Management of QoS with Software Agents

Hermann de Meer[1], Antonio Puliafito[2], and Orazio Tomarchio[2]

[1] Department of Computer Science
Telecommunication and Computer Networks
University of Hamburg - Germany
E-mail:demeer@informatik.uni-hamburg.de

[2] Istituto di Informatica, Università di Catania
Viale A. Doria 6, 95025 Catania - Italy
E-mail:{ap,otomarch}@iit.unict.it

Abstract

As distributed multimedia applications become more widely diffused, systems for QoS management are increasingly essential. In this paper we present an architecture for distributed QoS management based on software agents. We discuss the advantages of using agent technology in a problem inherently distributed and complex such as QoS management. Besides introducing several negotiation and coordination techniques among agents as fundamental properties for QoS management, it is argued in favor of decision making based on analytical and simulative modeling techniques. Furthermore, design and implementation issues are discussed in detail and our Java-based platform for an agent-based QoS management is introduced.

1 Introduction

Recent technological developments in the field of high-speed networks and low-cost multimedia stations are pushing the development and diffusion of new types of distributed multimedia applications. In particular, various services originally tight to different special-purpose networks can now be integrated into a single general-purpose network with high capacity (B-ISDN) [Jun96]. As another important property, the services are increasingly distributed in a network of indefinite size.

The new types of applications comprise several media, which range from video to audio, but include also traditional discrete data. In such applications, requirements are different in terms of throughput, acceptable delays, jitter, and lost and/or corrupted data units. These performance characteristics are particularly important as far as they translate to quality differences *perceivable by users*. Furthermore, *quality guarantees* must be provided to advanced applications incorporating continuous media such as audio and video [SN95]. A guarantee may extend over various levels and include a degradation path which may also have to be *negotiated* on between service providers and service users [RdM97].

Although many new architectures have been proposed for support of multimedia systems with QoS requirements as opposed to more traditional best-effort systems [ACH97], several shortcomings remain to be fixed. Among them, the provisioning of distributed control of resource usage and a system-wide customization of negotiated QoS levels is of predominant importance. User behaviors and system facilities like operating and communication system components have to be equally taken into account. Different types of agents with different specialties are to be applied for this purpose. Furthermore, such systems could be indefinitely large so that scalability is mandatory. A geographically distributed and functionally separated *QoS-management* scheme is needed that works under the indicated constraints. A distributed QoS architecture and management scheme must be highly flexible and compensate adaptively for environmental changes that can continuously occur due to randomly occurring failures, overload situations, or change of user behavior and intention.

Interesting proposals have been made, for example by Goldszmidt et al. [GY95], to overcome these shortcomings. But that work focuses merely on network management issues and therefore neglects the role of users and their cognitive competence and requirements with respect to QoS provisioning.

The approach we follow has been initiated in [FHvBdM97]. The concept of agent technologies [GK94, EW95] is applied for a *cooperative QoS management* based on autonomous agents. The cooperative QoS management scheme exhibits several characteristics traditionally being attributed to AI systems focusing on agent frameworks. These characteristics include *matching of conflicting interests* and complex *negotiations* between service providers' agents and service users' agents, thereby taking multidimensional parameter spaces into account. Furthermore, negotiation and coordination take place between service providers' agents in order to fulfill their *commitments* concerning given QoS guarantees on the one hand and the need of *efficient usage* of scarce resources on the other hand. *Agents adapt* to the particular requirements of certain applications as well as to temporary changes in the environmental conditions of system operation. Monitoring and measurements are incorporated in order *to control* user agents as well as system agents with respect to their mutual and individual commitments. Furthermore, system agents may have different roles in the collaborative provisioning of the committed service and its quality. The *interactions* between agents include exchanges of notifications, recommendations, and commands. The major focus is placed on cooperative recommendations as a basis for *coordination*. To achieve a reasonable cooperation, the agents are provided with information about the intentions of other agents. More details about the agent concept will be provided in Section 3.

Decision making is based on a *model-based* approach. *Analytical* as well as *simulative* techniques are explored to provide formal support of agent decisions with respect to resource reservation and adaptation strategies.

Besides discussing the concept of agent-based cooperative QoS management we will also focus on design and implementation aspects of our architecture.

It will be discussed how our Java-platform suits the needs and requirements of the agent architecture as a basis for implementation. In particular, attributes like openness and customizability are attractive for our purposes. In addition, migration of agents can relative easily be implemented, thus fostering adaptiveness and flexibility. The ultimate goal will be to support a progressive concept of programmable networks [Laz97].

Section 2 contains discussions of important issues of QoS management, with particular emphasis to conversational multimedia applications. In Section 3, after setting the reference context, we describe the proposed QoS management architecture, and we indicate the functions of each agent. In Section 4 we describe our Java-based platform for QoS management and in Section 5 a case study is presented. Finally, in Section 6 we give the conclusions and the prospects for the use of the proposed platform.

2 A Glimpse on QoS Management for Conversational Multimedia Applications

Before the agent-based framework can be discussed in more detail, some technical preliminaries shall be given underlying our conversational multimedia application. The architecture which we will refer to is shown in Fig. 1. Users are connected through the network (WAN) to providers of multimedia service. Specifically, our attention focuses on distributed multimedia conversational applications which are based on point-to-multipoint communication. Examples of such a scheme include educational application, such as teleteaching, and teleconferencing which are usually characterized by streams sent out by a source and received by several users. As the most suitable communication structure, it is suggested to support these types of applications by means of *multicast* techniques. With multicast, communication structures in form of trees are built up and organized according to the applications' and users' needs. An interesting feature of a multicast tree is the fact that it is logically shared to a greater extent among several users or applications. The further the sharing goes, the more cost effective the service providers can be. A larger extent of sharing will lead to *lower overall cost*. To the contrary, it is the declared goal of QoS architectures to suit users' and applications' requirements individually as far as possible and as flexible as possible. The goal here is to *minimize the cost individually* and to take possible technological constraints typically inherent in low capacity end-user equipment into account. Note that these issues of cost-effectiveness are not deliberately raised here, they are rather essential to the success of future distributed multimedia systems.

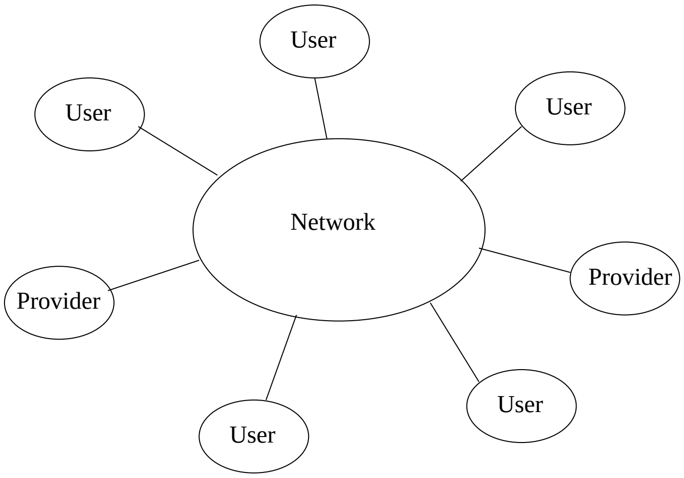

Figure 1: Reference Architecture.

The picture is even a little more complicated due to the fact that these systems will be typically applied in a highly dynamic environment. Network and other resources will be constantly removed from service and added back in due to failure and repair processes or due to general system evolution. Furthermore, overload situations will be rather common and have to be dealt with because multiplexing and resource sharing techniques are applied for reasons of efficiency.[1] It might also occur that the load imposed by users does

[1]Note, multiplexing is not contradictory to resource reservations. Both principles will be applied at the same time to cope efficiently with variable rate traffic. But a detailed

not convey with what had been anticipated. Appropriate measures have to be taken here according to some conventions since commitments were not respected individually, perhaps leading in turn to an overall critical state.

All the scenarios discussed in the last paragraph have in common that the management must be highly flexible and react appropriately to threatening events from the environment. As a measure, for example, the multicast tree could be reorganized by allocating more or alternative resources, if available. A more severe step would be to gracefully adapt the provided levels of qualities to lower, less resource demanding levels. Note that such a degradation has to be performed along a *prespecified degradation path* which has to be considered as part of the service providers' initial commitments. If adaptation or reconfiguration measures do not work out, it can be tried to cancel the original commitments and to establish new ones by means of *re-negotiations* with service users. Note that several evaluations and decision making processes are inherent to these tasks. Even a predefined degradation path can include several options which have to be judged with respect to some utility functions.

Of course, system states do not only degenerate but improve also sometimes. Our QoS management concept takes this also into account such that the "added value" may be taken advantage of in several ways. New services can be offered but existing ones can also be upgraded instead.

It should have become clear from the discussion in the preceding paragraphs that users must not freely access services offered by such a multimedia system. Rather, scarce resources must be reserved individually and have to be managed according to negotiated levels of quality and the implied cost. Once negotiated, services and their qualities will be guaranteed as mutual commitments between service providers and users. Note that the negotiation may be a multilevel process due to the inherent complexity in matching possibly contrasting expectations between service users and providers. The parameter space is usually of multidimensional form. Different dimensions are not mutually independent, but can be traded with each other in relation

---

discussion of these issues is beyond the scope of this paper.

to the currently available resources. The multidimensional QoS parameter space must be mapped on system resources in order to verify the possibility of accommodating new applications [KdM97]. Decisions must be made according to some utility functions whether new multimedia calls should be accepted or rejected from the beginning [dMF97]. Renegotiation may also be started if a user changes his mind and asks for lower or higher qualities. Note that the mapping of QoS parameters as perceived by users to system resources has to be performed across multiple levels and must incorporate all resources in a distributed system.

A simplified view is presented in Fig. 2. This structuring entails different levels of expertise which must be covered by a QoS management scheme.

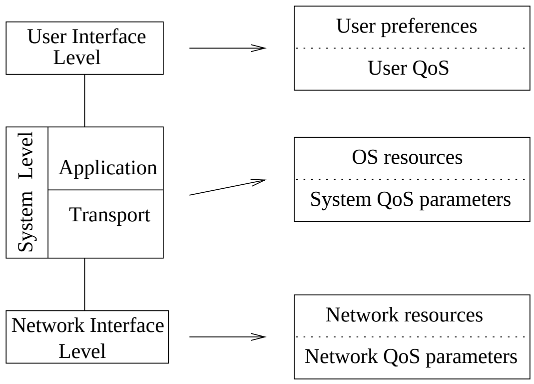

Figure 2: Various levels of QoS management expertise within end-systems.

Finally, service users and providers have to be monitored with respect to their joint QoS commitment. Reasons for QoS violations can thus be detected, responsibilities identified, and possible actions initiated according to some conventions as discussed in preceding paragraphs.

To summarize, we have identified the following main functions of QoS management:

- QoS negotiation,
- QoS admission control,

- QoS renegotiation,
- QoS mapping (among different levels),
- resource reservation,
- QoS monitoring,
- QoS adaptation.

3 An Agent-Based Framework for QoS Management

In this section we show how agent technology can be used to implement QoS management in a distributed environment. Specifically, we provide some concepts on commitments and conventions of system agents, discuss how agents interact with resource management facilities, outline their specific functions, and sketch the way in which they cooperate.

3.1 Distributed AI and QoS management

From the preceding discussions it is clear that an agent-based approach provides a promising design and implementation concept for our purpose. The identified main functions of QoS management resemble in many ways characteristics that have traditionally been attributed to distributed artificial intelligence. The relation will be discussed in more detail in what follows. Note that the type of system we are advocating may be composed of a large number of components, i.e., thousands or even indefinitely many nodes and links in a network, which all have to be organized such that several - competing - commitments are met. Therefore, *scalability* is of predominant importance. Consequently, we are advocating completely distributed control to be implemented in form of autonomous agents.

3.1.1 Commitments and conventions of system agents

We have provided a framework for distributed control in which agents can reason about their commitments. Commitments, in general terms, encompass two aspects:

1. Efficient usage of scarce resources,

2. Provisioning of QoS guarantees.

Since these tasks collide, compromises have to be found and decisions must be made to what extent a service with a given quality level can be supported. Admission control is performed on the basis of utility functions computed with the help of analytical and simulative models [dMF97]. Simulation is of particular importance if non-Markovian traffic models are assumed. Policies optimizing a utility gain are computed and integrated into an agent's knowledge base [MP97]. Other type of policies obey required thresholds of different QoS parameters and provide a set of admissible compromises.

As a particular promising approach, we focus on simulation based neural networks for admission control [FHvBdM97, NdAL93]. Each involved agent decides autonomously on the prospective gain of an additional admission based on a trained neural network taking into account reward or cost for

- each admitted media stream,

- the risk of additional admissions,

- the potential of future admissions and the availability of free resources.

Although reasoning and decision are locally performed by each agent, the agents are coordinated by communicating certain events and internal status information to each other. Bottleneck situations and severe QoS violations are taken into account with high priority by each involved agent, i.e., agents belonging to the same multicast tree. Such events result in higher appreciation of free resources by the agents. Multicast trees should be as balanced as possible, and thus maximizing the sharing effect, for efficiency reasons.

If agents realize an unbalanced situation in the portion of the multicast tree which is assigned to them, they can communicate these situations to other agents and affecting them as a result in their negotiation behavior with user agents. It could, of course, occur that agents communicating unbalanced states be in favor of opposite counter measures. This is taken into account by a weighting factor relating the importance of different agents in a current situation of a multicast tree. The closer an agent to the core router, the higher its importance is.

3.2 How agents interact within the system

The proposed agents structure enables us to realize a distributed QoS management. A distributed management strategy gives some advantage, such as less complexity for each agent (because each of them is used for a specific task), a better knowledge of the local situations and thus a better discovery of the system's problems, the possibility to make different control (correction) actions according to the specific *knowledge* and *capacity* of the agents involved in the operation.

A distributed management strategy can be well carried out using software agents. In fact, agents are already arranged for operating in environments and in a distributed manner, and they permit very flexible communication and co-operation schemes. We will now describe the agents present in the system and their general functions.

In Figure 3 we can notice the presence of various types of agents inside our system, representing different levels of expertise:

- *User Agent*,
- *System Agent*,
- *Network Agent*,
- *Service Agent*.

In Fig. 3 it is indicated that a user on the client system interacts with the QoS management component by means of an appropriate *User Agent*. As we

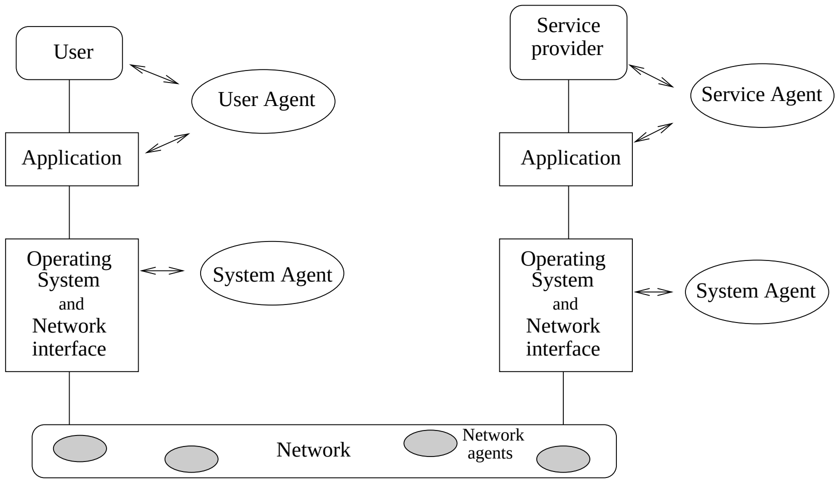

Figure 3: System architecture.

will see, such module is specific for each service (application) required by the user, and enables him to specify and negotiate the desired QoS parameters. It is also the agent which sends the user the request of renegotiation of QoS parameters when the lower level agents consider it as necessary.

A *System Agent* resides on each client and provides all the information about the client's system state. Relying on the services provided by the operating system of the host machine, it has the mechanisms to control the QoS on the client. This agent is the reference module within the client: both the User Agent and the Network Agents, which will be described in the following, rely on it for various operations. From an implementation point of view, the logical functionalities exploited by the System Agent may be seen as independent processes (or agents) concurrently executing on the node. It supports mapping of QoS parameters from one level to another, reservation of hw/sw resources inside the client, and the negotiation of QoS parameters. In addition, it deals with the communication and co-operation with the other agents of the system.

Attached to the network are *Network Agents*, whose goal is to *control* the network; such agents provide the possibility of:

- specifying a set of QoS parameters for a given connection,
- reserving suitable network resources,
- monitoring QoS parameters during existence of a connection,
- starting the corrective activities of QoS adaptation in case some QoS parameters are not respected (through communication and co-operation among agents),
- interaction with the network management system (in order to obtain new resources and/or optimize the use of the existing ones).

On each application server node a *Service Agent* is available which has to:

- monitor whether the QoS parameters (to be given for each service required by the clients) are respected,
- permit the use of a service to the optimal number of users by organizing an efficient use of the available resources,
- keep track of the users logged in and their occupied resources, in order to optimize the sharing of the network resources,
- send to the client an agent that permits to specify the most appropriate QoS parameters for the specific application; this agent will also contain the knowledge of the server's possibilities.

User Agents play a particular role. They are considered to be greedy to a certain extent as to get the service quality which suits them best. Therefore, user agents are designed to compete with the other type of agents. Negotiation and price auction are performed by applying results from *game theory* reflecting behavior as is known from markets [LS97]. Opposed to this, the agents of other types are supposed to cooperate supportively with each other. But note that once negotiation has been completed, *all* agents must respect their own commitments for the whole process to work well.

Additional conventions are needed, such as when to apply service abortion or refunding rules, if some commitments are not met [FHvBdM97]. Further study will be necessary here.

3.3 Specific functions of the agents

A key point of our architecture is built on the possibility to download (from the server) the User Agent related to the application to be run. QoS parameters at the user level depend on the type of application (and thus on the service required). It is the service provider who knows all the characteristics of the application; the provider can therefore include (in an agent) the information related with the combinations of QoS parameters more appropriate to the service required. In this way, the server can better meet the users' requirements, by optimizing, at the same time, the use of network resources. The agent, which has been prepared this way by the service provider, is sent to the client, where it offers the user the opportunity to select QoS parameters. At the client, this agent converts the QoS specified by the user into parameters to be passed to the System Agent.

The System Agent is the module that negotiates the QoS parameters received from the agents on the network and the agents on the server. System Agents have to manage also local resources: they are strictly related to the operating system and operate at the I/O device management level, process scheduling level, memory level, etc. If a request for connection establishment can be satisfied (agents also provide the function of *admission control* at the level on which they operate), then each agent allocates required resources for the application (resource reservation). It is assumed that the underlying network provides the technical prerequisites to implement the required functionalities.

While performing such operations, the System Agent can create other sub-agents, specialized in specific functions. In fact, according to the type of application involved, it is necessary to take into account several parts of the operating system and of the different levels of the network protocol stack. Of course, such agents depend on the hardware and software architecture of the client, and they rely on the services provided by the underlying hardware to perform the required operations of resource reservation. The implementation of such agents therefore depends on the hardware of the machine for which it is designed and on its operating system.

3.4 Cooperation and communication among agents

The network is a central element to be managed when dealing with distributed multimedia applications. The network often limits the QoS of an application. For transmitting videos across a network, high transmission capacity is required and traffic with strict timing constraints is produced. If such constraints are not respected, a degradation in the quality of the service is to be expected. In our architecture, the network is managed and controlled by a set of Network Agents that can be activated and run on several nodes on demand. If control or monitoring on a specific node or group of nodes of the network is required, then the appropriate agents can be sent onto the nodes in question and be run there.

The use of an agent-based infrastructure for the management of QoS is a good choice for implementing several adaptation strategies in which available resources need to be re-arranged in such a way that the user is still satisfied. Such operations, in general, imply an intense (and sometimes complex) exchange of messages among the parties involved. The agents can therefore adequately support such functions and permit more flexibility in doing such operations.

Particularly interesting in performing this kind of operations is exploiting agents' cooperation for optimization purposes. Each agent is able to collect informations about *nearest* agents, which complement the partial knowledge concerning the state of its own resources. As a result, each agent, although being autonomous to a larger extend, is influenced by and can influence the behavior of other agents.

When providing end-to-end QoS, each component involved has to contribute with certain guarantees on its own resources. If a component violates such guarantees, an agent will detect this and a cooperation among involved agents is started in order to reassign resource reservations to different components. Agents whose resources are not fully utilized may reserve additional resources, in order to compensate for the violations of other agents. The agents near the component on which violation occurred initiate a new round of reservation. Information on resources yet available and where to get from is gained through communication with other agents. To yield a global and efficient resource management providing end-to-end guarantees, agents always operate under the constraint of user satisfaction. Since we have no centralized management, agents have to adopt mechanisms for group decision making processes, for negotiation among competing proposals, handling resource conflicts and reaching consensus [GK95].

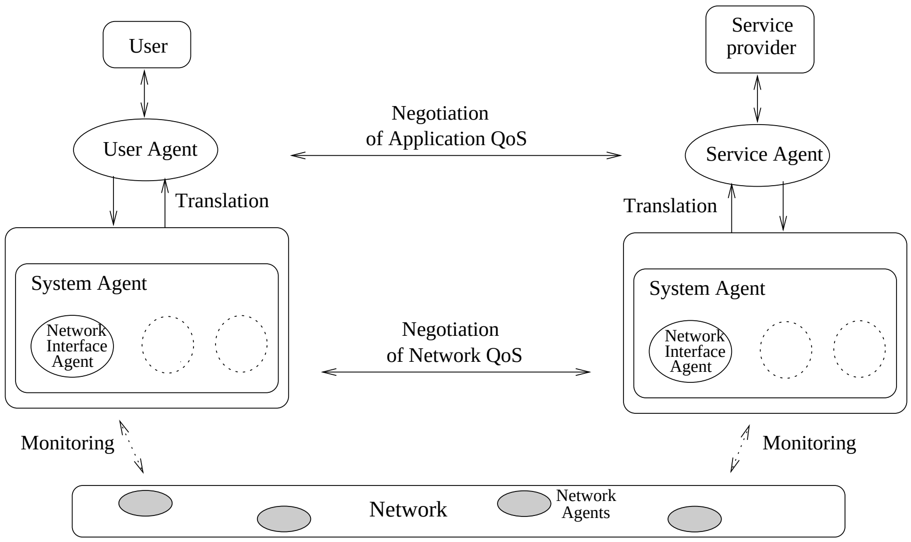

Figure 4: Interactions among various agents.

As we can see in Fig. 4, our architecture offers two kind of interactions between agents:

- horizontal communication between agents of corresponding levels,
- vertical communication between agents of adjacent levels.

Horizontal communication involves negotiation of QoS parameters specific to the corresponding level. For example, the User Agent and the Service Agent negotiate application level QoS parameters, while System Agents operate on network QoS parameters.

Requests for setting-up a connection with a given QoS occur among agents of adjacent levels, thus involving vertical communications. The agent of each level is in charge to translate his parameters into the ones corresponding to the adjacent level (QoS mapping). At the user level, typical parameters to specify the required QoS are related to the description of the reproduction quality at the device level, for example sample size (resulting from height, width and color specification for a video stream) and the media sample rate. Generally, the user has no specific knowledge of the underlying technological constraints. For example, he is usually ignorant of the transmission capacity necessary for a movie, the acceptable delays for a packet of an audio stream, etc. The User Agent will therefore make a preliminary conversion, passing to the System Agent a set of values (range) that represent the level of QoS desired by the user. Typical network parameters are, instead, throughput, end-to-end delay, jitter and loss probability of a packet.

After starting the application and establishing the connection via the network, System Agents together with Network Agents have to monitor the connection state in order to guarantee the respect of the negotiated parameters. If some of these parameters do not match with the negotiated values and if an adaptation phase cannot establish suitable QoS parameters a renegotiation phase is started.

As depicted in Fig. 4, a System Agent always instantiates a process called *Network Interface Agent* which deals with the negotiation of QoS parameters. Through this agent, other monitoring and control activities on the network are started: in fact, this agent communicates with Network Agents on the network nodes. Network Agents refer to the Network Interface Agent when they need to notify critical situations or problems on the network part which affects the node in question.

Depending on the specific application, a System Agent may execute other application specific software modules (agents) which are shown in Fig. 4 as dashed circles.

4 A Multi-Agent Java-based Platform for QoS Management

The reference architecture previously described, requires adequate mechanisms to allow agent migration and execution. Furthermore, agents should be properly executed regardless of the hardware and software platform of the node.

These basic requirements lead us to the adoption of Java as base language to develop our platform. Java is a new language defined by Sun Microsystems whose features make it ideal for the development of heterogeneous network applications [Gos95, Ham96]. With Java it is possible to write applications (*applets*) which, once directly inserted onto Web pages, can be sent to the user's browser simply by a click of the mouse and then executed using local CPU resources. This specific feature of Java has not only made it possible to create Web pages containing animation and sound and other features which were previously impossible, thus making the world of the Web more pleasant; its potential goes far beyond such simple use [You96].

Java is, in fact, a complete programming language which offers all the basic mechanisms for communication and synchronization between processes and guarantees the portability of the code generated across multiple architectures and operating systems in an efficient and secure way.

In this Section we will show how it is possible to use this technology to implement the reference architecture for QoS management previously described. From a more general point of view, we extend the results provided in [PTV97] to define and implement mechanisms for the execution of software agents on any server, wherever the code of interest is physically located. In turn, these software agents can request the execution of new processes, whose code does not necessarily have to be present locally but can be retrieved from remote servers. The approach proposed is highly flexible: assuming all the network nodes to be capable of executing the Java code, a software agent code can be executed on any of them, without any limit due to the presence of different hardware platforms.

According to the proposed approach to QoS management, the main requirements can be summarized as follows:

- to offer the opportunity to instantiate software agents on system nodes without having to re-compile the code,
- to simplify procedures for migration of an agent between two nodes,
- to instantiate a software agent on remote nodes where the application code has not been pre-installed.

The final aim can therefore be summed up as the possibility to activate execution of an agent on any node in the system (even when there are different hardware architectures), without having to worry about the physical location of the code.

The basic concept of the proposed approach can be expressed by the **run** primitive, defined as follows:

$$run(name, < in\_par >, < out\_par >, < execution >, < storage >)$$

where each parameter has the following meaning:

- *name:* identification of the software agent of interest;
- *in_par:* list of input parameters (if any);
- *out_par:* list of output parameters (if any);
- *execution:* identification of the node on which the application is to be executed;
- *storage:* identification of the server on which the application code is to be found.

If only the *name* parameter is present, the *run* primitive executes the *name* agent on the same node, expecting to find the code locally. If the *storage* parameter is provided as well as the *name*, the agent will be executed locally but its code will be retrieved from the *storage* server. Finally, if the *execution* parameter is given, the agent will be executed on the node specified. In its most complete form, the *run* primitive executes the *name* agent, which is physically located on the *storage* node, on the *execution* machine, providing the *in_par* input parameters and expecting the *out_par* output values.

To make execution of this primitive completely transparent to the user, the whole platform has been developed using the Java language. On each node of the platform, therefore, there has to be a *Java Virtual Machine* to allow Java applications to be executed without worrying about the underlying architecture. The classes making up the various applications are thus stored and transferred in *bytecode*.

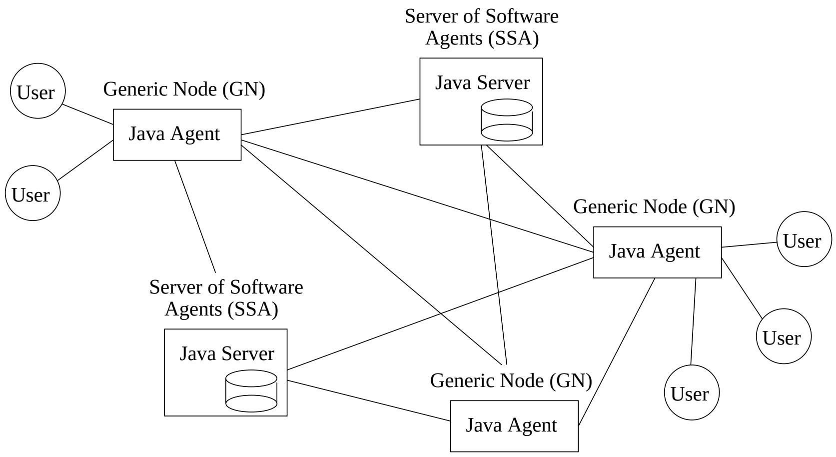

Figure 5: System architecture.

We assume that the system contains centers, named *Servers of Software Agents* (*SSA*), to collect agents' codes which play a role equivalent to that of the mass storage subsystems in a traditional computer system. In general, however, nothing prevents a *Generic Node* (*GN*) from simultaneously having the role of node to be monitored and being SSA: it can, in fact, both store and execute software agents.

To gain a better understanding of how it is possible to implement the functions described, the following sections give a complete description of the software modules installed in the GNs and SSAs and the relative forms of interaction.

Generic Node

The following software modules have to be present on a Generic Node:

- *Run_App*: this module allows the user to use the services provided by the platform. Run_App defines the *run* method which implements the *run* primitive described above. Thanks to the run method, Run_App makes it possible to execute, locally or remotely, an agent whose code is on the same GN or on an SSA.

- *Java Agent (JA)*: this is a process which is always executed on the node. It is a daemon due to whose presence on the node the latter is characterized as belonging to the platform. The JA manages all interaction with GNs and SSAs and is the environment in which the subsequent software agents are executed.

  The JA module uses the *NetworkClassLoader* class which implements the mechanisms required to load software agents and their constituent classes from remote nodes, dynamically instantiating them inside the current application. This class derives from the Java default ClassLoader class: it redefines the *loadClass* method normally used to load classes. Use of our NetworkClassLoader allows classes to be loaded from the network instead of from the local file system: it contains all the mechanisms needed to transfer the bytecode of the classes from the SSA servers and to convert them into objects ready to be executed in the application requesting them. We should emphasize that once the main class of an agent has been loaded, with explicit reference to the NetworkClassLoader, subsequent references to other classes in the agent are solved automatically and in a transparent way.

If the *run* method is called making explicit reference to the *storage* field, *Run_App* instantiates the NetworkClass Loader with this parameter: from now on any request for class loading will automatically be sent to the SSA where the application (and its constituent classes) physically resides. The *NetworkClassLoader* is provided with a *cache mechanism* by which memory of classes already used is available, so as to avoid asking the SSA for them whenever an instance is created.

If, on the other hand, the *storage* field is not given when the *run* method is called, *Run_App* will make reference to the default ClassLoader and all the class loading requests will be met locally.

Server of Software Agent

In an SSA the following modules are present:

- *Java Server (JS)*: this is a process always executed on the node (*daemon*), which intercepts and manages all the requests for class and software agent loading from the GNs.

- *App_DB (DB)*: this implements the database of software agents (stored in the *bytecode* format). Each software agent has the following list of associated fields:

  - *Name*: this identifies the agent;
  - *Descr*: this gives a concise description of the functions performed by the software agent;
  - *Node_type*: this indicates the type of node on which the agent can be executed. It should be pointed out that due to the approach adopted the software agents can be executed on any node equipped with a Java Virtual Machine. However, an agent can make reference to specific functions not present on all the nodes.

For example, an agent monitoring the number of users connected can be launched on a multi-user machine but not on a PC.

– *in_par* and *out_par*: list of input and output parameters.

The options offered by the platform can be summarized as follows:

- *Case 0:* $run(name, < in\_par >, < out\_par >, *, *)$

  Local execution of an agent whose code is stored on the same node.

- *Case 1:* $run(name, < in\_par >, < out\_par >, *, < storage >)$

  Execution on the local node (GN) of an agent whose code does not reside on the same node but on a different SSA.

- *Case 2:* $run(name, < in\_par >, < out\_par >, < execution >, < storage >)$

  Opportunity to request the execution, on a certain node (GN2), other than the node where the management action is taking place (GN1), of an agent whose code is stored on a different node (SSA server).

- *Case 3:* $run(name, < in\_par >, < out\_par >, < execution >, *)$

  Execution of a locally available agent (GN1) on a different node (GN2).

5 Case Study

In this Section we present an application example of agent-based QoS management according to the approach proposed in the previous sections. The system under exam represents a cooperative working environment constituted of N multimedia workstations communicating through a CSMA/CD protocol. Each workstation (WS) is running an experimental software which allows an easy interaction with the other workstations by providing the following features:

- it captures and transmits a video stream from the camera the WS is equipped with,

- it manages an audio stream,
- it allows the user to open data connections for e-mail, ftp and telnet services.

Thus each WS generates two flows of data: one with fixed real-time requirements for audio-video transmission and one for best-effort data traffic connections, without stringent constrains. Video/audio streams generated by each user are transmitted in multicast to all the other users, while data transfer occurs only with a given destination. The users constitute a closed working group, meaning that only transmissions inside the cooperative working environment are allowed. Each user is assigned a given amount of transmission capacity for video/data transmission which is guaranteed for the whole working session. The transmission capacity for data traffic, named *available capacity* (AC) is the one remaining after all the video/audio connections have been satisfied and is dynamically allocated and usually equally distributed among all the users. From a logical point of view the reference architecture can be depicted as in Fig. 6.

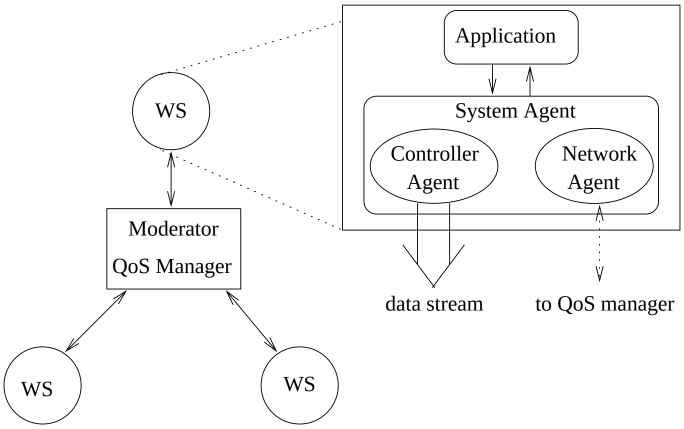

Figure 6: A Case Study.

When a user joins the group, the system agent immediately executes two agents: a *network agent* and a *controller agent*. These two agents are transferred into the WS and executed using the services provided by the platform. Thus, for this specific application, their permanent presence on the node is not required; they are loaded from the SSA of the local domain and then dynamically executed.

The network interface agent has the task to monitor the amount of data traffic generated by the user, while the controller agent is in charge to eventually limit the data traffic if it has exceeded a given threshold. The threshold value is not fixed, but time-dependent. The overall activity of the working group is in fact monitored by a third agent which plays the role of moderator and operates in order to equally distribute the transmission capacity for data traffic among all the users. The *moderator agent* is an instance of a network agent (Fig. 4) which, for the case into exam, is executed on one of the WS taking part to the cooperative working group, but a dedicated WS can eventually be used for this purpose. The moderator agent is notified by other network agents devoted to system monitoring on the amount of transmission capacity utilized by each user. It is not the moderator that has to poll each node in order to determine the transmission capacity used; it's the network agent executing on each WS that informs the moderator whenever a change on the transmission capacity usage occurs. The controller agent is responsible for the access to the network for the application. The values of the currently utilized capacity for each user are stored in a table managed by the moderator.

For illustration, let us introduce the concept of *reserved capacity* (RC) which represents the average value of available capacity assigned to each connection which is defined as:

$$RC = \frac{AC}{N}$$

, where AC is the available transmission capacity and N is the total number of multimedia connections. A user is generally allowed to exceed his quota RC temporarily by taking advantage of unused but reserved capacity of other connections. If the available transmission capacity has been fully utilized, a QoS degradation takes place, thus adapting to the shortage in a controlled manner as far as possible.

When a connection decreases the amount of capacity used (e.g. a file transfer ends), the network agent informs the moderator, who updates the table of current capacity usage and sets the resources free for further use through other demanding clients.

The moderator agent plays a crucial role in the system: it receives information from the network agents running in each WS and, according to the overall capacity reservation pattern, executes balancing actions, thereby adequately motivating the controller agents to reduce/increase the amount of data traffic being generated in the WS.

In implementing this service, two lines of investigation were carried out: the former aiming to evaluate the cost of the decentralized approach we propose against the more traditional centralized one; the latter to verify and evaluate how the proposed approach can really be implemented to deal with a practical case study.

In the examined case study, the information required by the moderator for performing its function was quite simple. That is, it doesn't require high computational effort on each node. Anyway, a more complex application may require more complex information to be retrieved from remote nodes. In a centralized architecture, the moderator should poll each node and then perform locally all the necessary calculations; using our approach, the computations are delegated to the agents on each node minimizing the traffic on the network and reducing the workload on the central machine.

Use of the platform presented, however, does have a cost in terms of utilization of the computational resources of the network nodes, as a consequence of the remote execution of management and control functions.

The agents which always have to be present on the platform nodes (Java Agents and Java Servers) execute in the background and do not consume CPU cycles when the application is not running. As with any other daemon process, the only resource they consume in *sleep* periods is virtual memory. They only become active when requested to load and execute a remote agent. Significant measures for the processes involved in our platform are given in Table 1. They refer to a Sun SparcStation with the Solaris 2.4

operating system. The first column indicates CPU utilization, the second the percentage of system memory used by the process, the third the resident set size and the fourth the total size of the process.

	CPU%	MEM%	RSS (KB)	SIZE (KB)
Java Agent	0.0	2.3	≈ 700	≈ 2800
Java Server	0.0	5.3	≈ 1600	≈ 2800
Application	$10 \div 30$	$10 \div 15$	≈ 4500	≈ 5500

Table 1: Calculation Resources Used by Processes.

Although the bytecode of the various modules is a few Kbytes (8 KB for the Java Server, 3 KB for the Java Agent + 5 KB for the NetworkClass-Loader), during execution it is necessary to take into account the size of the Java interpreter and the relative run-time system. When an application is being executed on a node, the CPU workload fluctuates with peaks reaching 20-30% utilization; the size in memory requirement grows according to the type of application, but is never very high and not in the with common applications.

In addition, subsequent applications on the nodes will execute with the same interpreter, that is, a new application will be executed as a new thread in the same process. This means that subsequent applications will not require execution of a new interpreter instance, thus reducing the amount of calculation resources consumed.

6 Conclusions

We have presented an infrastructure for distributed QoS management based on software agents. We discussed distributed agents, their properties and the main fields in which their application is promising. After identifying the main functions needed in a system for QoS management, we described our architecture, the role and the function played by each kind of agent.

We motivated the effectiveness of our approach in terms of the cognitive capabilities of agent schemes. Although many aspects need further study, we are convinced that our agent-based approach is the way to go for large scale, wide area distributed QoS management. The solution of the QoS management problem represents one of the key issues of the forthcoming multimedia capable information highway.

We gave a detailed description of our implementation platform which provides the mechanisms for distributing and controlling agents' executions in the network nodes. This platform has been entirely developed using the Java language which ensures full portability among different hardware and software platforms. Java seems also to be the method of choice supporting future programmable networks, i.e., open network design. This relates also to the planned extension to use the platform as an environment for migration of software agents among different network nodes to control the quality of multimedia applications.

To provide formal support of agent decisions, simulation can be applied in order to evaluate and control the overall system behavior, thus minimizing occurrence of oscillation effects and reducing communication overhead. Thresholds and granularity of time scale will also have to be determined to ensure system well-behavior.

References

[ACH97] C. Aurrecoechea, A. Campbell, and L. Hauw. A Survey of QoS Architectures. *Multimedia Systems Journal,* Special Issue on QoS Architectures, 1997.

[dMF97] H. de Meer and S. Fischer. Controlled Stochastic Petri Nets for Multimedia QoS Management. *Proc. 9. GI/ITG Conf. Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, MMB97*, September 1997.

[EW95] O. Etzioni and D.S. Weld. Intelligent agents on the Internet: Fact, Fiction, and Forecast. *IEEE Expert*, 10(4):44–49, August 1995.

[FHvBdM97] S. Fischer, A. Hafid, G. v. Bochmann, and H. de Meer. Cooperative QoS Management for Multimedia Applications. In *4th IEEE Int. Conf. on Multimedia Computing and Systems*, pages 303–310, Ottawa, Canada, June 1997. IEEE Computer Society Press.

[GK94] M. Genesereth and S. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[GK95] B.J. Grosz and S. Kraus. Collaborative plans for complex group action. *Technical Report TR-2-95, Harvard Univ., CRCT*, 1995.

[Gos95] J. Gosling. The Java Language Environment: a White Paper. Technical report, Sun Microsystems, May 1995.

[GY95] G. Goldszmidt and Y. Yemini. Distributed Management by Delegation. In *Proc. of the 15th International Conference on Distributed Computing Systems*, Vancouver, British Columbia, June 1995.

[Ham96] M. A. Hamilton. Java and the Shift to Net-Centric Computing. *IEEE Computer*, 29(8):31–39, August 1996.

[Jun96] J.I. Jung. Translation of user's QoS requirements into ATM performance parameters in B-ISDN. *Computer Networks and ISDN Systems*, 28:1753–1767, 1996.

[KdM97] H. Knoche and H. de Meer. Quantitative QoS Mapping: A Unifying Approach. In *Proc. of the 5th Int. Workshop on Quality of Service (IWQoS97)*, pages 347–358, New York, May 1997.

[Laz97] A. Lazar. Programming telecommunication networks. In *Proc. of the 5th Int. Workshop on Quality of Service (IWQoS97)*, pages 3–22, New York, May 1997.

[LS97] A. Lazar and N. Semret. Auctions for Network Resource Sharing. Technical Report 468-97-02, Center for Telecommunication Research, Columbia University, January 1997.

[MP97] P. Maryni and G. Pacifici. Real-time estimation of the link capacity in multimedia networks. In *Proc. of the 5th Int. Workshop on Quality of Service (IWQoS97)*, pages 101–112, New York, May 1997.

[NdAL93] J. E. Neves, L.B. de Almeida, and M.J. Leitao. Atm call control by neural networks. *Proc. 1st. Intern. Workshop on Applications of Neural Networks to Telecommunication*, 1993.

[PTV97] A. Puliafito, O. Tomarchio, and L. Vita. A Java-based Distributed Network Management Architecture. *Third International Conference on Computer Science and Informatics, CS&I'97*, March 1997.

[RdM97] J.P. Richter and H. de Meer. Towards Formal Semantics for QoS Support. Technical Report FBI-HH-B-198/97, Department of Computer Science - TKRN, Universityof Hamburg, Germany, June 1997.

[SN95] R. Steinmetz and K. Nahrstedt. *Multimedia: Computing, Communications and Applications*. Prentice-Hall, 1995.

[You96] E. Yourdon. Java, the Web, and Software Development. *IEEE Computer*, 29(8):25–30, August 1996.