Computer Science Department
Institute of computer Communications
and Applications

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

8TH SEMESTER PROJECT:

INTELLIGENT AGENTS FOR NETWORK MANAGEMENT

LIONEL MICHAUD

Assistant: Jean-Philippe Martin-Flatin

Lausanne, June 26, 1998

Table of Contents

1 Introduction

The objective of this project is to compile a state of the art on cooperative platforms based on intelligent agents, and to select of one of these platforms to experiment with the management of IP multimedia networks. Even though intelligent agents have already been used in many AI-related research domains, very few of these platforms have been used in network management yet.

Agent software is a rapidly developing area of research. The word agent is very popular in the computing press as it is within the artificial intelligence and the computer science communities. At present, agents are used for more and more applications in very diverse domains. Agent-based applications have been developed for fields as varied as manufacturing, entertainment, electronic commerce, user assistance, service and business management, and information retrieval. According to BIS Strategic Decision, "*Agents will be the most important computing paradigm in the next 10 years. By the year 2000, every significant application will have some form of agent feature." [2]*

With the rapid growth of the Internet, network management is a more and more demanding domain. Furthermore, the relentless growth in the information-processing needs of organizations has been accompanied by rapid development in computer and data-networking technology to support those needs, and an explosion in the variety of equipment and networks offered by vendors. Network administrators become overwhelmed with a lot of simple and boring tasks. Most of their work could be automated and simplified by the use of intelligent software.

Two approaches are possible in order to accomplish network management with intelligent agents. The first one is to start from the network management point of view and then move towards the artificial domain, trying to use some artificial intelligence concepts. The second one involves the opposite approach, that is to say, to start from the distributed artificial intelligence perspective, investigate on what has been done there, and then try to apply this to network management.

This document will first introduce some concepts used in the scope of this project (chapter 2). The primary goal of the project was to find a cooperative platform using intelligent agents. A detailed explanation of the different platforms investigated will therefore be given, accompanied with a personal critique (chapter 3). The elected platform, JAFMAS, will then be described (chapter 4). Finally, an implemented multimedia application based on JAFMAS will be depicted (chapter 6).

2 Literature Review

2.1 Agents

There is no standard definition of the term agent on which a consensus exists. Peter Norvig defines agents as "*anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.*" [33] Pattie Maes, of the software agents research group at MIT, has coined her own definition of the term: "*An agent is a computational system that inhabits a complex, dynamic environment. The agent can sense and act on its environment, and has a set of goals or motivations that it tries to achieve through these actions.*"[26] However the term agent remains vague, so let us see some of the commonly identified agent attributes to have a better view of what an agent can be or do. [37],[15],[4]

Autonomy. An agent can operate without the direct intervention of humans, which means that it should have a degree of autonomy from its user. This requires aspects of periodic action, spontaneous execution, and initiative. That means that the agent must be able to take preemptive or independent actions that will eventually benefit the user. [14]

Cooperation. Agents can interact with other agents and/or humans. This is often best viewed as a two-way conversation, in which each party may ask questions to the other to verify that both sides are in agreement about what is going on. As such, the two parties interact more as peers in agent-oriented systems. Inter-agent cooperation is a mechanism by which agents exchange their knowledge, their beliefs and their plans to work together and solve larger problems, which are beyond their individual capabilities.[12]

Reactivity. Agents can perceive their environment and respond in a timely fashion to the changes that occur in it.

Pro-activeness. Agents can exhibit goal-directed behavior by taking the initiative. They can reason about their intentions and beliefs, and accordingly plan their course of action.

Mobility. Agents can move to other environment. They can carry data along with intelligent instructions that can be executed remotely.

Temporal Continuity. Agents are continuously running processes.

Learning. Agents continuously adapt to changes in the environment. Ideally, there should be improvement in the behavior of the agent.

The combination of these different features results in different agent types. Here are the most important types. [9]

Autonomous agents. Agents that inhabit some complex, dynamic environment, sense and act autonomously in this environment and by doing so realize a set of goals or tasks.

Entertainment agents. Interactive, simulated worlds providing entertainment to a user.

Information agents. Agents that have access to potentially many information sources and are able to collate and manipulate information obtained from these sources to answer queries posed by users and/or agents.

Interface agents. Personal assistants who are in collaboration with the user in the same environment to provide assistance. The agents observe and monitor actions taken by the user in the interface, learn from it, and suggest better ways of doing the task. [29]

Reactive agents. A special category of agents which do not possess internal symbolic models of their environments; instead they act/respond in a stimulus-response manner to the present state of the environment.

Collaborative agents. These agents emphasize autonomy and cooperation with other agents in order to perform their tasks. They may have to negotiate in order to reach mutually acceptable agreements in some matters.

Mobile agents. Mobile agents are computational software processes capable of roaming wide-area networks. Such agents move from computer to computer and interact with their hosts. This field is getting more and more important with the development of the World Wide Web.

Intelligent agents. Agents that carry out some set of operations on behalf of a user or another program with some degree of independence. Any kind of artificial intelligence method can be used to give such independence. Intelligent agents usually include one or more of the agent type described above.

Of course, these agent types can be combined to form hybrid agents. In the future, I will use the term *intelligent agents* to designate agents with cooperative, autonomous, intelligent capacities.

2.2 Network Management in the IP World

In the earliest days of computer communications, it was quite a chore to reliably move bits from a mainframe to an application terminal. As technology advanced into the late-70's, terminal networks evolved into host networks: hosts were attached to a single "packet-switched" network where they were supposed to communicate. In the mid-80's, various economic and technological factors made *internetworking* feasible. [31]

In an internet, several networks are connected together through the use of routers and an *internetworking* protocol. The routers (sometimes called gateways), using the protocol, hide the underlying details of the actual networks, in order to provide a uniform service across networks.

For example, a site-level network might consist of a local area network based on Ethernet technology. This network, along with several other nearby site-level networks, might be attached to a regional network, consisting of several routers and point-to-point connections. In turn, this regional network, along with several other regional networks, might be attached to a national backbone, consisting of another set of routers. Finally, this national network, might be connected to several other backbone networks, and have international connections.

As networks grew, standards protocols were needed to allow these networks to interconnect equipment from multiple vendors. The Simple Network Management Protocol (SNMP) [5] is an open protocol that was established in the early 90's to fulfill that need. SNMP was widely adopted by the Internet Protocol (IP) world to manage local-area networks, wide-area networks and Intranets, and to a lesser extent, to manage distributed systems. After that, new technologies suggested new approaches to network management. The Common Object Request Broker Architecture (CORBA) proved to be a viable alternative to the traditional client-server paradigm, while intelligent agents started to spread from Distributed Artificial Intelligence (DAI) to distributed systems. [27]

Large networks cannot be monitored and managed by human effort alone. The complexity of such systems dictates the use of automated network-management tools. [35] As network installations become larger, more complex, and more heterogeneous, the cost of network management rises. A 1992 survey among executives of the 1,000 largest U.S. firms, indicates the high magnitude of the cost, with over one quarter of firms spending more than $100,000 on network-management products. [16]

Network management is a very broad domain and several paradigms can be used to achieve such a goal. A simple taxonomy proposed by Martin-Flatin et al. [27] consists of four types:

- ❖ Centralized paradigms
- ❖ Weakly distributed hierarchical paradigms
- ❖ Strongly distributed hierarchical paradigms
- ❖ Strongly distributed cooperative paradigms.

The aim of this project is to experiment with the last type: strongly distributed cooperative paradigms. A perfect example of a strongly distributed software using cooperative methods to solve problems is a multiagent system, that is a group of intelligent agents.

In the scope of this project we will concentrate on managing multimedia services and networks. These systems involve configuration and delivery of user requested services at the right time, cost and quality of service (QoS). A typical example is the management of multimedia application doing some videoconferencing through the Internet. A user would be able to select a certain quality of service and to reserve it for a certain amount of time. A user agent would translate that quality of service into bandwidth and interact with router agents in order to reserve a path through the network. More detailed explanation would be given in chapter 5.

2.3 JAVA

This section describes in detail various reasons why Java renders itself as a suitable choice to build MAS framework [13],[19],[28].

2.3.1 Architecture Neutral and Portable

Agents are inherently distributed. Thus, applications must be able to execute anywhere on the network without prior knowledge of the target hardware and software platform. Java provides the advantages of architecture neutrality and portability to agent developers.

The solution that the Java system adopts to solve the binary-distribution problem is a “binary code format” that is independent of hardware architectures, operating system interfaces, and window systems. If the Java run-time platform is made available for a given hardware and software environment, an application written in Java can then execute in that environment without the need to perform any special porting work for that application. Instead of machine code, the Java compiler generates bytecodes: a high-level, machine-independent code for an hypothetical machine that is implemented by the Java interpreter and run-time system.

The primary benefit of the interpreted byte code approach is that compiled Java language programs are portable to any system on which the Java interpreter and run-time system have been implemented. The architecture-neutral aspect is one major step towards achieving portability, but there is more to it than that. C and C++ both suffer from the defect of designating many fundamental data types as “implementation dependent”. Java eliminates this issue by defining standard behavior that will apply to the data types across all platforms. Java also specifies the size of all its primitive data types and the behavior of arithmetic on them.

2.3.2 Multithreaded

As the human world is full of multiple events all happening at the same time, the agent world should work the same way. Built-in support for threads is one of the most powerful tools in Java, not only to improve interactive performance of graphical applications, but also to run multiple events concurrently. Multithreading is the way to obtain fast, lightweight concurrency within a single process space. The Java library provides a Thread class that supports a rich collection of methods to start a thread, run a thread, stop a thread, and check on the status of a thread. Java thread support includes a sophisticated set of synchronization primitives based on the widely used *monitor and condition variable* paradigm.

2.3.3 Distributed

In any MAS, agents residing on different machines or different environments need to communicate information and knowledge about their goals, beliefs and intentions to each other in order to coordinate and cooperate so as to bring about a coherent solution. Thus communication is a very important aspect in the development of any MAS. Java especially lends itself as an extremely suitable choice in this regard. It offers an extensive library of classes and routines which cope easily with both UDP and TCP/IP protocols, and thus supports sending both broadcast and directed messages across the network. Moreover, it provides the feature of RMI (Remote Method Invocation) which is extremely helpful while creating a family of collaborating agents.

RMI enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. It is possible to generate mobile code using RMI as it is possible to transport objects between client and server. A Java program can make a call on a remote object once it obtains a reference to the remote object, either by looking up the remote object in the bootstrap naming service provided by RMI, or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can also be a client of other remote objects. RMI uses the Object Serialization feature of Java to marshal and unmarshal parameters. It does not truncate types and supports true object-oriented polymorphism. Methods invoked on remote server objects by clients are reachable through the TCP/IP protocol.

Java also supports broadcasting. It provides a multicast datagram socket class which is useful for sending and receiving IP multicast packets. A MulticastSocket is a UDP socket, with capabilities for joining groups of other multicast hosts on the Internet. This is an extremely useful feature for implementing a MAS. The ability to send broadcast messages helps in creating a truly scalable and flexible agent framework because the agents do not need to register with a centralized directory service to be able to receive messages from one another when they come on line. Also as the agents do not need to know the identity of the receiver agents in order to be able to send messages to them, there is no necessity for any start-up protocol. It also saves network bandwidth when the same message has to be sent to all the agents in a group.

2.3.4 Secure

Java is intended for use in distributed environments. Toward this end, a great deal of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems. Java is a strongly typed language and it does not support pointers, which make it a very robust language. The Java run-time system uses a bytecode verification process to ensure that code loaded over the network does not violate any Java language restrictions. The authentication techniques are based on public-key encryption.

2.3.5 Object-Oriented

Object-oriented design is a very powerful concept because it facilitates the clean definition of interfaces and makes it possible to provide reusable software. As a programmer, this means that you focus on the data in your application and methods that manipulate that data, rather than thinking strictly in terms of procedures. Java adopts the four principles of object oriented languages: inheritance, encapsulation, abstraction and message passing communication.

Though there are many languages which are object oriented in nature, Java is one of the few languages which enforces it. The user has no choice but to encapsulate all data in objects. Since agents are essentially built around a group of different object components, Java is an ideal language for developing them.

2.3.6 Database Connectivity - JDBC

The Java Database Connectivity kit lets Java programmers connect to any relational database, query it, or update it using the industry standard query language (SQL). This is a very useful feature as databases are among the most common uses of software and hardware today, and any application using Java can easily integrate with preexisting databases to update the local model of its agents. This is even more useful for MAS as agents usually possess a knowledge base. As this knowledge base gets bigger, it has to be organized and stored efficiently. A database connection can be very useful to store all this information.

3 Investigation of Existing Platforms

In order to perform network management with intelligent agents a multiagent platform was needed. Since so many people are presently working on such systems, a thorough search of what had been done and what was freely available had to be done. The World-Wide Web was the main tool for this research, along with the proceedings from ICMAS'95 and ICMAS'96.

The first goal was to find a platform including multiagent features and some kind of intelligence. That means the platform had to provide not only the features permitting to set up a distributed multiagent application, but also, the agents had to possess some reasoning abilities. Such a platform would have been ideal since the only thing that would have been left would have been to customize the agents to respond to the particular needs of the application.

It turned out that such an intelligent platform was nowhere to be found. So the approach adopted was to find a multiagent platform on which some intelligence could be plugged in. It was important to keep in mind that the platform had to be adaptive and flexible enough to allow intelligent modules to be plugged in without too much trouble. Lots of people are working on multiagent systems and therefore a lot of prototype exists out there. However, very few of them were satisfactory.

Section 3.1 describes multiagent systems. Sections 3.2 to 3.10 present the most relevant platforms investigated, in order of decreasing interest. The platform adopted for the application: JAFMAS will be detailed in chapter 4.

3.1 Multiagent Systems

3.1.1 Definition

Multi-Agent Systems (MAS's) are an outgrowth of the Distributed Artificial Intelligence community. Durfee et al., define a multiagent system as *"a loosely-coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities.*" [3] These problem solvers, which are essentially autonomous, distributed, and maybe heterogeneous in nature, are called agents. Research in multiagent systems is mainly concerned with how they coordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems. The concepts that are important and have to be taken into consideration in studying a multiagent system are the following:

Communication: What protocol does the system use? Is it flexible? Does the system provide directed communication and/or multicast communication?

Programming Language: Is it a standard language? Easy to use? Compatible with other components? Portable?

Flexibility: Is it easy to adjust the system to a particular application? What are the constraints and requirements?

Architecture: Is the system object oriented? Layer based? Well designed?

User Interface: Can the agents be visualized? How does the user interact with the system?

Scalability: Does the system adapt itself to different situations? Can the system be widely extended?

User friendliness: Is it easy to start using the system? Is the learning curve low?

Identification: How do agents identify one another? Is it a centralized way or through communication?

Security: Are they any security features provided? Are the communications among agents encrypted?

Extra features: Are any extra-features available? Are the agents mobile? Are any coordination constructs available?

3.1.2 Communication

Communication enables the agents to exchange information on the basis of which they will coordinate and cooperate with each other. In a multiagent system, several ways have been proposed for agents to exchange information with each other. Agents can directly exchange messages, or they can organize themselves into a federated system and communicate through special facilitator agents, or they can broadcast messages. Another possibility is to use a shared data repository in which information can be posted and retrieved.

In order to address the many difficulties of communication between agents, a first language called Knowledge Interchange Format (KIF) has been created to represent knowledge. KIF is a computer-oriented language for the interchange of knowledge among disparate programs. It has a declarative semantic (i.e. the meaning of expressions in the representation can be understood without appeal to an interpreter for manipulating those expressions); it is logically comprehensive (i.e. it provides for the expression of arbitrary sentences in the first-order predicate calculus); it provides for the representation of knowledge about the representation of knowledge; it provides for the representation of nonmonotonic reasoning rules; and it provides for the definition of objects, functions, and relations. [18]

The ARPA Knowledge Sharing Effort (KSE) designed another common language called KQML. [7] KQML, now adopted by most researchers in the AI domain [11], stands for the

Knowledge Query and Manipulation Language. KQML is an agent communication language that is used in systems ranging from research experimental systems to real business production systems. The advantage of KQML is that is does not standardize on any representation language. In KQML there is not really a shared semantic but a shared ontology. When using KQML, a software agent transmits messages composed in its own representation language, wrapped in a KQML message.

3.1.3 Extra Features

Two other factors were also taken into account throughout the platform research. First the documentation available with the software: a reasonably detailed and understandable documentation was required in order to adopt the platform. Second the availability of the platform: with a null budget, the platform had to be freely available.

The following chapters describe the most interesting platforms found during the investigation, starting with the one using Java. The platform chosen, JAFMAS, will be described in greater detail in chapter 4.

3.2 JATLite

http://java.stanford.edu/java_agent/html/

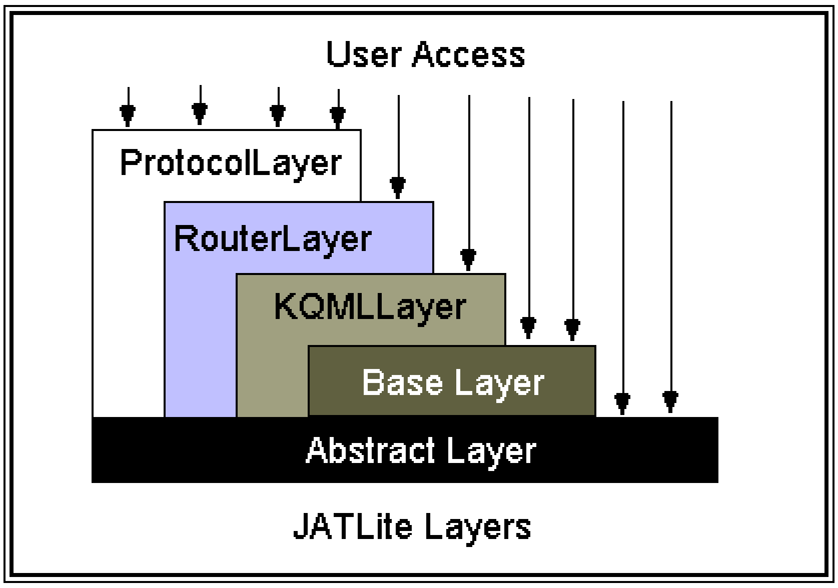


Figure 3.2-1 *JATLite is built as a series of increasingly specialized layers*

JATLite is being developed by the Computer Science Department at Stanford University. It provides a set of Java packages that facilitates agent framework development using the Java language. JATLite provides basic communication tools and templates for developing agents that exchange KQML messages through TCP/IP. It defines a special construct called an Agent Name Server (ANS) which stores all the names and addresses of existing agents. When an agent is created and connected to the network, it first registers with the ANS. In registering, the agent passes the ANS its name, port number and the domain of its local host. If an agent knowingly terminates, it first sends a remove address message to the ANS, which echoes the message to all the other agents. JATLite also provides special Agent Router functionality which allows Java applets to exchange messages with any registered agent on the Internet. The Agent Router allows any registered agent to send messages to any other registered agent by making a single socket connection to the Agent Router. Messages are forwarded without the sending agent having to know the receiving agents' address and making a separate socket connection with the Agent Name Server (ANS) infrastructure.

Although JATLite does provide essential functionality required for building a multiagent application, it does not define a methodology for specifying the social behavior of agents. Moreover, the concepts of the ANS and the Agent Router are inherently centralized in nature. Each time an agent joins the system, it has to register with ANS; and when it leaves the system, then the ANS also has to be informed. All communication must go through the Agent Router. Thus, any application developed using JATLite cannot be truly scalable. [24]

3.3 Aglets

http://www.trl.ibm.co.jp/aglets/

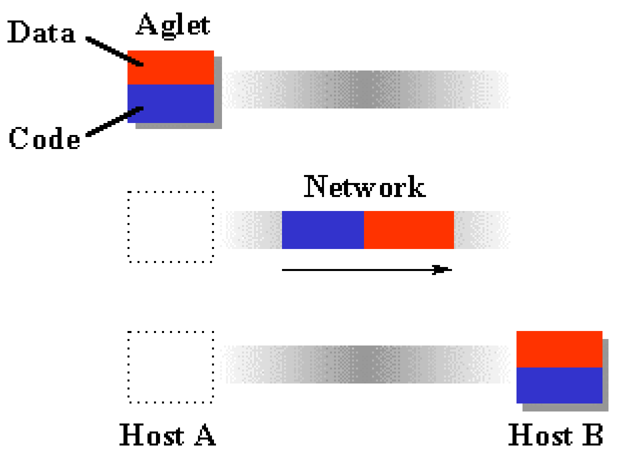


Figure 3.3-1 *Aglets Serialization through the network*

IBM Japan is developing a framework for Java "Agent Applets". Some of their work has been submitted to the Object Management Group (OMG) for consideration in regard to OMG's request for a Mobile Agent Facility (MAF). Aglets Workbench is a visual environment for building network-based applications that use mobile agents to search for, access and manage corporate data and other information. IBM Aglets are mobile Java programs which may travel and execute in specialized nodes in the network. The Java Aglet

Application Programming Interface of the framework defines the methods necessary for Aglet creation, message handling in the network and initialization, dispatching, retraction, deactivation/activation, cloning and disposing of the Aglet. The Aglets workbench includes an Agent Web Launcher named Fiji and a Visual Agent Manager named Tahiti. Fiji is a Java applet based on the Aglets Framework and therefore capable of creating an Aglet and retracting an existing Aglet into a client's web browser. Tahiti uses a unique graphical user interface to monitor and control Aglets executing on a given computer. It also implements a configurable security manager that provides a fairly high degree of security for the hosting computer system and its owner. Although, Aglet is more intended to allow agents to move than a framework for multiagents, it can be combined with JKQML to allow communication among agents.

Based on KQML, JKQML has been developed by IBM to provide a framework and API for constructing Java-based, KQML-speaking software agents that communicate over the Internet. JKQML allows the exchange of information and services between software systems, creating loosely coupled distributed systems. JKQML provides flexibility for the extension of the framework, and it supports the following three protocols: [25]

- KTP (KQML transfer protocol): a socket-based transport protocol for a KQML message represented in ASCII.
- ATP (Agent Transfer Protocol): a protocol for KQML messages transferred by a mobile agent that is implemented by Aglets.
- OTP (Object Transfer Protocol): a transfer protocol for Java objects that are contained in a KQML message.

JKQML is based on the 1997 proposal for a new KQML specification.

Aglets Workbench is a very versatile tool for creating secure mobile agent-based applications, however it does not deal with the important issue of implementing coordination, cooperation and coherence in agent-based applications. IBM Aglets can only engage in directed communication as they use the TCP/IP protocol. However, including the new features of JKQML, Aglets could be the most adequate tool in case some mobility is needed inside the agent community. Another advantage of Aglets is that it is quite simple to use the API, it goes pretty fast to get something done, and a very nice user-interface is provide to control the agents. More investigation would be needed to check if what is behind the API is as good as the Aglet user interface. [20]

3.4 Concordia

http://www.meitca.com/HSL/Projects/Concordia/

Mitsubishi Electric Information Technology Center of America has developed a Java-based framework for development and management of network-efficient mobile agent applications for accessing information anytime, anywhere, and on any device. Concordia offers a flexible scheme for dynamic invocation of arbitrary method entry points within a common agent application. It provides support for agent persistence and recovery and guarantees the transmission of agents across a network. Concordia has also been designed to provide for fairly complete security coverage from the outset. Within Concordia, an agent's travel plans are specified by its Itinerary. The Itinerary is a completely separate data structure from the agent itself. Concordia provides two forms of asynchronous distributed events: selected events and group-oriented events. The event selection paradigm enables agents to define the types of events they wish to receive. In contrast, group-oriented events are distributed to a collection of agents (known as an event group) without any selection.

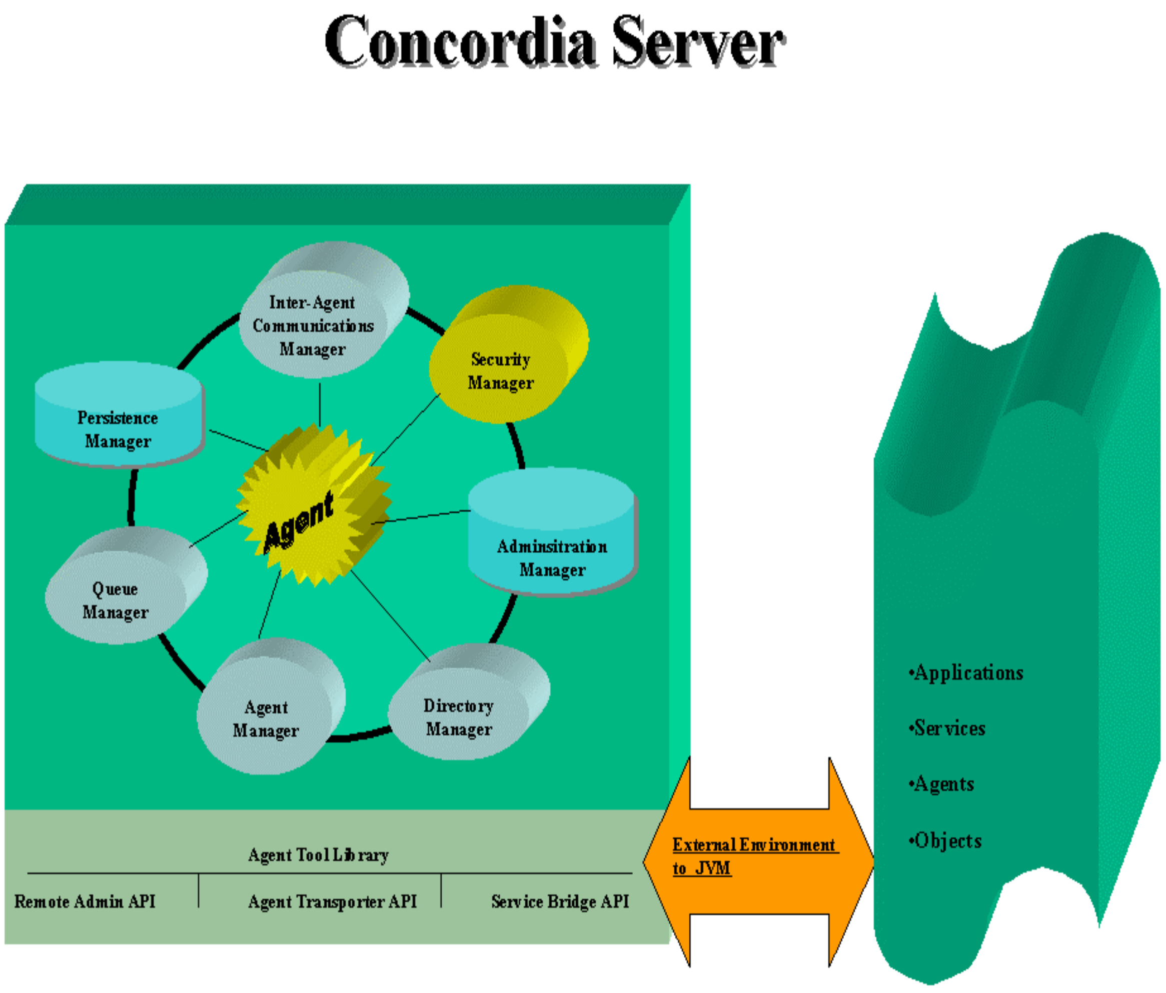


Figure 3.4-1 *Concordia server architecture*

Although Concordia provides a useful set of services for implementing agent mobility, security, persistence and transmission, it does not provide any methodology to specify how agents in a multiagent system coordinate, cooperate and negotiate to bring about a coherent solution. Emphasis here is on the communication aspect in an agent-based application. Moreover, the fact that the agent itinerary is outside the agent implies that where the agent travels is maintained in a separate logical location regarding the place where the agent lives. This results in Concordia agents not being totally autonomous. [6]

3.5 Odyssey

http://www.genmagic.com/technology/odyssey.html

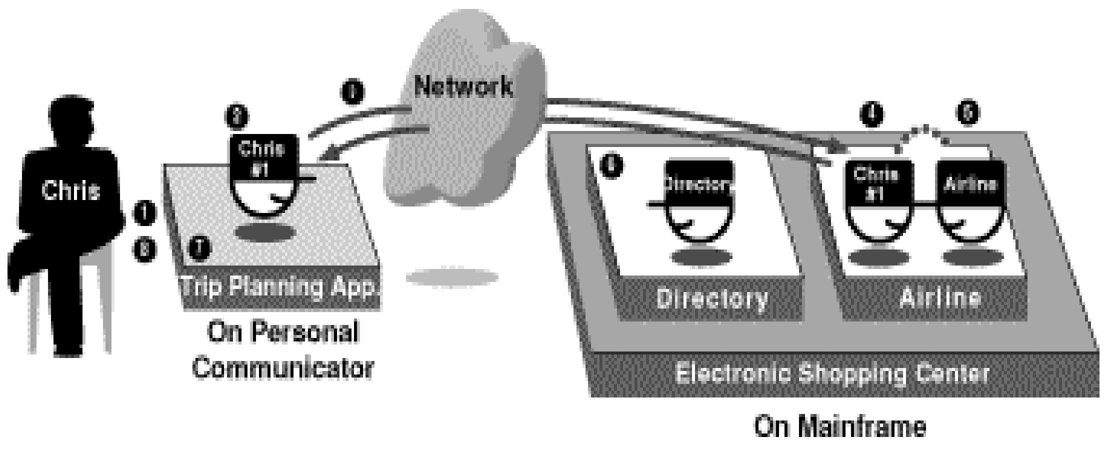


Figure 3.5-1 *Typical Odyssey Application*

Odyssey is General Magic's implementation of mobile agents in Java. It is an outgrowth of Telescript, the first mobile agent commercially available. Odyssey is an agent system implemented as a set of Java class libraries that provide support for developing distributed mobile applications. Odyssey technology implements the concepts of places and agents. It models a network of computers, however large, as a collection of places. A place offers a service to the mobile agents that enter it. A communicating application is modeled as a collection of agents. Each agent occupies a particular place. However, an agent can move from one place to another, thus occupying different places at different times. Agents are independent in that their procedures are performed concurrently. Odyssey provides Java classes for mobile agents and stationary places.

Odyssey is General Magic's flagship to enter the arena of Java-based mobile agent computing. The current version of Odyssey provides the basic functionality of mobile agents but it does not go far beyond. The developer must adhere to a very rigid structure while implementing mobile applications. Unlike IBM Aglets and Concordia, it does not provide an extensive security mechanism. Odyssey does not support broadcast communication and speech-act messaging. Presently on the downhill slope, Odyssey has been passed by its concurrent mentioned above. Nonetheless, the director of IBM's Aglets program has recently joined the General Magic's Odyssey team, so there might be a future convergence between the concepts of these two tools. A new platform is promised for this summer. [30]

3.6 Voyager

http://www.objectspace.com/voyager/index.html

Voyager is a Java-based agent-enhanced Object Request Broker (ORB) developed by ObjectSpace Inc. It allows Java programmers to quickly and easily create sophisticated network applications using both traditional and agent-enhanced distributed programming techniques. It provides for creation of both autonomous mobile agents and objects. Voyager agents roam a network and continue to execute as they move. Voyager can remotely construct and communicate with any Java class, even third party libraries, without source. It allows seamless support for object mobility. Once created, any serializable object can be moved to a new location, even while the object is receiving messages. Messages sent to the old location are automatically forwarded to the new location.

Voyager is a very efficient tool for constructing agent-based distributed applications. However, it does not provide any classes for defining the social behavior of agents, does not support broadcast communication and speech-act messaging, and does not pay any specific attention to providing security. It also emphasizes on the mobility aspect, feature not required in the scope of this project. [36]

The MAS's brought up until here are the most famous in the agent world. Although they provide some extensive features, most of them emphasize on mobility and do not take enough care of the social behavior of the agents. The following MAS's will be more concerned by the agents and their interaction with other agents rather than the mobile aspect.

3.7 IA Factory

http://www.bitpix.com

Bits and Pixels is an American company using intelligent agents for various applications. Tired of starting implementing their agents from scratch, they developed what they call the "IA Factory". The goal of the "IA Factory" is to supply the programmer with an API so that he does not have to go through the entire network programming and debugging. This framework gives you a generic agent which one can extend to give the agent its specific behavior. In the simplest case, a table of behavior is sufficient. As the agents get more complicated, a class can be extended to give the agent complex and interesting behavior.

The "IA Factory" comprises five packages:

bitpix.agent :	- Full client-server agents compatible with sockets - Extensible command language processor for agents - Compatibility with KQML - Rule processor compatible with KIF

bitpix.list :	- Simple tree-based data format - String parsing facilities - Compatibility with KQML
bitpix.think :	- Logical deduction engine - Back-propagation neural network - Classification neural networks - Suitable for hybrid AI systems
bitpix.move :	- Networks, clients, servers, communicating processes - Autonomous bots - Intelligent Agent simulations - Modular filter processes - Animation classes - Models for KQML exchanges
bitpix.draw :	- Structured graphics objects - Shaded graphics classes - IFS graphics for textures and natural objects - 2D transform geometry - 3D transform geometry

All this really looks good except you have to pay to get the full version. The complete commercial release cost about $1500. That's the only way to get the Java source code for all the packages described above. Some source codes, as the AI factory, are freely available. So if one wants to see how agents are generated, one can. However, if one wants to go into the code generated and customize the agents to get them accomplish some complex behavior, one will get in trouble since there is no way to get what's behind the API. In other words, that means that one will see that to create an Agent you have to write:

Agent myAgent = new Agent()

But the Agent source code is not available.

Customization possibility is given through a file that can be passed as parameter to the agent constructor. That file needs to describe a table of behavior according to a given syntax. Even though this opportunity is interesting, the possibilities stay very limited.

A more extended possibility is to write CLIPS code. CLIPS code is what gives the intelligence to the agents. A set of rules can be defined and a forward chaining system will interpret them. The Java Expert System Shell (JESS) is used as the forward chaining system. JESS supports the development of rule-based expert systems, which can be tightly coupled to code written in the Java language. This is a very interesting possibility but it does not suppress the necessity to change the Java code generated in order to implement a consistent application.

The IA Factory is really interesting since it is the only platform to provide additional intelligent feature to the multiagent infrastructure. Although, part of the source code is not available, the "Intelligent Agent Factory" stay, however, very useful for little agent applications. The idea to integrate an interpreter for a rule language is very good. JESS can be integrated in any Java application to give it the ability to “reason”. This idea will be examined in more details in section 4.5.

3.8 RETSINA

http://www.cs.cmu.edu/~softagents/retsina/

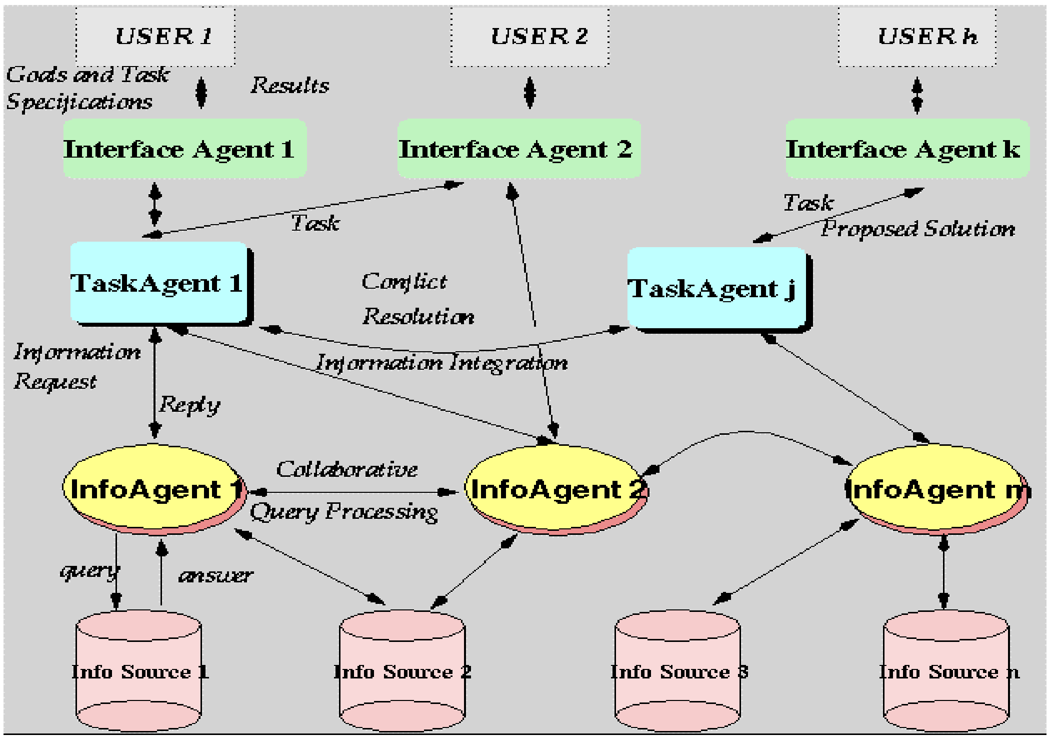


Figure 3.8-1 *Agent organization*

RETSINA, developed at Carnegie Mellon University, stands for Reusable Environment for Task Structured Intelligent Network Agents. The RETSINA framework is being used to develop distributed collections of intelligent software agents that cooperate asynchronously to perform goal-directed information retrieval and information integration in support of performing a variety of decision-making tasks. A collection of RETSINA agents forms an open society of reusable agents that self-organize and cooperate in response to task requirements. Their designer focused on three crucial characteristics of the overall framework that differentiate RETSINA from others:

- Use of a multi-agent system where the agents operate asynchronously and collaborate with each other and their user(s)
- Agents actively seek out information
- Information gathering is seamlessly integrated with problem solving and decision support

A Middle Agent Software has been developed to face the connection problem. The connection problem is to find the other agents that might have the information and capabilities that you need. Agent Name Server and Matchmaker can be created so that agents can communicate with each other. The RETSINA Agent Name Server is a set of Java programs that allows your software agents to communicate over the Internet.

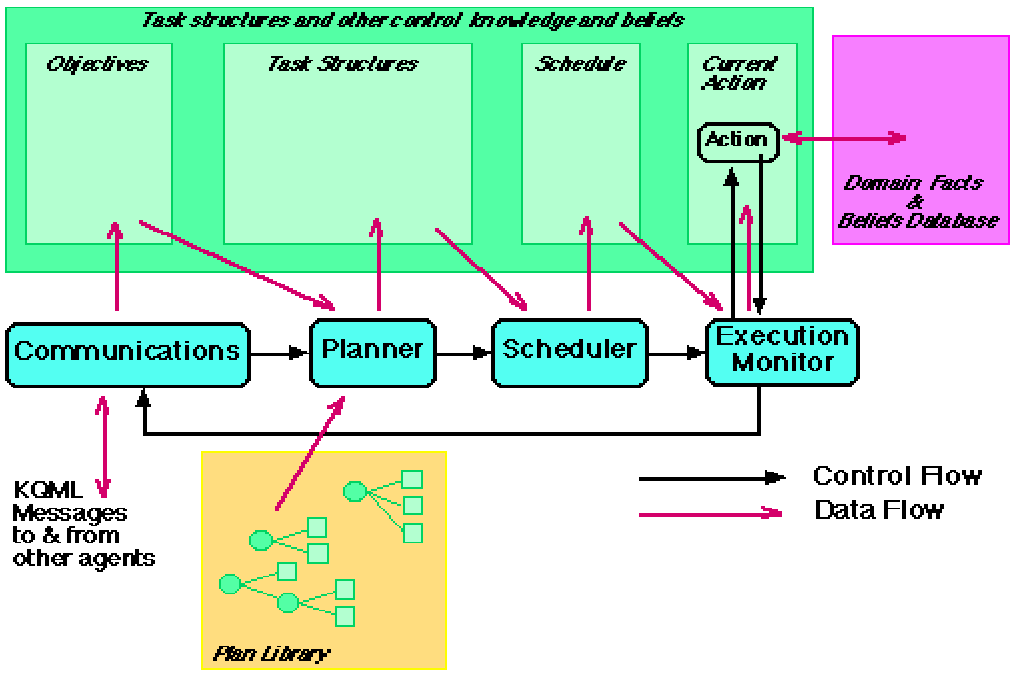


Figure 3.8-2 *Agent Architecture: Functional view*

However it seems that only some particular issues are addressed in the software available. The set of Java classes provided emphasized on how agents can find information or advertise their capabilities. No particular attention is given regarding the behavior of each agent and how they interact with other agents. The Name Server API provided use a centralized approach, making the system less scalable and fault-tolerant. As RETSINA is an open system, any agent on the Internet can communicate and interact with the actual community.

Even though some new concepts are used in the way agents cooperate with each other and dynamically arrange themselves in community it seems like RETSINA is not the adequate platform for the need of network management.

3.9 MAST

http://www.gsi.dit.upm.es/~mast/

The Multi-Agent System Tool (MAST) has been developed by the Intelligent Systems Group of the Department of Telemetric Systems Engineering at the Technical University of Madrid for the Esprit-9119 project MIX. The MAST architecture has been conceived as a general purpose distributed framework for the cooperation of multiple heterogeneous agents. This architecture has been redesigned and implemented within a research project investigating a particular class of hybrid systems: those integrated by connectionist and symbolic components.

MAST offers a decentralized model of control, uses the mechanisms of message passing for communication purposes and is implemented from an object-oriented perspective. Moreover, some features are introduced that make this platform especially suitable for symbolic/connectionist hybridization.

The MAST architecture consists of two basic entities: the agents and the network through which they interact.

3.9.1 MAST AGENTS

From an external perspective, an agent is structured as a set of elements:

- Services: functionality offered to other agents.
- Goals: self-imposed tasks (functions that an agent carries out in self-interest, not to meet a demand from another agent).
- Resources: information on external resources (services, ontologies, groups, etc.)
- Internal objects: data structures shared by all the processes that are launched by the agent to carry out service requests or to achieve goals.
- Control: specification of how service requests are handled by the agent.

Several communication primitives are implemented, including different synchronization mechanisms (synchronous, asynchronous and deferred communications) and higher level protocols, such as Contract Net.

3.9.2 NETWORK AGENTS

At the network level, a yellow-page service is offered by a specialized agent, the Yellow_Pages (YP) agent. At birth, agents register with a particular YP agent, giving their net address and information on the services they offer, and the services they might need. Agents can also subscribe to groups. Groups refer to dynamic sets of agents, and can be used as aliases in service petitions. Thus, these petitions can be addressed to an individual agent, to the agents subscribed to a group, or to all the agents offering the service. The YP agent responds to a registration request providing all the information that an agent needs to know (e.g., the addresses of the agents offering services that it will request). Such information is updated continuously by the YP agent.

A YP agent acts as a sort of active repository, not as a router. It registers and diffuses information regarding the structure of a set of agents who cooperate in a particular application. Therefore, different YP agents can be simultaneously active, even in the same machine (provided they use distinct ports for communication). By using the information received from the YP agent, the remaining application agents can establish direct communication links, thus making the risk of network collapse (due to saturation of the communication channels in the YP agent) negligible.

3.9.3 KNOWLEDGE INTERCHANGE

One important problem regarding the exchange of messages in a set of agents is how to facilitate the mutual understanding of the content of these messages. The solution adopted by MAST permits the inclusion of a parameter in message headers expressing the language used to codify its content. Moreover, the content can make reference to concepts in an ontology shared by the sender and the recipient agents alike. The current implementation includes tools for the automatic translation of messages written in a reduced version of CKRL (Common Knowledge Representation Language) both to and from C++ code. CKRL was designed by the MLT consortium (project ESPRIT-2154) as an interchange language for symbolic machine learning algorithms.

On the other hand, it is always possible to compose the content of messages in a free format. But, in this case, sender and recipient need to agree previously on the structure of the body of the messages that they interchange.

Also a specialized language (Agent Description Language, MAST-ADL) has been designed to simplify the specification of the agents cooperating in solving a problem in a hybrid framework. Finally, some tools have been implemented to translate the MAST-ADL description files to standard C++ programs.

The MAST architecture is very interesting and it is probably one of the most complete multiagent system tools. The fact that it is implemented in C++ makes it, however, less portable and multifunctional than Java-based frameworks. Most of what MAST services provide (interoperability between heterogeneous agents) are points that do not even need to be addressed in Java as it is included in the language. Also, MAST is composed of a multitude of C++ libraries and it does not seem very user friendly to a profane.

3.10 dMARS

http://www.aaii.oz.au/proj/dMARS-prod-brief.html

dMARS is an agent-oriented development and implementation environment designed for building complex, distributed, time-critical systems. It is developed by the Australian Artificial Intelligence Institute (AAII). It is intended for rapid configuration and ease of integration, and it helps with system design, maintenance, and reengineering. dMARS agents are designed according to the BDI (Beliefs, Desires, and Intentions) model. They are able to reason about their environment, their beliefs, their goals, and their intentions. They model their expertise as a set of context-sensitive plans. These plans can both react to changes in the environment and proactively pursue the agent's objectives. Using dMARS, multi-agent systems can be implemented as lightweight processes within a single UNIX process, as separate UNIX processes on the same machine, or as a distributed configuration communicating over a TCP/IP network. Interfacing with other processes is achieved via a simple, well-defined communication protocol. The system provides comprehensive libraries and components to support the development, implementation and testing of an application, therefore minimizing the need to develop application-specific support code.

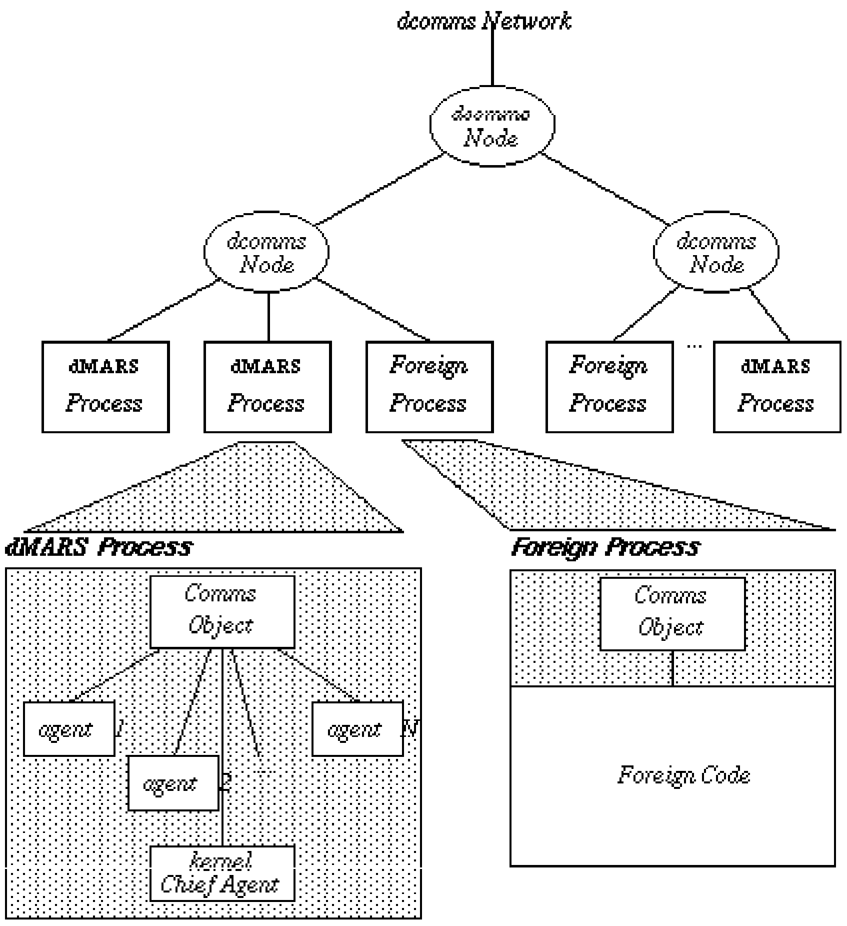


Figure 3.10-1 *Interaction of dMARS processes with foreign processes*

dMARS is written in C/C++ and thus does not provide for true architecture neutrality and portability. Unlike Java-based systems, applications developed in dMARS can only run on limited platforms and require using different compilers for different platforms. It supports only a limited number of C++ compilers. [10]

3.11 Miscellaneous

A lot of other platforms exist out there. The above list does not attempt to be exhaustive. It is also possible that other platforms we did not come across are much better than the ones depicted above. Some other platforms include: Cybele Agent Infrastructure, AgentTalk, AgentTcl, Telescript, Swarm, Echelon, COOL, and InteRRap. Those platforms have not been investigated thoroughly and have been rejected either because they are commercial products and the code is not available, or because they are implemented in a non portable language (i.e. not Java), or simply because they did not fit the requirements needed for this particular kind of application.

4 JAFMAS

JAFMAS stand for Java-based Agent Framework for MultiAgent Systems. Deepika Chauhan has developed JAFMAS during his thesis at the University of Cincinnati. [9] As mentioned above, JAFMAS is the platform elected for implementing network management via a multiagent system. JESS is the inference engine that has been chosen to give our agents some intelligence. JESS will be described in section 4.5.

4.1 JAFMAS Description

http://www.ececs.uc.edu/~abaker/JAFMAS/

Though numerous agent-building tools are currently available, most of these tools suffer from the drawback of being essentially communication centered. They do not define the social behavior of the agents in a community of agents. Java-based Agent Framework for Multi-Agent Systems, as the name suggests, is a Java-based framework for representing and developing cooperation knowledge and protocols in a multiagent system. The framework enables the agents to work together and coherently achieve their goals and those of the multiagent community as a whole. JAFMAS defines a generic methodology for multiagent application development and provides a set of services that relieves the developer from the effort of programming cooperation mechanism form scratch. It guarantees that essential interoperation, communication and cooperation facilities are available to support agent application developers. JAFMAS is concerned with coordinating intelligent behavior among a collection of intelligent agents forming the multiagent system. Agents should coordinate their knowledge, plans and goals to that they can take actions which results in a joint coherence solution to the problem at hand.

JAFMAS addresses special attention to the following issues:

1. Communication protocol
2. Interaction between agents
3. Coherence and Coordination in the multiagent community

4.1.1 Communication

As seen in section 3.1.2, communication enables the agents in a multiagent system to exchange information on the basis of which they coordinate their actions and cooperate with each other. JAFMAS support both directed and broadcast communication. It also enables the user to combine these two approaches to form federated systems. The directed communication mechanism is implemented via the RMI mechanism.

The broadcast mechanism is very useful in situations where the output of one agent is the input to many other agents or the agent does not know the identity and addresses of other agents in the system. Also, if the length of the message is substantial, and there are a large number of agents in the system, broadcast communication saves network bandwidth, as multiple copies of the same message do not need to be made. This mode of communication also enables the implementation of subject-based addressing. Each agent can subscribe to a group and any messages intended for the subject is sent only to the agent subscribing to that particular group.

On the other hand, directed communication is useful when an agent is engaged in a dialog with a particular agent, and knows exactly whom to send the message to. Broadcast messaging gives an agent a way to request for special services or to advertise his presence. Based upon this information, the agent can thereafter engage in a directed communication with the agent in concerned.

As JAFMAS uses Java's RMI mechanism for directed communication, the agents can transport objects or even agents across the network. That means mobility would be possible even if the framework does not explicitly support it.

4.1.2 Interaction

Interaction means a type of collective action wherein one agent takes an action or makes a decision that has been influenced by the presence or knowledge of another agent. The inherently heterogeneous and distributed nature of a multiagent system makes the implementation of interaction mechanism among agents a difficult process.

JAFMAS support speech-act based messaging, which means agents can communicate with their peers by exchanging messages and interact together through explicit linguistic actions. The message structure of the framework enables the JAFMAS agents to express their beliefs and their intent using an agent-independent semantics. Each message can be associated with a "performative", which is the speech-act component of the message. The "performative" determines what one can "do" or "perform" with the content of the message. Agents can thus convey belief, knowledge, or intention. The content of a message can be as simple as an affirmative or negative answer, or it can be as detailed as transmitting a complete object.

4.1.3 Coherence and Coordination

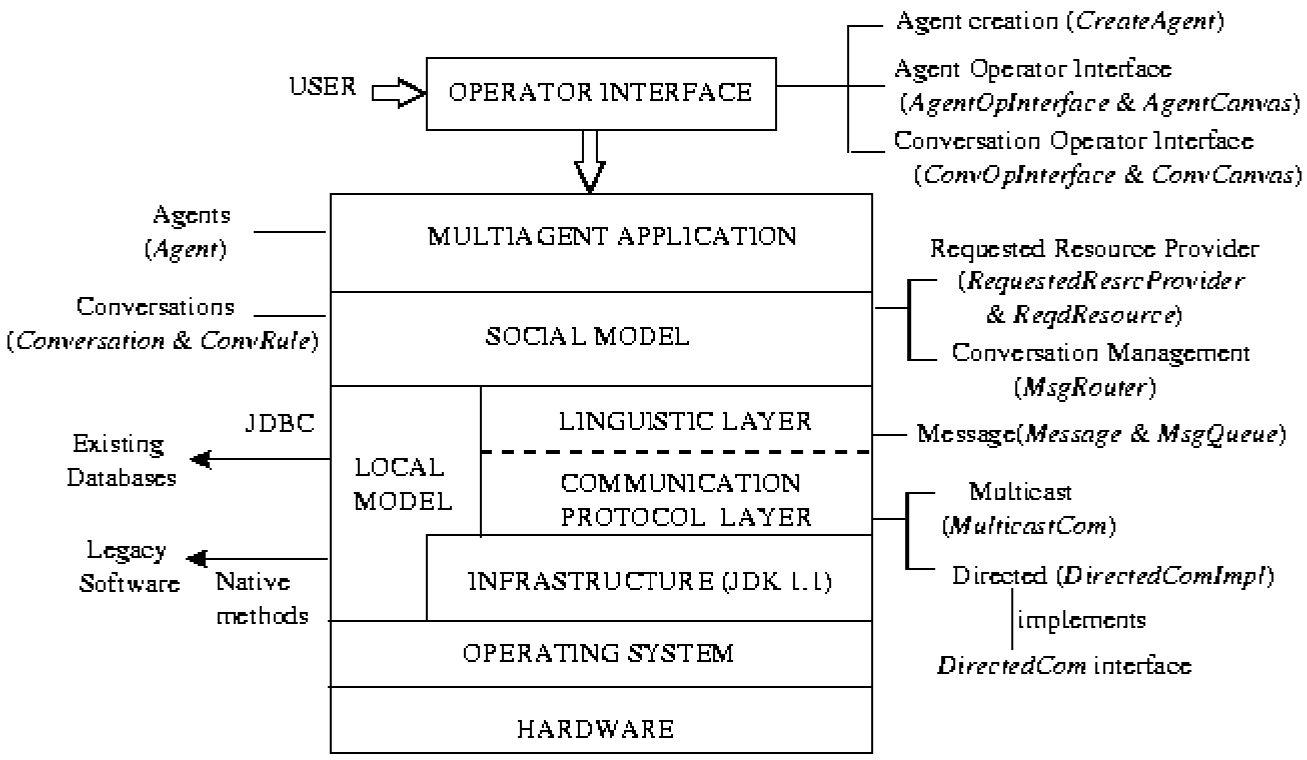
Coherence refers to how well the entire system, as a whole, behaves while solving a problem. Coordination is more the property of interaction among a set of agents performing some collective action.

JAFMAS adopts the view that the coordination problem can be solved by having knowledge about the interaction processes taking place among the agents. JAFMAS defines conversations. A conversation is an agent's plan to achieve some goal, based on interactions with other agents. Agent conversations can be visualized as automata models. As such, conversations contain alternative courses of execution based on the expected actions of other agents in the organizations. Conversation execution begins in an initial state and terminates in a final state. Within a conversation, agents can exchange messages according to mutually agreed conventions, change state and perform local actions. Each conversation has its own thread of execution, therefore agents can engage in multiple conversation at the same time.

In order to implement a MAS that gives a coherent solution, JAFMAS allows the analysis of agent conversation models for logical consistency and system coherency. As conversation can accept an automata representation, effective mathematical models can be used for conversation representation. Petri nets are used in JAFMAS to analyze coherency and coordination in agent conversations. [8] Petri nets have emerged as a very promising performance modeling tool for systems that exhibit concurrency, synchronization and randomness.

4.2 JAFMAS Architecture

Implemented in JDK 1.1, JAFMAS provides sixteen main Java classes as shown in Figure 4.2-1 (name of classes between parenthesis). Those classes provide the essential communication, interaction and coordination mechanisms to application developers by dividing the services provided into distinct layers.
Figure 4.2-1 shows the entire JAFMAS architecture and the classes composing the different layers. The local model is application specific and is left to the user to implement.



USER
OPERATOR INTERFACE
Agent creation (*CreateAgent*)
Agent Operator Interface (*AgentOpInterface & AgentCanvas*)
Conversation Operator Interface (*ConvOpInterface & ConvCanvas*)
Agents (*Agent*)
Conversations (*Conversation & ConvRule*)
MULTIAGENT APPLICATION
SOCIAL MODEL
Requested Resource Provider (*RequestedResrcProvider & ReqdResource*)
Conversation Management (*MsgRouter*)
JDBC
Existing Databases
LOCAL MODEL
LINGUISTIC LAYER
COMMUNICATION PROTOCOL LAYER
Message(*Message & MsgQueue*)
Multicast (*MulticastCom*)
Directed (*DirectedComImpl*)
implements
DirectedCom interface
Legacy Software
Native methods
INFRASTRUCTURE (JDK 1.1)
OPERATING SYSTEM
HARDWARE

Figure 4.2-1 *JAFMAS Architecture*

4.3 JAFMAS Object Model

The following picture shows the JAFMAS object model. Only the most important classes are shown. In order to implement a JAFMAS application, four classes needs to be inherited and extended.

- The CreateAgent class provide a graphical interface to let the user enters the parameters to an agent creation.
- The Agent class implements the local behavior of the agent
- The Conversation class is extended for each conversation that the agent can have.
- The ConvRule class is extende for each rule changing the state of a conversation.

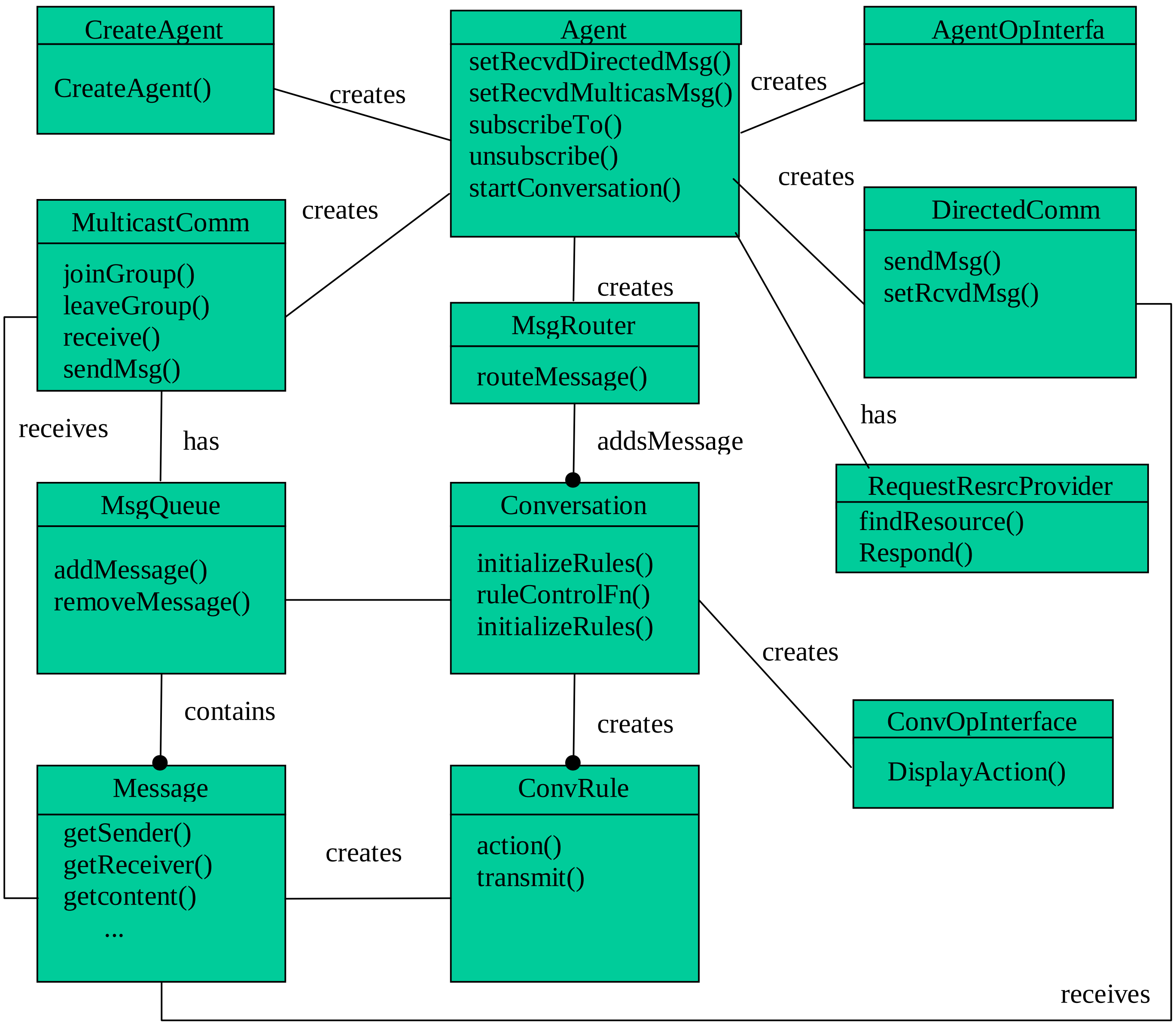


Figure 4.3-1 *JAFMAS Object Model*

4.4 JAFMAS Methodology

JAFMAS proposes a five-step methodology for building a multiagent application. This methodology first focuses on the logical issues of the problem being considered and then converges to the implementation. These primary stages in MAS application development with JAFMAS are:

1. **Identifying the agents**: Starting with a list of aims we want the system to satisfy, we begin by considering various entities that will be interacting with each other to achieve this global aim. These entities represent the agents in the system. Upon identifying the agents, we try to clarify the aims of each agent and the services they provide. We divide these agents into different categories by grouping the agents with similar aims and services into a single category. These categories help us in identifying the agent classes in the application. After identifying the agent classes, we identify the application specific classes that each agent class may use.
2. **Identifying the conversations**: According to Organization Theory the coordination problem can be tackled by having knowledge of the interaction processes among the agents. This knowledge is about the problem solving capability of the entire system, and not of specific agents. When developing agents, we view the world as being composed of intelligent things. We begin to model the interaction between these things in the form of conversations. We identify every possible conversation an agent can engage in, and represent those conversations by developing an automata model for each of them. These different automata models identify the different conversation classes in the application. After identifying the conversations, we identify the application specific classes that the conversations may use.
3. **Identifying the conversation rules**: Each conversation is represented by an automata model. Alternative actions of an agent produce different states in the conversation (automaton) and the current state of a conversation influences how the agent will act/react in the next moment. Thus conversations are rule-based descriptions of what an agent does in certain situations. We need to identify these various actions of an agent and the conditions which bring about rule execution. All this helps us in identifying the conversation rules for each conversation.
4. **Analyzing the conversation model**: In order to design a multiagent system that finds a coherent solution to the entire system problem, it is important to do an analysis of the logical consistency of all agent conversations. They should be analyzed to verify the coherency of the system. Automata models like concurrent finite state machines, and Petri Nets can provide useful tools for checking system coherency found by analyzing the conversation models. If any inconsistency exists in the model, the design is returned to step 2 of the methodology for redefinition of the conversations.
5. **MAS Implementation**: As a final step in the methodology, we need to choose a suitable tool for multiagent application implementation which ensures communication, interoperation and coordination support. JAFMAS provides a set of Java classes that can be used to code MAS applications as a part of this methodology.

4.5 JESS

http://herzberg.ca.sandia.gov/jess/

The Java Expert System Shell (JESS) [17] is a clone of the popular expert system shell CLIPS [34], rewritten entirely in Java. With Jess, programmers can conveniently give Java applets and applications the ability to “reason”.

Developed by Ernest J. Friedman-Hill at the Sandia National Laboratories in California, Jess is effectively an interpreter for a rule language borrowed from CLIPS. Given CLIPS heritage this rule language is basically a small, idiosyncratic version of LISP, making Jess a LISP interpreter written in Java. Like CLIPS, Jess uses the Rete algorithm to process rules, a very efficient mechanism for solving the difficult many-to-many matching problem.

Although Jess can be used as an interactive command-line interface, it can easily be integrated in any Java application. There are even optional Jess commands that let the user create and manipulate Java objects from Jess. Some extra-features are provided so that a reduced version of Jess can be run in an applet. Jess also possesses a networking interface and a model for graphical user interface.

5 The Application

5.1 Description

Even though a large part of the semester was dedicated to the platform investigation, the final goal of the project was to use the most appropriate platform for network management. The application developed restricted itself to the particular case of network management for multimedia application.

Multimedia network applications allow users connected to the network to transfer multimedia information to each other. Such applications require the use of a large network bandwidth. The purpose of the "intelligent network" developed is to satisfy as best as possible the demands or the users by accepting as many requests as possible. Intelligent agents can achieve this goal by interacting both with each other and with the application users, in order to satisfy these requirements. Two types of agents are needed:

♦ User Agents: they implement a layer between the user and the network
♦ Router Agents: they are attached to a particular router and communicate with that router in order to make it as efficient as possible

Figure 5.1-1 below describes the ideal model for such an application.

♦ The user stations, connected to the network, are the end-hosts. Usually an end-host has a unique router through which it is connected to either an internet or an intranet.
♦ A user agent plays the part of an intermediate between the user and the network. It should display a simple graphical user interface to let the user interact with the system. The user agent should also show some intelligence abilities. For example it should be able to remember and learn from the user interaction and react according to it. Typically it is an "interface agent" as described in section 2.1.
♦ The network nodes are the routers managed.
♦ A router agent is attached to each router to manage it. In the ideal case, the agents should be attached to real router doing some substantive routing. The agent does not have to live in the router address space; it can also run on a proxy machine and use standard TCP/IP channels to communicate with it. The agents would then talk to their respective router via the Simple Network Management Protocol (SNMP) [35], or possibly through the simpler more efficient HTTP protocol.

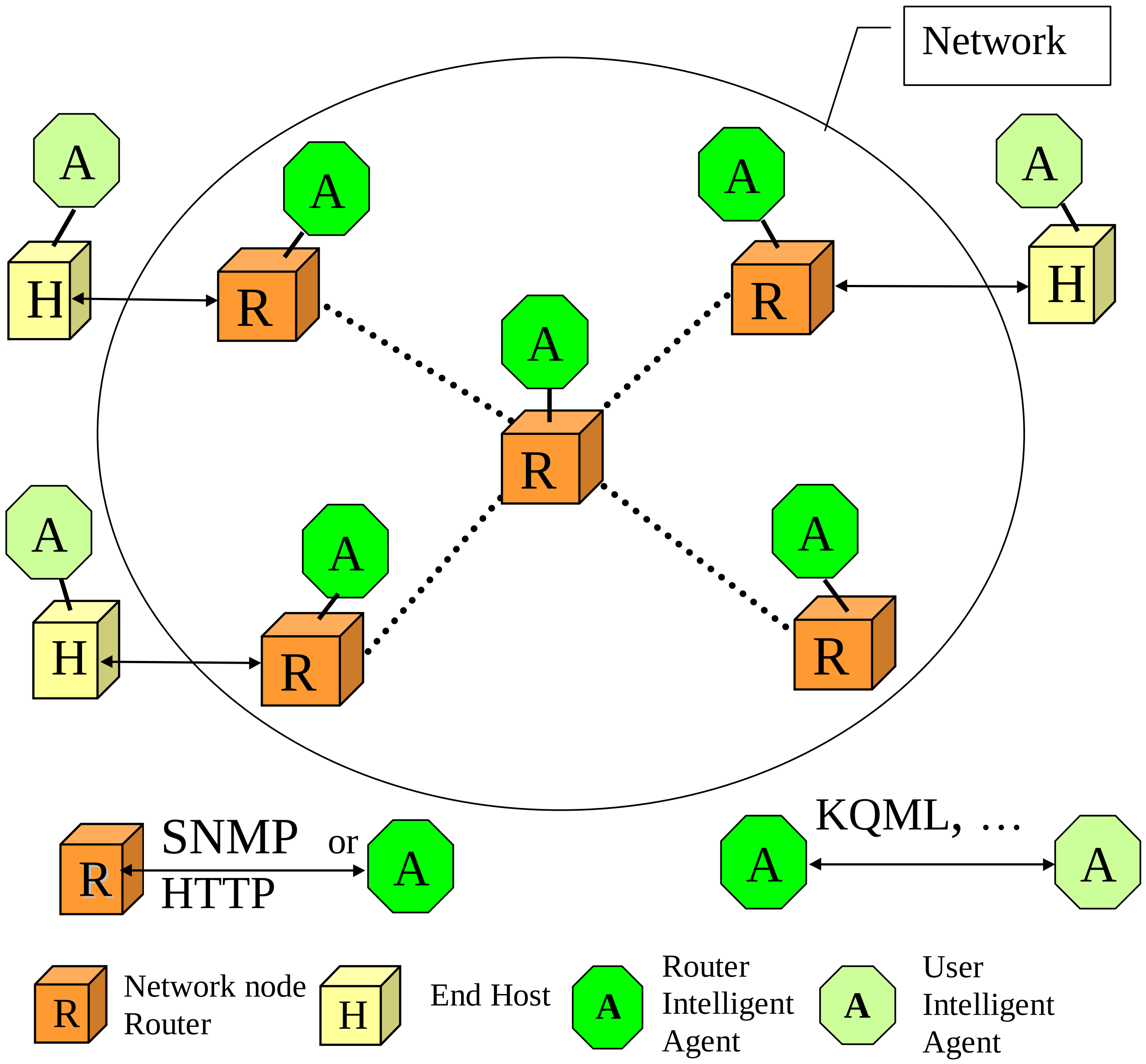


Figure 5.1-1 *Ideal model*

As no network was at hand during the realization of the project, a simple network had to be implemented to be able to test the agents and develop their abilities. In the simple simulated version, all that a router is supposed to do is forward messages where the routing table tells it to. Due to time limitations, routers have been implemented and included inside the router agents. This means that the router agent is also routing messages and acting as the router it is supposed to manage. Therefore, no communication is necessary between these two actors. Also, the communication among intelligent agents has been implemented via message interchange without any use of speech-act languages such as KQML.

5.2 Network simulation

5.2.1 Simple Case: 2 EndHosts 4 Routers

To develop and to test the application, a simple network has first been set up. The network was designed to be large enough so that the concepts could be tested but not too complex so that development was not too time consuming. The network chosen was constituted of four routers and two end-hosts. Management of this network involves six agents, among which two of them are interface agents. This simple arrangement has also the advantage of possessing a redundant path: messages going to the opposite end-host can go through the upper router or to the lower one. That way we will be able to test if dynamic routing is possible, that is, if the upper router is too busy, then the agents should be able to change the routing table in order to make the packets take the lower path.

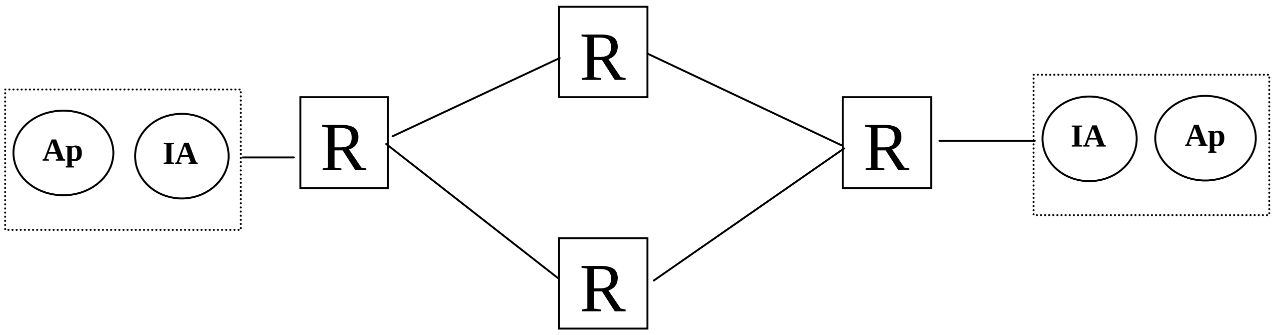


Figure 5.2-1 *Simple network scheme*

5.2.2 Second case: n EndHosts 6 Routers

In order to test the agents at a larger scale, a more consistent network was necessary. Even though each agent can run anywhere on the Internet, the application was tested on one computer. Since only a restricted number of processes can run on the machine the network could not be too large. Therefore, the second network has been implemented as shown on figure 6.2-2. This second case allows as many end-hosts as desired to be attached to each router even though only two are represented in the figure below. This scheme lets network traffic go in diverse direction and therefore gives the opportunity to test router overload and dynamic configuration.

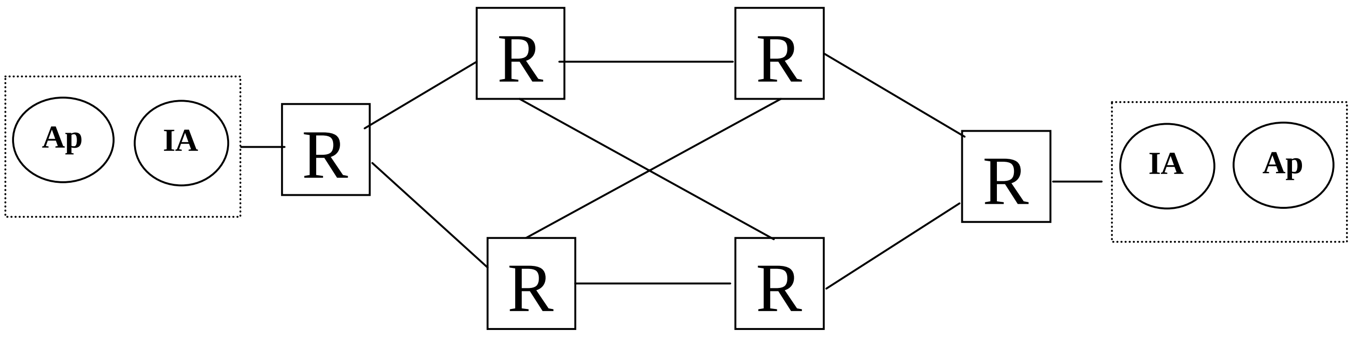


Figure 5.2-2 *Second Network scheme*

5.3 MAS Implementation

As said previously the network implementation has been included inside the agents. Therefore two main actors constitute the system: the router agent (including the router implementation) and the end-host agent. This system, following the JAFMAS infrastructure, is organized as pictured in the object model below.

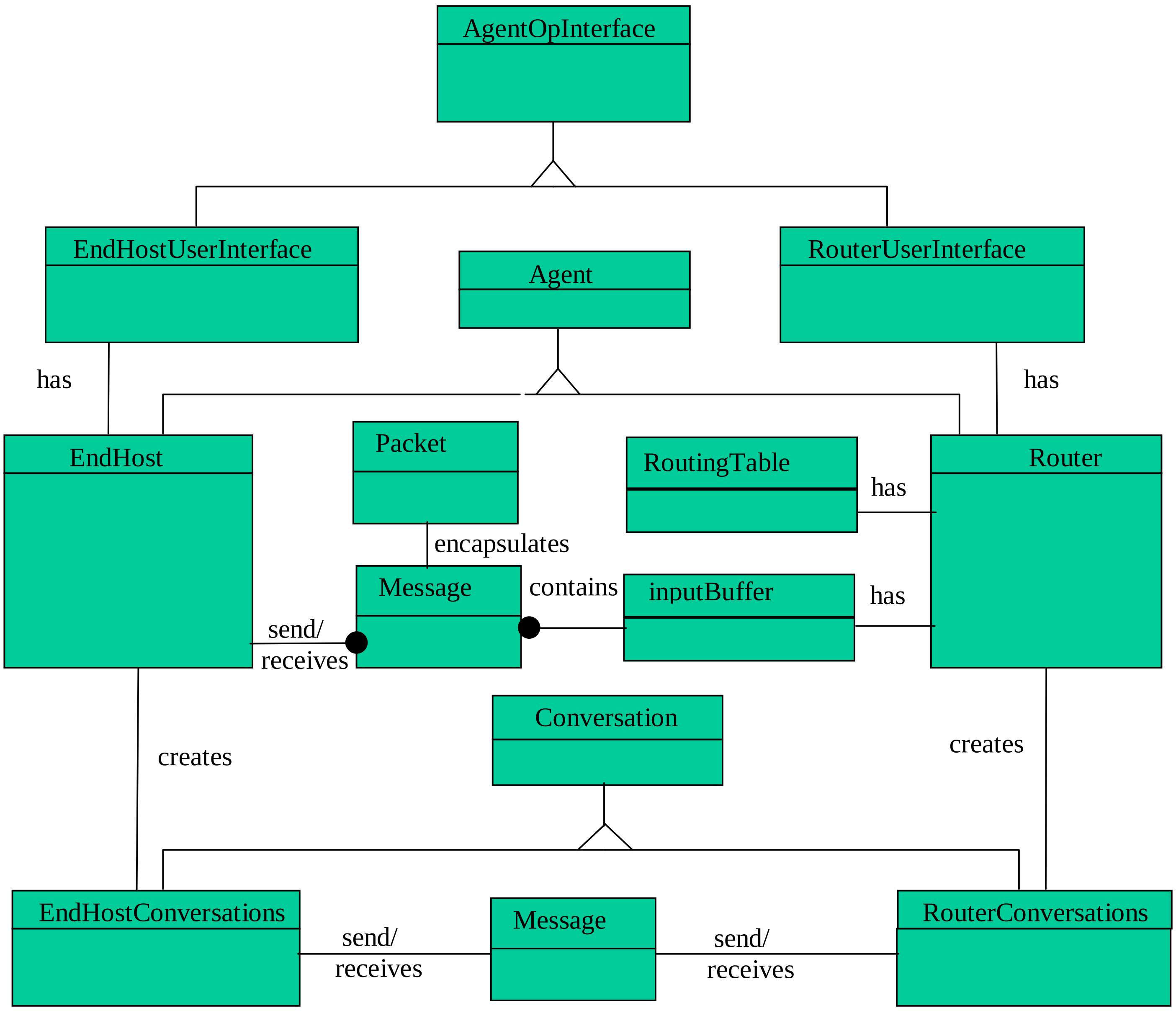


Figure 5.3-1 *Overall agent object model*

5.3.1 Router Agent

The router agent is an agent managing a network node. It is composed of two layers:

a) Router Layer

The router layer is the network itself. Its goal is to route the arriving messages according to their destination. A simple routing table is associated to it so that the router knows where to forward the messages. A fixed size buffer is also used to temporary stock the arriving messages. The router treats the messages as fast as it can. If the buffer is overloaded, the extra messages are discarded. It is the work of the agent to synchronize the traffic so that no messages are lost. Each router possesses also a certain bandwidth. That means that no more bandwidth than what is possessed by the router can be allocated to user requests. A simple graphical user interface is provided so that information can be observed on what is happening. One can also change the router bandwidth, option useful for testing and debugging.

b) Intelligent Layer

The intelligent layer is the actual intelligent agent. It communicates with other agent's intelligent layers and with the router layer in order to manage the node properly. When it has to accomplish a goal that involves other agents, it creates a Conversation object (inherited from JAFMAS) to talk with other agents and reach the given objective.

5.3.2 End-host Agent

The end-host Agent plays the role of intermediate between the user and the network. A graphical interface is displayed so that the user can interact with the system. Ideally, the end-host agent should interface the multimedia application, so that the whole agent infrastructure is invisible to the user. In our case, as described in section 5.7, the user has to explicitly reserve some bandwidth or send messages. Only fake data is sent, however the simulated network is able of transporting real data and therefore act as a real network.

However, the end-host agent is not only a composition of graphical component. The underlying layer should support an intelligent infrastructure able of conversing with the network. An end-host is usually attached to a unique router. The end-host agent will have to talk with his router agent in order to reserve some bandwidth. It should also be able to interpret the demand requested by the user. For example, if the user asks for a good connection, the end-host agent should know (or learn with feedback from the user) that a good connection is about 500 kb/s.

5.3.3 Message Class

The message class encapsulates all the information exchanged amongst agents. Every instance of that class has a performative variable. This variable represents the nature of interaction that is going to take place. The class provides methods for specifying both the sender and the receiver agent. It is also possible to specify the intent of the message. The intent enables the sender agents to express the intention which required them to send the message, and the receiver agents to filter the message upon looking at only the intent slot. It also facilitates message routing. Messages are of two general types:

- ♦ **Declarative messages** are used to announce the presence of an agent. This type of message is used to set up the network at the beginning: each agent broadcasts his name and then waits for the agent he is connected to to answer.
- ♦ **Content messages** contain a description of the piece of knowledge being accompanied with the actual data. They are used to transport data packets along the network. A data packet can be assimilated to an IP packet on the Internet. It possesses the sender address, the destination address and the actual data.

5.4 Features Implemented by the MAS

5.4.1 Connection Admission Control

The first feature implemented by the agent infrastructure is to admit or refuse connection according to the network load. A user asks his agent for a connection to another end-host for a certain bandwidth. The user agent has then to communicate with his router agent to determine if the new connection can be accepted or not. In order to implement that service a path reaching the desired destination has to be found throughout the network. The simple following algorithm provides a solution assuming "*I*" is a router agent.

```
If I have bandwidth then
    If I have next neighbor
            Ask next neighbor
            If next neighbor ok then
                    Send Accept to previous neighbor
            Else
                    Send Refuse to previous neighbor
    Else /* the end-host is reached */
            Send Accept to previous neighbor
Else /* no bandwidth available */
    Send Refuse to previous neighbor
```

Algorithm 1 *Simple Connection Admission control*

The difficulty in implementing this algorithm is to find the way back. The Routing table only gives information about how to reach a host, but several paths can lead to the same host. The solution adopted is to include a stack of visited hosts in the message so that the path used by the message is contained in itself.

This first algorithm works fine. However, it does not take into account the fact that redundant paths can exist in the network. This second algorithm corrects that problem.

```
If I have bandwidth then
    If I have neighbors
        While I have neighbors
            Ask neighbor
            If neighbor ok then
                Send Accept to previous neighbor
                exit
        Send Refuse to previous neighbor
    Else /* the end-host is reached */
        Send Accept to previous neighbor
Else /* no bandwidth available */
    Send Refuse to previous neighbor
```

Algorithm 2 *Alternative Path Connection Admission Control*

5.4.2 Reservation Mechanism

This new mechanism allows a user to reserve a channel between him and another user. The naïve approach would be for the router agent to simply reserve the demanded QoS before sending back the “accept” message in the previous algorithm. However, this approach does not prevent several users from reserving the same resources. In other words, some concurrency issues have to be taken into account. To avoid such a situation, the bandwidth variable has been “locked” so that only one process at a time can change its status.

5.4.3 Translate Demand into Bandwidth

In order to make the user interface more friendly, the system should allow the user to choose between different levels of QoS rather than directly specifying a network bandwidth. It is then up to the agent to translate that QoS specification into the corresponding bandwidth. For the time being, this feature is implemented via a simple mapping table between the displayed levels and the bandwidth. A possible improvement would be to set up the QoS according to the destination desired. For example, a channel from Switzerland to Africa would have a much lower QoS than a channel inside a local area network.

5.4.4 Unsatisfied Demand

If the router agents refuse a demand, the user can then choose amongst two utilities.

- ♦ The user agent is not only intelligent, but also possesses some kind of simple memory. Indeed, if the router agents refuse a demand, the user agent can nonetheless memorize it. The user has therefore the possibility of delegating the tedious task of renewing the request at regular interval to the user agent. The user can also specify at what frequency the agent should recontact the network. When the request is finally accepted, the user is notified and the multimedia session can start. This feature is implemented via a thread that sleeps during the given interval, and then recontacts the network.
- ♦ If the QoS is not essential to the user, he can alternatively choose to request the user agent to find the highest available QoS. In this case, the user agent will renew the demand with decreasing QoS levels until the demand is accepted.

5.4.5 Auto-Configuration

Another feature implemented is the dynamic configuration of the network. When a host goes down and an alternative path stays available, the neighboring hosts should update their routing table so that the new coming requests will go through the alternative path. There are two ways for a router to know that one of its neighbors is down:

- ♦ The first one is that the neighbor informs the router that it is about to go down before dying. This is certainly not very realistic
- ♦ The second one is that when the router tries to contact the neighbor no response is obtained.

In both cases, the routing table should be updated as soon as the failure is detected.

5.5 Additional Possible Features

5.5.1 Feedback from Overloaded Routers

If a router is overloaded by incoming messages, his attached agent should contact the neighboring sending agents to inform them that messages are being lost. The agents should then coordinate their efforts to find a more efficient solution. The simplest solution would be to contact the sending end-host to warn them about the situation.

5.5.2 Optimum Path Search

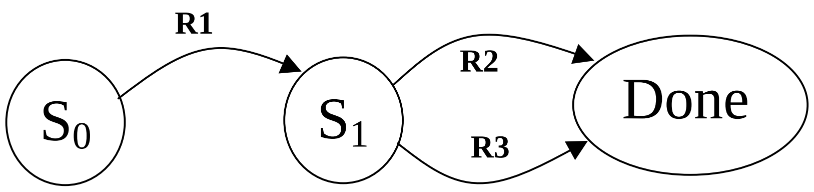
Although the previously adopted algorithm (Algorithm 2) finds a path if it exists, it does not necessarily find the best one. An alternative solution would be to search through all the paths available to reach a given end-host and to reserve the optimum path, taking into account the cost of using each router. This type of approach could, for example, be of use in the cases where a pricing mechanism would be set up for each router. This would typically be the case if the routers were owned by different companies and each company had its own tariff. Such an algorithm would always find the cheapest path to talk to a given end-host. A compromise between price and QoS is another possibility that could also be considered.

5.6 Conversations

In order to implement the reservation mechanism described in section 5.4.2, two "JAFMAS conversations" have been implemented.

5.6.1 EndHostAskConv

EndHostAksConv is the conversation started by the user agent when a reservation is needed. The figure below represents the states through which the conversation can go, and the rules allowing to move from one state to the other.

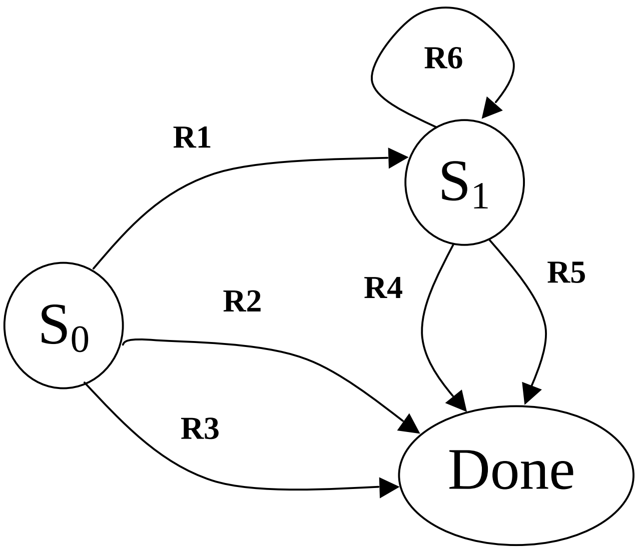


R1 **Transmit:** ask Qos to router agent
R2 **Receive**: refuse from my router
DoAfter: display qos refused
R3 **Receive**: accept from my router
DoAfter: display qos accepted

Figure 5.6-1 *EndHostAskConv*

5.6.2 RoutAcceptConv

RoutAcceptConv is the conversation started by each demand received by a router agent. It follows Algorithm 2 described above.



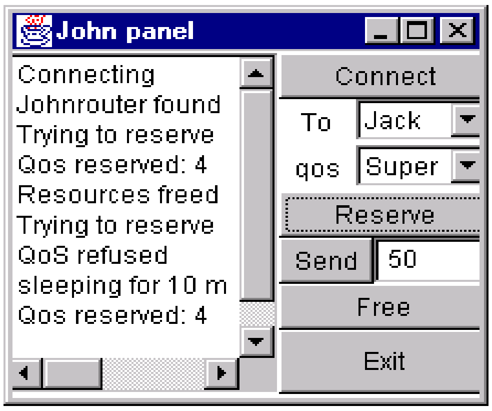
R1 **Receive:** Change message
Suchthat: I have bandwidth and I have neighbors
Transmit : change to next neighbor
R2 **Receive**: Change message
SuchThat: I have bandwidth and no next neighbor
Transmit: Accept to previous neighbor
R3 **Receive**: Change message
S**uchThat**: I don't have bandwidth
Transmit: Refuse to previous neighbor
R4 **Receive**: accept from next neighbor
Transmit: accept to previous neighbor
R5 **Receive**: refuse from next neighbor
SuchThat: no alternative path
Transmit: refuse to previous neighbor
R6 **Receive**: refuse from next neighbor
SuchThat: alternative path
Transmit: change to alternative neighbor

Figure 5.6-1 *RoutAcceptConv*

5.7 User Interface

5.7.1 User Agent GUI

First of all the user has to click the connect button in order to be connected to his network. Once connected, the user can select the end-host name to which he wants to communicate with and the QoS at which he would like the channel to be established. After reserving a channel, the number of messages to be sent can be selected and the send button will send those messages through the network.


John panel

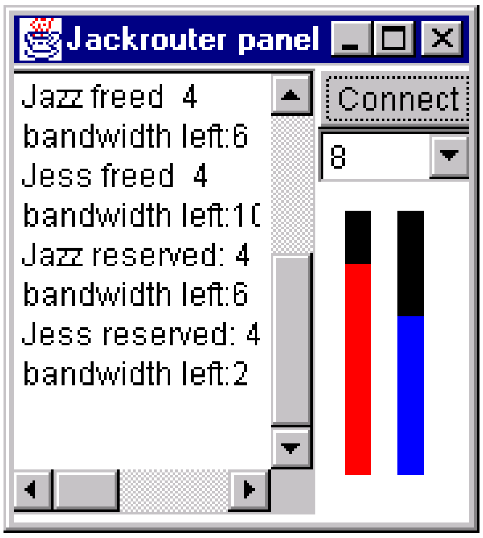
```
Connecting
Johnrouter found
Trying to reserve
Qos reserved: 4
Resources freed
Trying to reserve
QoS refused
sleeping for 10 m
Qos reserved: 4
```

To: Jack
qos: Super
Send: 50

Figure 5.7-1 *User Agent GUI*

5.7.2 Router GUI

The red level shows how much bandwidth is reserved for the router.
The blue level shows how many messages are contained in the buffer.
The bandwidth can also be set via a multiple-choice roll-down menu for testing purpose.

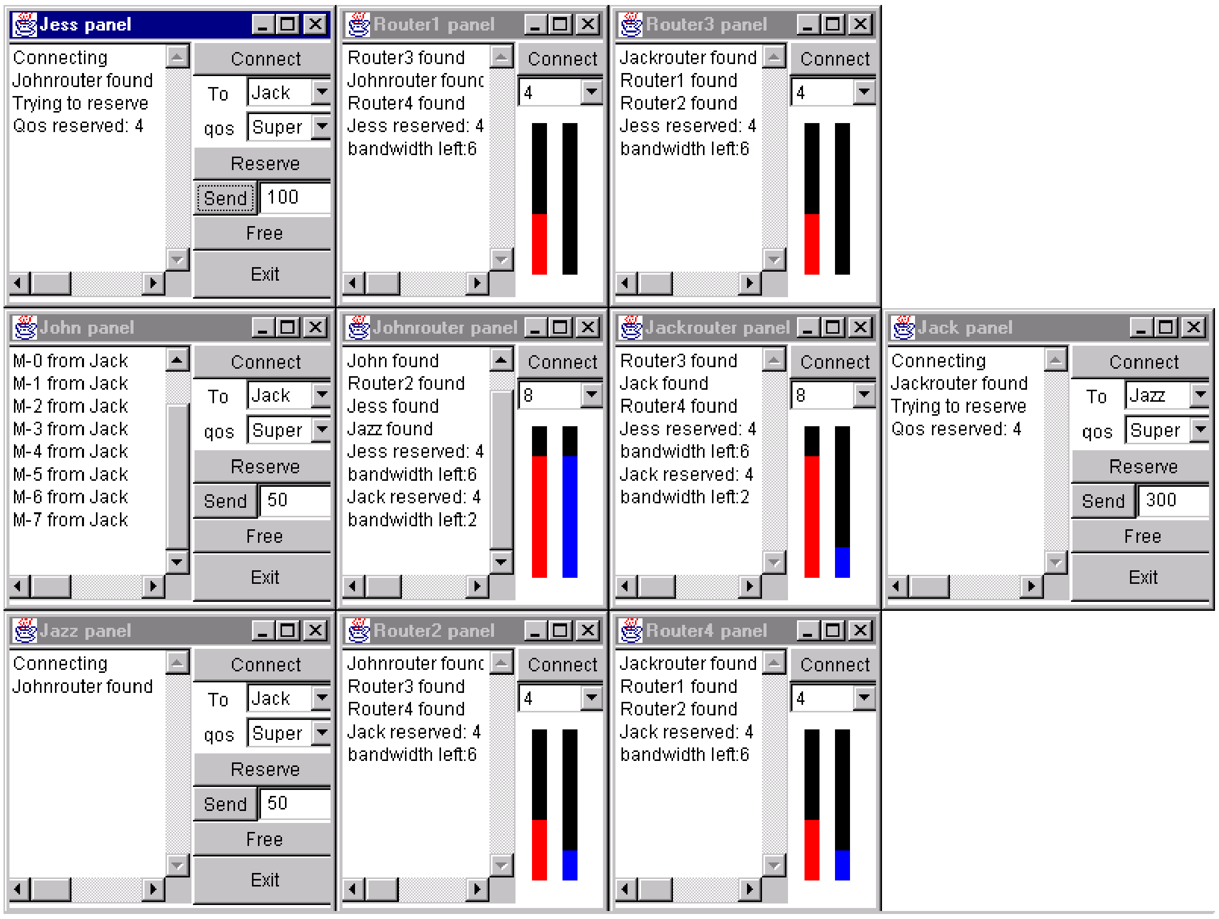

Jackrouter panel

```
Jazz freed  4
bandwidth left:6
Jess freed  4
bandwidth left:1(
Jazz reserved: 4
bandwidth left:6
Jess reserved: 4
bandwidth left:2
```

8

Figure 5.7-2 *Router Interface*

Figure 5.7-3 shows the agents in action for the second implementation of the network. Four users are connected to a network of six routers.Two connections are actually reserved: Jess - Jack and Jack – Jazz..



Jess panel:
```
Connecting
Johnrouter found
Trying to reserve
Qos reserved: 4
```
To: Jack, qos: Super, Send: 100

Router1 panel: 4
```
Router3 found
Johnrouter founc
Router4 found
Jess reserved: 4
bandwidth left:6
```

Router3 panel: 4
```
Jackrouter found
Router1 found
Router2 found
Jess reserved: 4
bandwidth left:6
```

John panel:
```
M-0 from Jack
M-1 from Jack
M-2 from Jack
M-3 from Jack
M-4 from Jack
M-5 from Jack
M-6 from Jack
M-7 from Jack
```
To: Jack, qos: Super, Send: 50

Johnrouter panel: 8
```
John found
Router2 found
Jess found
Jazz found
Jess reserved: 4
bandwidth left:6
Jack reserved: 4
bandwidth left:2
```

Jackrouter panel: 8
```
Router3 found
Jack found
Router4 found
Jess reserved: 4
bandwidth left:6
Jack reserved: 4
bandwidth left:2
```

Jack panel:
```
Connecting
Jackrouter found
Trying to reserve
Qos reserved: 4
```
To: Jazz, qos: Super, Send: 300

Jazz panel:
```
Connecting
Johnrouter found
```
To: Jack, qos: Super, Send: 50

Router2 panel: 4
```
Johnrouter founc
Router3 found
Router4 found
Jack reserved: 4
bandwidth left:6
```

Router4 panel: 4
```
Jackrouter found
Router1 found
Router2 found
Jack reserved: 4
bandwidth left:6
```

Figure 5.7-3 *Agents in action*

6 Conclusion

6.1.1 Summary

All in all, we have seen that using intelligent agents for network management is absolutely feasible. Although the application implemented is still very small, some interesting concepts have been implemented and proved to be efficient. The ideas that have come out of this research could certainly be used at a much larger scale. In particular, the elected platform has fulfilled our expectations, and could certainly be of great use for this type of application. The multiagent infrastructure set up is only a small glimpse of what could potentially be done for network management. Although the speed of execution of the code is far from being impressive, advance in technology should help overcome this handicap in the near future. For instance, compiling the code rather than interpreting it would give an impressive speed-up on a PC running Windows.

6.1.2 Benefit

Since the project did not have an explicit restrictive specification, I was forced to look for the information by myself and to evaluate it. Indeed, a large amount of time was spent on the platform investigation. Even though browsing the web for long hours can get boring, a lot was learned in the way information can be obtained. At first, I had the impression I was not being very efficient, but I now realize that those long hours are necessary and pay off in the long run. The time invested in finding an appropriate platform largely balanced out the time spared during the development.

In order to do the investigation, a theoretical background in artificial intelligence and network management was necessary. Starting off at a novice level, I have learned a lot in the network management domain. The concept of intelligent agents was also new for me and was of a great interest.

6.1.3 Future work

A simple application applying some basic concepts has been built around a multiagent infrastructure. But now that the infrastructure is set up and the network traffic is simulated, more time could be dedicated on network management itself. A wide range of possibilities is waiting to be implemented in order to make the system more efficient. Future work on this project should focus on making the agent more intelligent and giving them an appropriate way of conversing with other agents.

A speech-act language could be used in order to have a more flexible conversation protocol. KQML seems to be an appropriate language for this kind of application. JKQML [25], the Java version of KQML implemented by IBM, should be investigated in more details, as it would provide all the needed KQML features and be very easy to integrate in the actual system which is already implemented in Java.

The actual implemented agents show some intelligence, but a lot more could be done. In order to improve the agent ability to reason, Jess [17], the Java expert system shell could be incorporated inside each agent. With Jess our agents would be able to reason and act accordingly.

But what is reasoning without knowledge? A knowledge base would possibly be necessary to give the agents something to reason about. Then, if the knowledge base gets larger, each agent could maintain a database connectivity so that our agents would now have a brain and a memory.

With such agents, the system would then be able to provide services like:

1. Learn from the user interaction, and help him during session set up
2. Dynamic configuration of the network to be as efficient as possible
3. If network bandwidth is charged, a pricing mechanism should find the cheaper path to reach a given end host.
4. The actual reservation mechanism is unidirectional, an option should allow a user to reserve a bi-directional path so that his partner can communicate back. Also, a group of users should be able to interact via a dedicated channel to do video-conferencing for example.
5. Agents could also provide some coordination mechanism. For example, an end host agent should help a user to locate and contact a partner. More information in that domain can be obtained in a paper by Albayrak et al. [1].
6. The QoS allocation could be done according to a market-based approach, as described in the Yamaki paper [38].
7. And many more …

Acknowledgments

I wish to thank Professor Jean-Pierre Hubaux and the entire ICA staff for giving me the opportunity to participate in their research study and use their material. In particular, I am very grateful to my assistant, Jean-Philippe Martin-Flatin, for his continual and supportive help throughout the semester. Finally, it was a pleasure to work with my colleague, Christophe Andrey, with who I worked during the first part of the semester.

Bibliography

[1] Albayrak Sahin, Haase Raik, Riegel Holger, The Unified Messaging Agent Service (UniMAS), DAI Lab, Technical University of Berlin, Germany

[2] Aparicio, M.,"IBM Intelligent Agents", FIPA Opening Forum Proceedings, Yorktown, New York, 1996

[3] Appleby, S & Steward, S., Mobile Software Agents for Control in Telecommunications Networks, BT Technological Journal 2, pp. 104-113, April 1994

[4] Bradshaw M. Jeffrey, An Introduction to Software Agents

[5] Case J., Fedor M., Schoffstall M. and Dawin J., RFC 1157. A Simple Network Management Protocol (SNMP), IETF, May 1990

[6] Concordia, Mobile Agent Computing – A White Paper, http://www.meitca.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html, Mitsubishi Electric Information Technology Center of America, 1997

[7] DARPA Knowledge Sharing Initiative, External Interfaces Working Group, Specification of the KQML Agent-Communication Language, June 1993

[8] David R. and Alla H., Petri Nets and Grafcet - Tools for modelling discrete event systems, Prentice Hall, 1992.

[9] Deepika Chauhan, JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation, ECECS Department, University of Cincinnati, 1997

[10] dMars, http://www.aaii.oz.au/proj/dMARS-prod-brief.html, Australian Artificial Intelligence Institute (AAII), 1996

[11] Finin Tim and Fritzson Rich, KQML - A Language and Protocol for Knowledge and Information Exchange, Computer Science Department, UMBC

[12] FIPA application Types, http://www.cselt.stet.it/fipa/fipa_rationale.htm, 1996

[13] Flanagan David, Java in a Nutshell, 2nd edition, O'Reilly, May 1997

[14] Foner L., What is an agent anyway? : A Sociological Case Study, MIT Media Lab, Cambridge, MA, 1993

[15] Foner N. Leonard, What's an Agent, Anyway?, Agents Group, MIT Media Lab

[16] Forrester, Source : Adatpted from Forrester Research, Inc. (May 1992)

[17] Friedman-Hill J. Ernest, JESS – The Java Expert System Shell, Distributed Computing Systems, Sandia National Laboratories, Levermore, CA, 1998

[18] Genesereth R. Michael, Knowledge Interchange Format: Specification, Stanford University, March 1995

[19] Harold Elliotte Rusty, Java Network Programming,1st edition, O'Reilly, February 1997

[20] IBM Aglets, IBM Aglets: Programming Mobile Agents in Java, A White Paper, http://www.ibm.co.jp/trl/aglets/whitepaper.htm, IBM Tokyo Research Laboratory, 1996

[21] ICMAS '95, Proceedings of the 2nd International Conference on Multi-Agent Systems, The AAAI press, 1995

[22] ICMAS '96, Proceedings of the 2nd International Conference on Multi-Agent Systems, The AAAI press, 1996

[23] ITU-T, Recommendation X.711. Data CommunicationNetworks - Open Systems Interconnection (OSI); Management. Common Management Information Protocol Specification for CCITT Applications. ITU, Geneva, Switzerland, March 1991.

[24] JATLite, JATLite overview, http://java.stanford.edu/java_agent/html/JATLiteOverview.html/, Stanford University, 1997

[25] Java IBM Yamato Lab, http://www.alphaworks.ibm.com/formula/jkqml/, April 98

[26] Maes Pattie, MIT Media Laboratory Project, Software Agents, http://casr.www.media.mit.edu/groups/casr/maes.html

[27] Martin-Flatin J.P., A Survey of Distributed Enterprise Network and Systems Management Paradigms, Submitted to JNSM, Special Issue on Enterprise Network and Systems Management, December 1997

[28] Niemeyer Patrick and Peck Joshua, Exploring Java, 1st edition, O'Reilly, 1996

[29] Nwana H.S., Lee L. & Jennings, Coordination in Software Agent Systems, British Telecommunication Technology Journal 14, pp 21-42, October 1996.

[30] Odyssey, Odyssey Frequently Asked Question, http://www.genmagic.com/agents/odyssey-faq.html, General Magic Inc., 1997

[31] Rose T. Marshall, The simple book : An introduction to Networking Management, Second Edition, Prentice Hall, 1996

[32] Rumbaugh James, Object Oriented Modeling and Design, Prentice Hall, 1991

[33] Russel Stuart & Norvig Peter, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995

[34] Savely Robert, CLIPS Reference Manual – Advance Programming guide, 1997

[35] Stallings William, SNMP, SNMPv2 and CMIP : the Practical Guide to Network Management Standards, Addison-Wesley Publishing Company, 1993

[36] Voyager, Voyager Technical Review, http://www.objectspace.com/voyager/voyager_white_papers.html, ObjectSpaceInc, 1997

[37] Wooldridge Michael, Jennings R. Nicholas, Intelligent Agents: Theory and Practice, Submitted to Knowledge Engineering Review, October 1994.

[38] Yamaki Hirofumi, Wellman P. Michael, Ishida Toru, A Market-Based Approach to Allocating QoS for Multimedia applications, Departement of Information Science, Kyoto University, Japan.