LOAD BALANCING USING MOBILE AGENT APPROACH

MASTER OF PHILSOSPHY
RESEARCH PROJECT FIRST-TERM REPORT

SUPERVISOR

Professor LYU, Rung Tsong Michael

PREPARED BY

WONG TSZ YEUNG (00017840)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE CHINESE UNIVERSITY OF HONG KONG

Contents

List of Figures

Abstract

The agent community has 2 groups of people doing active research. One is the multi-agent. Another is the mobile agent. Multi-agent system (MAS or AS) is a group of programs which are stationary on their own computing environment. They communicates among themselves to fulfill a given task. Mobile agents are programs which may be dispatched from a client computer and transported to a remote server for execution. In this report, we will discuss the differences between these 2 approaches . On the other hand, load balancing is an old research topic. Nevertheless, it is widely used on the construction of clusters of computers. In this report, a survey on load balancing and corresponding approximation algorithm will be illustrated. In nowadays cluster systems, nearly all of them are implemented on homogeneous computers in the sense that the computers are running on the same type of operating system. In this report, we will discuss on the possibilities to have a cluster of computers running on heterogeneous operating systems. Using mobile agent in constructing such a system seems to be a good solution. To conclude our discuss, we will have the pros and cons towards this issue.

1 Multi- and Mobile Agents

The research of software agents has been going on active research on the current decade. There are many conferences holding annual on the software agents including the **International Conference on Multi-Agent Systems** (ICMAS), the **International conference on the Practical Application of Intelligent Agents and Multi-Agent Technology** (PAAM), and the **International Conference on Autonomous Agents** (AA). These conferences are separated into 2 domains: **multi-agent systems** and **autonomous(or mobile) agents**.

The breakdown of this section is as follows. Section 1.1 revises the multi-agent technology and briefly evaluates it. Section 1.2 discusses the topics in mobile agents [4] .

1.1 Multi-Agent System, MAS

1.1.1 Introduction

The goal of multi-agent system (MAS) is:

> Creating a system that interconnects separately developed agents, thus enabling the ensemble to function beyond the capabilities of any singular agent in the system [4].

This implies that an agent in a MAS have to cooperate with other agents in the same MAS in order to perform a particular function. Also, this also implies that all the agents in a MAS can be implemented (or provided) by more than one vender (or programmer). Let us illustrate the multi-agent systems hypothesis in a real application.

Consider the case of a businessman wishing to arrange a trip from Hong Kong to New York. This is the sort of service that people can delegate this task to their *personal travel software agents*. This is the just the same as the case that the boss assigns his/her secretary to arrange such a trip. To arrange this trip, we have to decide:

1. Which airline(s) should we choose to take?
2. Which hotel(s) should we choose to stay?
3. Do we have to rent car(s)?
4. ...etc

In the case of a human agent, say the secretary, he/she may visit all the airline companies to arrange the flights and the itinerary of the trip. Or, he/she may *consult* other human agents, say a flight agent, who in turn may consult other agents.

However, there are a lot of service and information providers (or agents) are available, they will be time consuming for human agents to consult other flight agents one by one. Therefore, the *promise* of the MAS is that the personal travel agent (PTA) will negotiate with other software agents representing the information or service providers. In this way, your itinerary is generated with minimal or possibly without any human intervention.

This is truly a multi-agent problem where the inter-operations among separately developed agents provides a service that is beyond the capability of any agent in this set up. Economically speaking, all these agents have comparative advantages over each other due to *specialization*, and trading their services is good for all other agents.

1.1.2 Problems and Solutions

Information Discovery Problem

In the travel scenario, the PTA has to obtain travel information such as which airline has the suitable flight(s). For a human agent, one will simply go to *Yahoo!* to search for the relevant information. However, for a software agent, it needs some kinds of *yellow page* that lists different kinds of services. This is called **Information Discovery Problem**.

In a MAS, the solution is divided into 2 parts:

- Discover; and
- Locate

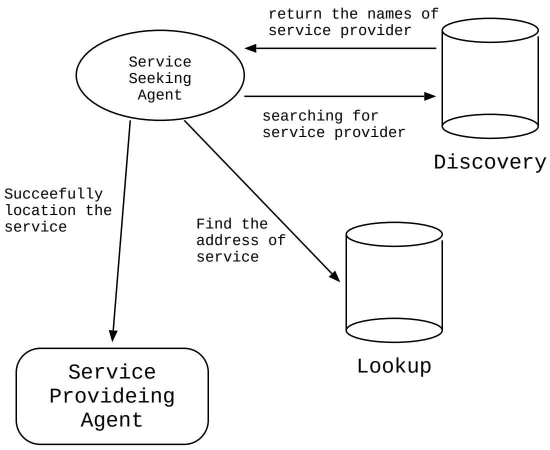

Figure 1: Discovery and Location Service.

Discovery service is just a *yellow page* service: you ask for a particular kind of service, the discovery service will list all the available agents(service providers) to you. After choosing the suitable service provider, you will need the **location service**, which is a *white page* service. The location service will return the address of the service provider to you.

In the Jini environment [7], it has a similar solution. The discovery and the location services are provided by the *discovery server*.

Communication and Ontology Problem

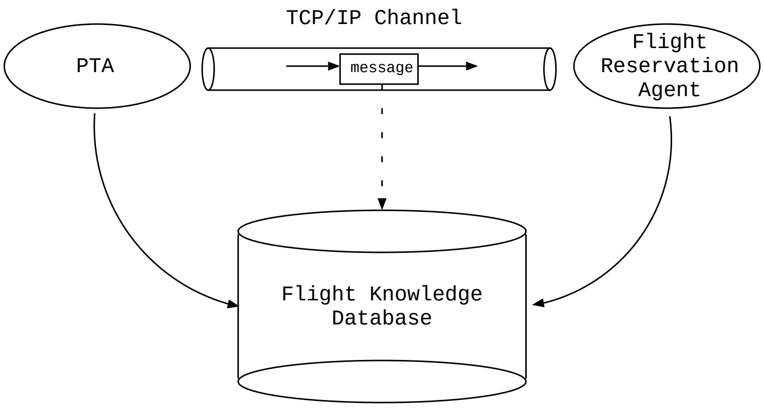

Figure 2: Communication channel, language and ontology problems.

The PTA in our example must communicate with the flight reservation system when it has found the chosen service provider. Then, both parties must adopt a common transport protocol that enables the requests and results transmissions. Fortunately, we can rely on the TCP/IP protocol.

Next, you can use TCP/IP protocol to transport our messages. However, what should our language of communication be? For example, on the Internet, the most important communication language is HTML. Nevertheless, HTML is only for presenting information instead of communicating between parties. An agent-to-agent language is preferable. However, there is no standard language exists to express the structure and content of messages. This problem requires the communicating parties to agree on what instructions, assertions, requests etc. will be supported, and what syntax is used. A number of inter-agent communication languages have been proposed, such as **KQML** *(Knowledge Query and Manipulation Language)* [8] and **FIPA ACL** *(Agent Communication Language)* [9].

The last question is the ontology problem. For the PTA and flight reservation system to interact, both systems must agree on and then share a common definition of travel-related concepts. The ontology (or concept definitions) specifies the terms each party must understand and use during communication, identities of airlines, types of tickets, etc. Creating an ontology involves explicitly defining every concept to be represented. In our example, we have to define what are the attributes of a ticket, what does each attribute means?

The most important concern of this problem is that there are less research addressing this issue. This problem is at the core of the agent inter-operability issue. Without a common knowledge base, how can different agents communicate among themselves? For example, '*first leg*' has a clear meaning within the travel context, however it may mean something else in another context. In developing a MAS, programmers may choose to use KQML or ACL as a communication medium. However, every system may only share their own knowledge within themselves. Therefore, we cannot plug another agent into the current existing MAS, unless we are using the same knowledge base.

1.2 Mobile Agent, MA

1.2.1 Introduction

Mobile agent has the same goal for as the multi-agent system:

> Users can delegate tasks to an agent. The agent will do the tasks for you.

Instead of passing message among agents, mobile agent can migrate across the network (hence mobile) representing users in various tasks (hence agents) [2].

Mobile agent seems to be a good solution in developing distributed system. Danny Lange and Mitsuru Oshima had stated **seven good reasons for using mobile agents** [3] :

- They reduce the network load.
- They overcome network latency.
- They encapsulate protocols.
- They execute asynchronously and autonomously.
- They adapt dynamically.
- They are naturally heterogeneous.
- They are robust and fault-tolerant.

We can location different application domains that mobile agents have potential deployment [2].

1. Data-intensive applications, for example information retrieval. Usually, when an user wants to process a large volume of data, he/she seldom download all the data and process them. A mobile agent will be helpful in this case. The user sends an agent to the data source, then the agent will process and filter the data locally. This saves a lot of network transmission.

2. Disconnected computing, for example laptops and PDAs. Nowadays, these devices already have the ability to connect to the network. However, they frequently disconnect from the network. Mobile agent can be deployed in this case as it can execute asynchronously. An user can send an agent before the computer disconnects from the network. When the computer connects to the network again, the user can collect back to the results processed by the agent.

However, sad to say, there is no application that would not exist without mobile agents [4]. Every mobile agent application will have its client-server counterpart such as RPC or sockets.

1.2.2 Pros and Cons Using Mobile Agent

Mobile agent also has the problems in MAS. The following paragraphs will illustrate other issues in mobile agents [4].

Agents can provide better support for mobile clients

Pros : As mobile agent can work asynchronously, we just need a short period of time to connect to the network while we are launching an agent. Even when connected, mobile agents have only relatively low-bandwidth connections. It can also save storage on the client side because mobile agent has the ability to perform information retrieval and filtering at a server. It only returns relevant information to the client.

Cons : It is less clear that the entire network's servers architecture needs to be adapted to mobile agent to meet the needs of the clients. One would prefer to solve this problem at the edge of the network and make mobile clients as robust as non-mobile clients by providing proxy clients at the edge of the network [4].

Agent-based transactions avoid the need to preserve process state

Pros : The need to preserve the entire process state at a client and a server during each flow of a complex operation adds considerably to the burden of client-server computing. Unless the client and server applications provide methods for making the RPC state persistent, which adds a significant performance overhead, failure of either client or server results in the loss of at least one and possibly many operations and recovery mechanisms at either end may generate useless network traffic. The ability of the mobile agent to carry its state around with it appears to relieve the sending computer of the the need to preserve state.

Cons : Although there is an advantage here in the use of mobile agents in terms of the robustness of client-server operations, it raises also an overhead question and also challenges for the efficient design of mobile agents. it is likely that similar robustness could also be achieved with alternative client-server methods.

Agents enable electronic commerce

Pros : Mobile agent offers a number of possibilities for electronic commerce [4].

1. The agent can express the application-level protocol required to perform a transaction. Mobile agent is a good method for vendors to distribute the client end of a transaction protocol in a device independent way.
2. Alternatively, the mobile agent may be able to present the consumer's desire as a query to a number of potential vendors to determine degree of match, price, availability, etc.
3. The agent can provide a secure mechanism for the transaction by providing authentication, and privacy (or authorization).
4. The agent can provide a transaction currency for settlement.

Among the above four possibilities, the last two can be replaced by secure RPC or other kinds of secure transmissions, such as SSL (secure socket layer). The transcation can be done securly with SET (Secure Electronic Transaction) protocol.

The first possibility offers the vendor to distribute clients via network, preferably to targeted consumers. However, this exposes the 'dark side' of mobile agent: junk clients, virus clients, dishonest clients.

The second possibility provides a new opportunity, not readily implementable by conventional methods, which is to the advantage of the consumer. He/she is not locked into dealing with a particular vendor. He/she is free to choose every vendor inside the mobile agent system. This provides the client a great flexibility in choosing services as well as vendors. For example, consumers can 'blacklist' a particular vendor, or he/she can choose the vendor that sells products at the lowest price.

Cons : There are very little technical disadvantages here. However, we can address several commercial issues that hinders the growth in using mobile agents in electronic commerce.

- Difficulty of propagating agent execution environments onto large number of third-party servers. In order to receive and execute incoming agents, servers need to install the agent execution environments. However, not all vendors will trust and use the agent execution environments.
- Willingness of the third-party server providers to permit users the ability to customize server behavior. As mobile agents are free to run whatever code on the server, this will be a great risk for the server to receive malicious agents that want to hack into the server or perform unexpected instructions.
- Willingness of the third-party server providers to support the computational load of mobile agent. That is undesirable to receive agents that introduce heavy load to the server. What if an agent is wrongly implemented and consumes all the server's resources (e.g. recursive forks)?

- Willingness of the consumers to trust the security issues implemented. Technically, it is not a big deal to implement a complex and secure transaction protocol. However, the thing that really matters is on the consumer side. What if the consumers are not willing to trust the security protocol? This problem is not only the problem for mobile agent e-commerce application, but other applications will also face this problem.

Above only mentions a small portion of the reality. Many problems still remain.

2 Load Balancing

2.1 Introduction

Load balancing is an *ancient* research topic. Suppose our system consists of n servers which are having the same physical characteristics. The n servers are providing the same type of service. For example, we have n web servers serving the web surfers. In order to fully utilize every server, we will split the users to different servers. However, we cannot let the users to wait. The more they have to wait, the more upset they will be. In order to *minimize the maximum waiting time* of every customer, we have to balance the loads among the servers. This is called **Load Balancing**.

2.1.1 How to balance

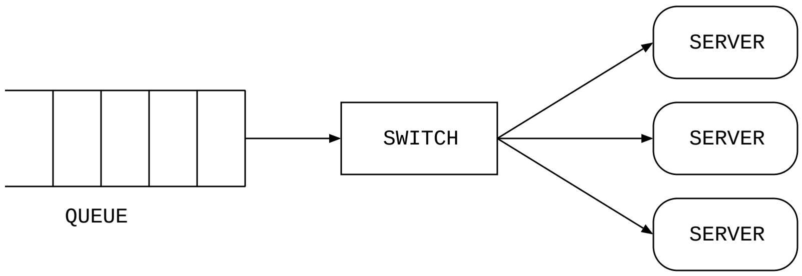

Figure 3: Load balancing system.

The above figure shows a typical structure of a load balancing system. Before beginning our discussion, we impose the following assumptions:

- There are static number of customer arrivals, say N.
- The service requested by customers are heterogeneous, i.e. different service rates are needed.
- We do not know when a customer will arrive.
- When a customer arrives, we do not what kind of service he/she is requesting (i.e. we do not know the service time).
- Whenever the server finishes serving a customer, the customer will leave the system.

The goal of this system is :

> To minimize the maximum time needed, i.e. minimize the time when the N^{th} customer leave system

We may have the following set of rules :

1. Initially, the queue is empty and the servers are idle. Then, an incoming customer will be served by an arbitrary **idle** server.

2. Consumers continues to enqueue and are served by the server until there is no idle server.
3. Whenever there is no idle server, the switch will calculate which server has the *minimum load*. Then, it assigns the customer at the head of the queue to be executed on the minimum-load server.
4. This continues until the queue is empty and there is no more customer arrival.

The above scheme adopts the **greedy algorithm**. This set of rules seems to be a very good one. However, there are proofs showing that this scheme is **not an optimum solution**. To change our scheme, we can :

- Change the behavior of the switch.
- Change the whole design such as using different queueing models.

2.2 Bounds for Different Balancing Rules and Designs

2.2.1 System Design

In 1966, Graham [5] has derived such bound. His model is different from our previous design.

1. Initially, we are given n servers, or processors, and a set of m tasks $T = \{T_1, T_2, ..., T_m\}$.
2. We are also given a partial-order $\prec$ on T and a function μ: $T \to [0, \infty)$.
3. Once a processor P_i begins a task T_j, it works without interruption on T_j until completion of that task, taking altogether $\mu(T_j)$ units of time.
4. It is also required that if $T_i \prec T_j$, then T_j cannot be started until T_i is completed.

The system works as follow:

1. We are given a linear ordering $L : (T_{k_1}, ..., T_{k_m})$ of T called a *task list*.
2. At any time t a P_i completes a task, it immediately scans the list L, starting from the beginning, until it comes to the first task T_j which has not yet begun to be executed.
3. If all the predecessors of T_j (i.e. those $T_i \prec T_j$)have been completed by time t then P_i begins working on T_j. Otherwise, P_i proceeds to the next task $T_{j'}$, in L which has not yet begun to be executed, etc.
4. If P_i proceeds through the entire list L without finding a task to execute, then P_i becomes *idle*.

5. P_i remains idle until some other P_j completes a task at which time P_i (as well as P_j) immediately scans the list L as before for possible tasks to execute.

6. If two processors P_i and P_j, $i < j$, simultaneously attempt to begin the same task T_k, T_k will be assigned to P_i, which has the smaller index.

The processors all start scanning L at time $t = 0$ and proceed in the above-mentioned fashion until some time ω, the least time for which all the tasks have been completed.

2.2.2 Impressive Example

In the coming example, we have the following conditions:

- 3 processors.
- 7 tasks.
- $\mu(T_1) = \mu(T_2) = \mu(T_3) = \mu(T_4) = \mu(T_5) = \mu(T_6) = 1$; and $\mu(T_7) = 3$.
- No constraint in $\prec$.

Situation 1

We have $L = (T_1, T_2, T_3, T_4, T_5, T_6, T_7)$.

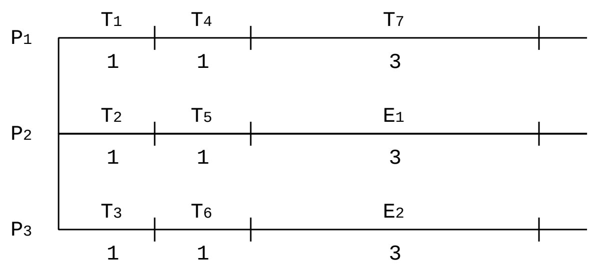

Figure 4: Load balancing situation 1.

In this case, $\omega = 5$.

Situation 2

We have $L^{'} = (T_1, T_2, T_7, T_3, T_4, T_5, T_6)$.

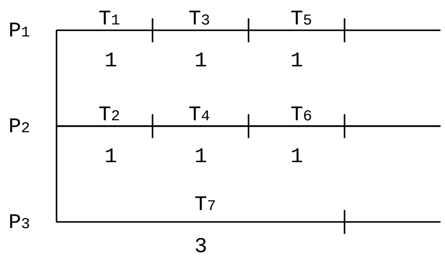

Figure 5: Load balancing situation 2.

In this case, $\omega^{'} = 3$.

With a slight change in task list L, we can improve the total time. There are other factors that seems to improve our result.

- Change the task list L.
- Decrease the function μ.
- Relax the partial-order $\prec$.
- Change the number of processors from n to $n^{'}$.

However, the results may not always be improved, it may be worse. The bound is :

$$\frac{\omega^{'}}{\omega} \leq 1 + \frac{n-1}{n^{'}}$$

That means no matter how you change the above parameter(s), the new total length $\omega^{'}$ will not be increased (or decreased) by the above bound.

2.2.3 Greedy Approach

Now, we are back to our original problem. We know that no matter how bad our design is, there is still a bound that governs the maximum time needed. Our original design uses greedy approach to locate the suitable server. In [6], the authors have already calculated the bound:

$$\frac{(3n)^{\frac{2}{3}}}{2}(1 + o(1))$$

3 Load Balancing using Mobile Agent Approach

3.1 Introduction

In figure 3, we have illustrated the design of a typical load balancing system. In our approach, the switch will gather information (the loading status) of all the servers in order to make load balancing decision. Currently, the approach is **centralized**: let the switch to collect loading status from all machines periodically, say every 5 seconds.

My approach to this decision is a **decentralized** one, in the sense that the computations of load balancing decision making are not performed on the switch, but on the back-end servers. When the switch finds a customer at the head of the queue, it will encapsulate the customer's request into an mobile agent and the agent will go to find a server that has the minimum load autonomously.

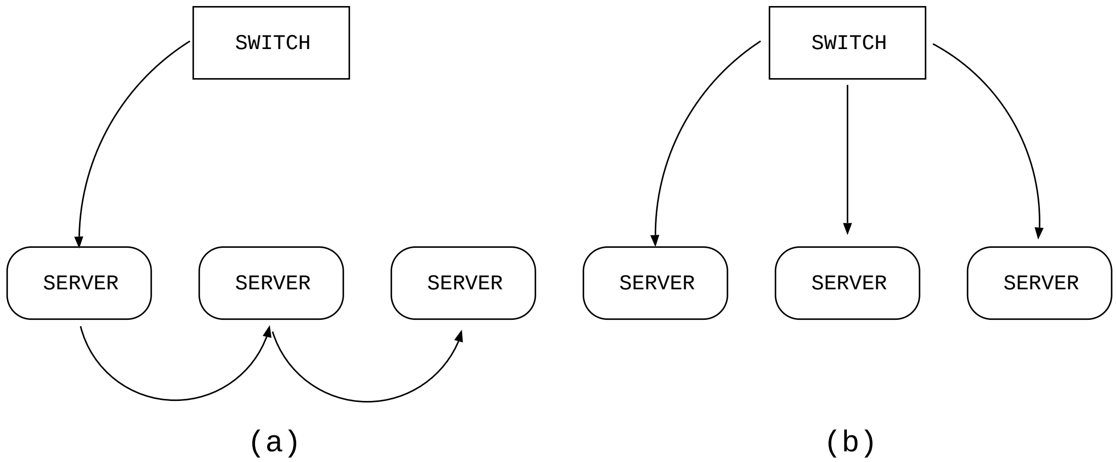

Figure 6: Load Balancing Using Mobile Agent Approach.

The above figures shows two approaches using agents to calculate load balancing decision. One is a *Multi-Hop Mobile Agent* approach. Another is a *Multi-Mobile Agent* Approach.

3.1.1 Multi-Hop Mobile Agent Approach

In this approach, only one agent will be created for a task. An agent will search for the suitable server one by one. If the agent finds a server idle or having a low-load (smaller than a threshold ϵ), the agent will choose this server. Otherwise, the agent will go through all the servers and pick the one that has the lowest load.

The advantages of this schemes:

- We distribute the load of the switch to the server, this can increase the efficiency of the switch in creating agents.

- This scheme allows the switch to create several agents. This can help to make several decision at one time.

However, our scheme will be a failure if the number of servers is too large. As we have to travel all server once, the load information will be out-of-date or incorrect at the time when decision is made if we have to visit a lot of servers. If the number of servers is small and the agents are migrating fast enough, this will be a very efficient scheme.

3.1.2 Multi-Mobile Agent Approach

In this approach, several mobile agents will be created for one task. These mobile agents will go to one server to calculate the load status. After calculation, they will go back to the switch and make a decision. However, this method has no difference to the centralized approach.

We will have the new approach combining the 2 approaches.

3.1.3 Multi-Multi-Hop Mobile Agent Approach

Since the efficiency of the multi-hop approach is restricted by the number of servers, we are now sending multiple agents in this new scheme. After all the agents finished calculating the sub-minimal, they will gather together to calculate global minimum load.

3.2 Pros and Cons Using Mobile Agent

Pros

- By using mobile agent, we can calculate the load status with respect to the *nature of the service*. For example, if the task is a memory intensive process, we have no reason to treat the CPU load as the load information. Therefore, we can plug a suitable instruction to the mobile agent in order to calculate the load status.
- Since we calculate the load status at the time that task arrives on the head of the queue, we are always making use of the latest information.

Cons

- The mobile agent approach introduces undesirable load to servers as the agents calculate the load information before the execution by incoming tasks.
- Mobile agent is typically written by script language or Java. They are much slower than currently adopted approaches.

4 Conclusion

In this report, two kinds of agents are examined: Multi-agent system and Mobile agent. Their advantages as well as disadvantages are discussed. Load Balancing is an old research topic, however, many domains are adopting this issues such as clusters of computers. In this report, the possibilities of combining mobile agent and load balancing are illustrated. This shows that with the autonomous characteristics of agents, we can improve the current load balancing schemes.

References

[1] Colin G. Harrison, David M.chess, Aaron Kershenbaum, "Mobile Agents: Are they a good idea?", IBM T.J Watson Research Center, 28 March 1995.

[2] Dajan Milojicic, "Trend Wars", IEEE Concurrency, No.3, 1999.

[3] Danny B. Lange and Mitsuru Oshima, "Seven good reasons for mobile agents", Communications of ACM, Vol. 42, No.3, 1999, Page(s): 88-89.

[4] Hyacinth S. Nwana and Divine T.Ndumu, "A Perspective on Software Agents Research", Knowledge Enginerring Review Vol. 14, No.2, Page(s): 1- 18.

[5] R.L. GRAHAM, "Bounds for Certain Multiprocessing Anomalies", The Bell System Technical Journal, November 1966, Page(s): 1563-1581.

[6] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, "On-Line Load Balancing", Foundations of Computer Science, 1992. Proceedings, 33rd Annual Symposium on, 1992, Page(s): 218-225.

[7] http://www.sun.com/jini

[8] http://www.cs.umbc.edu/kqml/

[9] http://www.cselt.it/fipa/