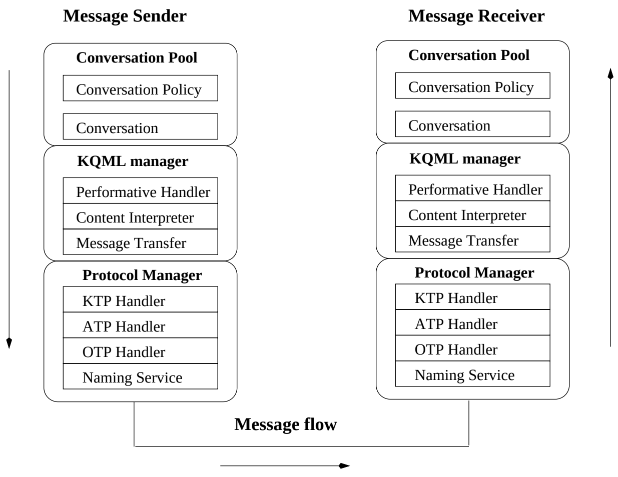


Fig. 8. Structure of JKQML

networks with different characteristics (throughput, delay, jitter, and loss) were discovered successfully and displayed on a local user interface. Figure 5 and fi gure 4 illustrate some transactions between the participating agents.

4 Issues and Perspectives

In an open and heterogeneous environment like the Internet, agents that interact and coordinate their activities face some major challenges:
- How can they find one another and specially, locate the facilitators? As the number of facilitators grows, finding their location becomes a real concern. The idea of introducing a *facilitator directory* that forwards external inquiry to all facilitators across the Internet could address this problem.
- Although many ACLs exist today, the communication language chosen should express concisely the message content of an agent. That is to say, the message semantics of an ACL must be consistent across platforms.
- With any kind of message transport protocol (KQML transport protocol (*ktp*) or agent transport protocol (*atp*)), the issue of fault tolerance due to network failure remains. Multi-agent systems must rely on a robust and reliable environment. However, the heterogeneous nature of the Internet offers no guaranty.
In addition to negotiation on the network layer, we are looking forward to extending our model to the application and system layers as well. This way, with a suitable QoS translation scheme between these layers, it is possible to cover a complete end-to-end QoS negotiation.
We intend to investigate the alternative of mobile agents as a message transport protocol. Enabling facilitators to move around the network, deliver information and collect advertisements like mobile agents is an option we are interested in. These *mobile facilitators* can interact on site with local QoS brokers and ser-

vice agents. In addition, as the new Foundation for Intelligent Physical Agents (*FIPA*) ACL standard is emerging, we are looking forward to implement our model in this language.

5 Related Work

A number of service discovery protocols have been implemented for different platforms. Some examples are the Service Location Protocol (SLP) designed by the Internet Engineering Task Force, the Dynamic Host Configuration Protocol (DHCP), the CORBA architecture [7] with its *Trader* and *Naming Services*, and recently Sun Microsystems's Jini.

5.1 The Service Location Protocol (SLP)

The idea of using multiple agents for the discovery of services across a local area network has already been used by the SLP. In this model, a user agent (UA) acts on behalf of a user or client in need of a service while a service agent (SA) declares its services to a directory agent previously discovered. In addition, 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 7 with several facilitators. Only facilitators may forward requests from one region to another. In addition, we use KQML messages to convey these requests across the Internet.

5.2 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. 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*. 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. Whereas services are registred 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.

5.3 Jini

Jini is a network operating system by Sun Microsystems aimed at a broad range of electronic devices and software services assembled 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 softwares by contacting the Look Up Server.

6 Conclusion

In this paper, we have presented a framework for resource discovery and quality of service negotiation over the Internet. The main point is that our framework relies on the concept of multi-agent systems and agent communication language. In contrast to automatic resource discovery protocols like the SLP, our scheme scales to the entire Internet. To illustrate its effectiveness, we designed a prototype based on the IBM Java KQML API with several agents: user agent, QoS broker agent, facilitator agent, and network agents that interact in the KQML agent communication language. Although this approach may look attractive, its main drawback lies in the important number of facilitator agents that the system must deal with. In the future, we intend to let these facilitators move from host to host with information just like mobile agents.

References

1. Droms R.: rfc1541. Technical report, IETF, Network Working Group, *http://www.cis.ohio-state.edu/htbin/rfc/rfc1541/.html*, October 1993.
2. Genesereth R. Michael, and Ketchpel P. Steven: Software agents. *Communications of the ACM*, 37:48, July 1994.
3. Patil, Ramesh S. , Fikes, Richard E.: The DARPA knowledge sharing Effort: Progress Report, In: Michael Huhns and Munindar P. Singh (Ed.), *Readings in Agents*, Morgan Kaufmann, 1998, pp 243-254.
4. Perkins C.: SLP white paper, Technical report, Sun Microsystems, *http://playground.sun.com/srvloc*, 1998.
5. Keith W. Edwards.: *Core Jini*, Sun Microsystem Press, June 1999.
6. Kone Tadiou Mamadou, Akira Shimazu, and Tatsuo Nakajima : The State of the Art in Agent Communication Languages. (submited) to *Knowledge and Information Systems*, 1999.
7. Randy Otte, Paul Patrick, Mark Roy : *Understanding CORBA, The Common Object Request Broker Architecture*, Prentice Hall, 1996.
8. Stanford KSL Network Services : *Ontolingua*, ontologies editor. *http://www-ksl-svc.stanford.edu:5915/*.
9. Tsuchitani Hajime and Furusawa Osamu : JKQML. *AlphaWorks, IBM*, 1998.