Designing a Videoconference System for Active Networks

Mario Baldi, Gian Pietro Picco, and Fulvio Risso

Dip. Automatica e Informatica, Politecnico di Torino
C.so Duca degli Abruzzi 24, 10129 Torino, Italy
Phone: +39 11 564 7067 Fax: +39 11 564 7099
{mbaldi,picco,risso}@polito.it

Abstract. Active networks are receiving increasing attention due to their promises of great flexibility in tailoring services to applications. This capability stems from the exploitation of network devices whose behavior can be changed dynamically by applications, possibly using technologies and architectures originally conceived for mobile code systems.
Notwithstanding the promises of active networks, real-world applications that clearly benefit by them are still missing. In this work we describe the design of a videoconference system conceived expressly for operation over active networks. The goal of this activity is to pinpoint the benefits that mobile code and active networks bring in this application domain and to provide insights for the exploitation of these concepts in other application domains.

1 Introduction

The role of computer networks is becoming increasingly important in modern computing. This fact poses unprecedented challenges in terms of performance and flexibility which affect the protocols and standards that constitute the communication infrastructure underlying computer networks, as well as the technologies and methodologies used to build distributed applications.

Researchers are devising approaches coming from different perspectives and addressing different layers of abstraction. However, a set of approaches that exploit some form of *code mobility* is currently emerging among the others as particularly promising and intellectually stimulating. Code mobility can be defined informally as the capability to change dynamically the bindings between the code fragments belonging to a distributed application and the location where they are executed. The rationale for code mobility is to reduce network traffic and increase flexibility and customizability by bringing the knowledge embedded in the code close to the resources [5]. This powerful concept is being popularized by a new generation of programming languages and systems that provide abstractions and mechanisms geared towards the task of relocating code. These technologies, often referred to collectively as *mobile code systems*, are targeted at the development of applications on large scale distributed systems like the Internet.

Concurrently with these developments, other researchers are investigating means to introduce flexibility in computer networks by assuming the availability of next-generation network devices whose behavior can be changed dynamically according to users' needs. These *active networks* are receiving a great deal of attention in industry and academia and seem naturally suited to leverage off of the developments in the field of code mobility. In fact, technologies and architectures conceived for building distributed applications exploiting code mobility can be used for supplying dynamically the network devices with application-dependent code that changes their behavior. In this scenario, networks become active because they take part in the computation performed by the applications rather than being concerned only with the transfer of data.

Researchers presently interpret the idea of active network at least with two different nuances. The first, broader interpretation of the term is that the distinction between network nodes and end-systems becomes blurred in terms of functional characteristics. This approach is embodied for example in the work described in [6] and [18]. In this setting, network devices, e.g., routers, can execute mobile code implementing a distributed application which can benefit by direct access to functionality and information at lower layers in the network stack. This *programmable switch approach* [16] does not affect the way current lower and mid layer protocols are designed and deployed. On the other hand, the second interpretation aims at modifying the heart of network protocols, by extending the control information contained in network packets with code describing how to process the packets at the intermediate nodes along the path to the destination. Protocol deployment is then performed on demand, without requiring software preinstallation and upgrade, and yet is under the control of the applications. A packet augmented in this way, also called *capsule* [16], is reminiscent of what the mobile code community calls a mobile agent—an autonomous unit of mobility containing code and state. Toolkits are being developed [17, 7] to support the creation and deployment of capsules.

Active networks and, more generally, code mobility are promising ideas. Nevertheless, despite the great deal of interest and effort in these research areas, contributions that characterize precisely and possibly quantitatively the advantages of the approach are only beginning to appear [11, 2], while applications that demonstrate these advantages in real-world domains are still largely missing.

The goal of the research described in this paper is to assess the benefits brought by code mobility in the context of active networks. This is achieved pragmatically by constructing a videoconference system according to the aforementioned programmable switch approach. Videoconference is an increasingly popular distributed application which poses significant challenges and constraints and thus can be considered a reasonable testbed for our purposes. Our long term goal is to characterize qualitatively and quantitatively the implementation currently being completed at our university against conventional ones, in order to identify precisely the tradeoffs involved. However, the work presented here focuses on the *design* of our videoconference application and aims at identifying and suggesting novel architectural opportunities enabled by code mobility and active networks.

The paper is structured as follows. Section 2 discusses briefly the requirements of a videoconference system and how these can be satisfied by an architecture that exploits mobile code on an active network. Section 3 describes in detail the architecture of our prototype, and identifies two variants which feature different degrees of distribution. Section 4 provides information about the ongoing implementation. Finally, Section 5 contains brief conclusive remarks and discusses future lines of research on the subject of this paper.

2 Videoconferencing on an Active Network

Videoconference systems can be split grossly in two categories. In *peer-to-peer* conferencing systems, the participants are connected through a multicast network and the videoconference flow generated by each participant is distributed to the others exploiting multicast delivery. An evident advantage of this first approach is its good scalability. However, conference management is complicated by the characteristics of the architecture, especially as far as security is concerned. A relevant example of this category is the tool suite for MBone, the Multicast Backbone [1].

The alternative, popular especially among commercial systems, employs a *centralized* architecture based on a conference server which receives the conference streams from the clients operated by the participants and replicates such streams back to all the clients connected. Centralization not only simplifies the problem of secure access, but also enables customizability. The server can perform additional computation on an incoming videoconference stream on behalf of the clients instead of simply replicating the stream towards them, thus enabling conference users to get control on the delivered quality of service. A centralized approach is affected by the usual drawbacks in terms of scalability and fault-tolerance. An example of centralized system is CuSeeMe [4], while the H.323 standard [8] defines an architecture that accommodates both categories.

Neither of the two solutions described seems suitable for the large scale scenarios that are being envisioned for the Internet. Let us consider for instance the broadcasting of a sport event. In a multicast network, the conference streams are not tailored to the users: the same stream is routed to all the audience members, no matter whether they are connected through a high-bandwidth local area network or a dial-up connection. Thus, the former get a quality lower than their potential, while the latter get unpredictable quality due to the packets discarded to tailor the rate of the data flow to their low capacity access. On the other hand, a centralized architecture requires a huge amount of computational power in the conference server and overloads the network since separate multimedia streams are maintained through the network between the server and each client.

Ideally, a videoconference system should exhibit features coming from both architectures, thus enabling user customization without preventing the scalability of the architecture. Interestingly, the latter is, by and large, one of the goals that motivated active networks and in general the approaches relying on code mobility. Thus, their exploitation in solving the aforementioned problems seems a natural step.

Our architecture is based on a conference server that we call *reflector*. Customizability and scalability are then provided together by:

1. enabling the users to "upload" application code into the reflector, thus changing its behavior and *customizing* it to their needs;
2. running the reflector on the intermediate nodes of the network, where it can use the information managed by the device to become aware of the status of the network and *adapt* to it;
3. enabling the reflector to *migrate* on a different node as a consequence to adaptation.

The first point provides a degree of customizability even higher than the one provided by centralized systems. Customization is not limited to changing the parameters of the reflector, rather it allows to change the code that governs its behavior. The reflector is then basically a "shell" where each client can plug-in dynamically the code describing some customized processing. Thus, for instance:

– Different coding algorithms can coexist in a flexible way. The reflector does not need to be equipped with plenty of coding algorithms in order to encompass the needs of a wide range of clients. These are linked in the reflector dynamically and on demand.
– The quality of service for the encoded videoconference stream can be changed dynamically to fit user needs. The desired policy can be arbitrarily complex and can be changed at run-time. For instance, it is possible to specify application-dependent criteria to discard packets in presence of congestion, as suggested in [3].
– The videoconference flows coming to a client from different participants can be treated differently. For example, a participant can request the reflector to give higher priority or to guarantee a higher quality of service to some flows, and carry the others with a lower quality. The advantage is that flows can be discriminated according to application level information, like the identity of the current speaker.

Point two, that actually provides the rationale for exploiting an active network, deserves some elaboration. At the time of writing, network devices equipped with a run-time support for execution of mobile code are not widely available. However, many vendors have already announced new releases of their products that feature hardware or software support for the Java language [14]. In the implementation of our prototype we cope with this problem, characteristic of the whole research area, by adopting the approach followed by many researchers: we simulate an active device by running the mobile code on a workstation directly connected to a network device.

Point three is at the core of the work described in this paper. The reflector analyzes constantly the data available on the network device and can trigger a

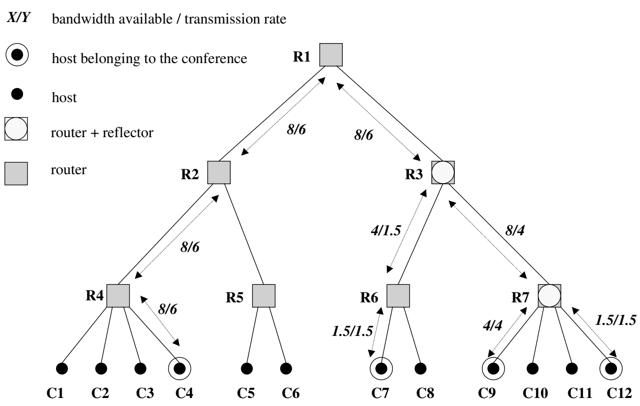

Fig. 1. Cloning reflectors.

migration to adapt to some change in the conditions of the network. We devised two architectural solutions to enable adaptivity through migration. In both cases, the criteria that rule adaptation are embodied in the code of a component of the reflector that is designed to be modular and interchangeable, as described in the next section.

In the first solution, the reflector responds by *relocating* itself in a position of the network that is optimal with respect to some cost function, e.g., distance from the clients. According to the classification presented in [5], the reflector relies on a weak form of mobility where its execution state is not preserved across migration: only its code and a portion of its data state are transferred. This solution is suitable for conferences characterized by a limited number of participants or by heavily clustered participants.

In turn, our second architectural solution is conceived for conferences with a great number of participants. It relies on *cloning* the reflector rather than migrating it; the latter can be regarded as a special case of cloning where the original reflector is terminated. This way, multiple reflectors are injected into the network upon some special event (e.g., when a new participant joins a session), and perform subsequent transformations of the conference streams providing for improved scalability. For instance, Fig. 1 shows a network with two reflectors placed on two different routers, `R3` and `R7`, that are responsible for serving hosts `C4`, `C7`, `C9`, and `C12`. Notably, the reflectors have been cloned only in the positions of the network where transformation of the videoconference streams is needed. Moreover, whenever possible only one stream is transmitted between reflectors and separate streams are generated close to the participants, thus minimizing the overall network traffic generated by the conference.

The next section illustrates the details of both architectures. In principle, the two architectures can be combined effectively in a single system that, upon the occurrence of some user-specified condition, is capable to switch automatically between the two modes of operation.

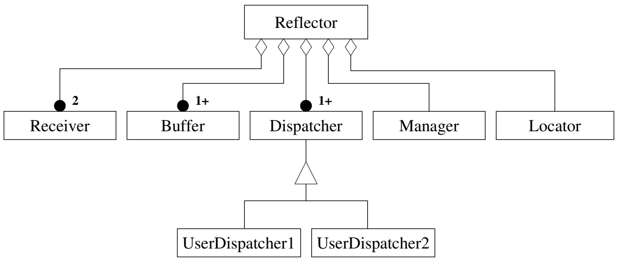

Fig. 2. The components of the reflector.

3 The Architecture

The design described in this section strives for modularity and reconfigurability, in order to leverage off of the opportunities provided by code mobility in changing dynamically the code associated with a component. We specified the design of our prototype using the OMT [12] object-oriented notation. The full OMT design is available in [9].

3.1 The Reflector

The reflector is composed of five classes, depicted in Fig. 2. Receiver, Buffer, and Dispatcher provide the "work power" of the reflector and deal with receiving and transmitting the data packets that constitute the conference streams. The classes Manager and Locator are the "brain" of the reflector. The Manager object is the control component that governs the behavior of the reflector. The Locator object monitors information available on the device the reflector is residing on and can signal to the corresponding Manager object the need for migration or cloning in order to adapt to events in the network. All the aforementioned components are described in the following.

Receiver Instances of this class are responsible for handling the input videoconference streams. Each instance of Receiver contains a separate thread of control that receives the multimedia streams from all the clients. The receiver stores the packets in the element of a buffer pool corresponding to the sender of the packet, identified using information in the protocol headers. The audio and video components are handled as two separate streams which are received on two distinct sockets; thus, two instances of Receiver are spawned in each reflector. The receiver is a key element as far as performance is concerned and keeping it simple is desirable to achieve good performance.

Buffer Instances of this class contain circular lists of packets. The number of buffers contained in the reflector is usually twice the number of clients currently connected to it, in order to separate the processing of audio streams from the one for video streams. Each Buffer object is accessed concurrently by Dispatcher instances, that can read the packets stored in it by the Receiver object in charge of the buffer. Each buffer provides methods to return the packet stored at a given position as well as to determine the index of the packet most recently stored. Clearly, the length of the buffer is a key parameter in the configuration of the reflector, and depends on the bit rate and maximum delay tolerated by each client.

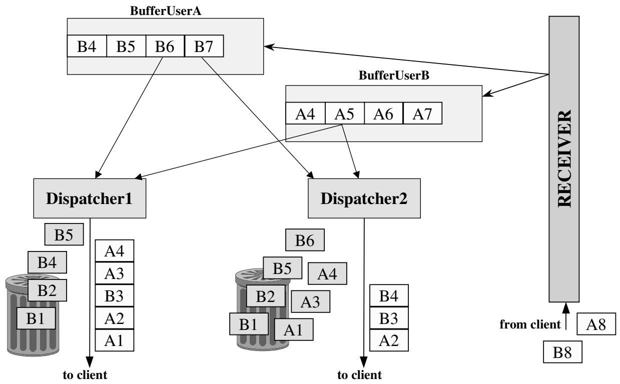

Fig. 3. Exploiting customized Dispatcher objects.

Dispatcher Instances of Dispatcher have their own thread of control and are responsible for the processing that transforms input streams and replicates them to all the clients. The class Dispatcher, in Fig. 2, implements a default behavior that always retransmits the most recent packets in the audio and video buffers. Clients can supply their own classes specialized from Dispatcher, in order to benefit by additional, customized processing and possibly adapt to the network conditions. The selection of a customized Dispatcher object may take place at setup time, but it can also be performed during the conference, by exploiting mobile code mechanisms such as remote dynamic linking. The customized processing embodied in a Dispatcher object may be as simple as an application-aware discarding of frames in order to adapt to network load, or as complex as a recoding of the stream, e.g., to convert it from MPEG to H.261. This scheme can exploit effectively layered encoded video because in presence of bandwidth reduction, e.g. due to congestion, the dispatcher can send to clients only the high priority layers and discard the others with little computation overhead.

Figure 3 illustrates how customized Dispatcher objects can filter out differently the same conference streams. In the figure, the leftmost Dispatcher object privileges the stream coming from client A, while the rightmost treats both streams the same way, although with reduced bandwidth.

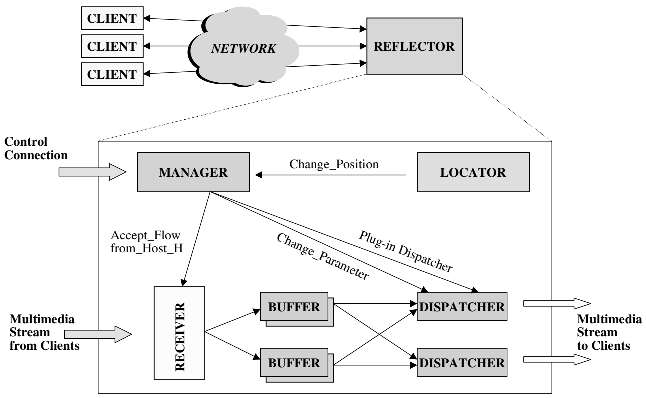

Fig. 4. Managing the videoconference.

Locator The object of this class has a separate thread of control and is responsible for gathering information on the status of the network. It can react to a change in network conditions by suggesting either migration or cloning of the reflector. The new location chosen is supposed to optimize a given cost function, and is communicated to the Manager object that triggers the actual relocation.

A cost function could, for instance, minimize a weighted combination of relevant indexes like traffic on selected links, overall traffic on the network, maximum or average delay experienced by conference participants, and economic cost. The data used by the locator to take its decision can be obtained either by monitoring the network or directly from the databases of the network node on which it is running. In the first case, the locator could probe for example the round trip delay of the path to clients using the `ping` mechanism. In the second case the locator exploits directly the information contained in the router on which it is executing, e.g., the status of its queues, the traffic statistics of its interfaces, or the topological database of a routing protocol.

The amount of intelligence and computational complexity embedded in the Locator are implementation dependent. Customized Locator classes can be created to cope with different videoconferencing scenarios, and substituted to the default one. Also, the locator can be reconfigurable by applications and users that can tune its parameters and thresholds to better suit their needs. In particular, the strategy for relocation, i.e. migration vs. cloning, can be specified dynamically.

Manager As shown in Fig. 4, the instance of Manager coordinates the operations of the reflector to which it is associated with and is in charge of communications with clients. Typical tasks of the Manager include joining new members to the conference, accepting incoming code for the dispatchers, dealing with security issues, and taking care of the migration or cloning of the reflector.

The manager handles control communications with the clients through a control connection for sending feedback about the operations of the reflector and receiving requests, e.g., to change transmission parameters. These requests are interpreted and then satisfied by invoking the appropriate methods on the

Dispatcher. The control connection is used, in particular, to inform the clients upon relocation of the reflector so that they can redirect their streams to the new location, as described in the next section.

3.2 Migrating the Reflector

The movement of the reflector should be as transparent as possible to users, to avoid even temporary service disruptions. This is especially important if the locator component is aggressive and movements are frequent. Temporary service disruption could take place if there is a time frame in which the reflector is no longer operational in the old location and is not yet operational in the new one. In order to avoid this situation, migration takes place in two phases:

1. The manager object generates a clone of the reflector which is sent on the network node chosen by the locator. As soon as the new reflector begins its execution on the hosting node, it sends a notification to the manager of the old reflector, which in turn informs the clients about the new location of the reflector. After the clients receive this notification they start transmitting their streams to the new location. The execution state of the reflector is lost, and only a portion of the data space is carried with it. In particular, the buffers are not transferred at destination, as they contain volatile information that is better handled by the old reflector. In contrast, the client profiles are transferred with the reflector clone, in order to maintain the information about the clients currently connected.
2. When all the clients have redirected their flows to the new reflector, the latter sends a control message to the Manager object of the original reflector, which terminates the process running at the old location.

3.3 Using a Capsule to Join the Conference

The minimum setup for a conference includes an instance of Reflector and two clients. Each clients receives the conference streams from the reflector. However, clients can be either *active* or *passive* depending on whether they originate their own streams or not. The startup of a conference session is determined by a client that "rings" another one, the *conference owner*. The identity of the conference owner must be known a priori through some sort of off-line announcement[1]. The conference is associated with an access control list that identifies the conditions under which a client can be allowed to join a conference. If the calling client matches the access control list, the conference owner injects a reflector in the network and the conference begins. Other participants can join the conference in a similar fashion by calling the conference owner.

[1] As an example, consider MBone conference announcements which are distributed through `sdr` and contain the multicast address of the conference. In the framework of our conference system the announcement would contain the address of the conference owner.

This approach does not scale to large distribution events like an Internet TV broadcast, because the conference owner would be overwhelmed by join requests. Also, the component that keeps the conference access control list and that should be responsible for handling new join requests is actually the reflector. However, the difficulty is that the position of the reflector is changing over time and its address cannot be bound to the conference announcement.

A solution to this problem involves the use of a capsule, i.e. an active packet that contains a join request for the client that created it and travels autonomously towards the conference owner, thus actually implementing a mobile agent. At each node, the capsule object checks whether a reflector is running there, and possibly hands the join request to it. The reflector handles the join request by matching it against the access control list and and notifies the sender of the capsule about acceptance or rejection of the request. The capsule contains also additional information about the client needed to setup the connection, like the capability to provide a customized dispatcher.

In case the capsule object does not run into the reflector on its way to the conference owner, this is eventually reached and forwards the capsule to a reflector, whose position is known to the conference owner through its control connection. In case the conference has not yet started and no Reflector instance is executing in the network, the request is handled directly by the conference owner as described above. Sending the join request as a capsule is particularly effective when the cloning approach is exploited. The existence of many reflectors and their displacement increase the probability that the capsule runs into one of them and consequently the join request is handled closer to the client that issued it. The approach can be further extended by allowing a joining client to send the capsule to other conference participants besides the owner. This increases the reliability of the system since the possibility of joining the conference is not conditioned by the status of the conference owner.

4 The Prototype

A prototype of the videoconference system described here is being implemented at our university using the Java language. The support for code mobility is provided by the μCode toolkit developed in parallel by one of the authors [10], which provides a flexible mobile code infrastructure with limited overhead.

The implementation of the client relies on the public domain tools `vic` and `vat` to generate respectively the video and audio streams. The streams playback is performed by an application developed using the specialized API of the Java Media Framework (JMF) [15]. In addition, a console written in Java allows the user to communicate with the reflector and manipulate the parameters of the session. A new version of the client integrating the console with the visualization and playback of the conference streams will be implemented as soon as standard capture and encoding support are made available within the JMF. Authentication and access control features are currently not implemented, as we decided to focus on the assessment of the impact of customizability and mobility rather than security.

5 Discussion and Further Work

The work reported here investigates the opportunities opened by code mobility in the context of active networks. The paper focuses on design aspects, describing an original architecture for a videoconference system based on a conference server that migrates or clones itself in order to adapt to events in the network. The proposed architecture provides improved scalability and allows clients to customize the server's behavior by exploiting mobile code.

Our work is inspired by the work described in [11], which describes the implementation of a chat server that migrates to adapt to the position of the participants. In that work, the focus is on the optimization achieved by network-aware positioning of the server with respect to a traditional fixed displacement. In our work the potential of code mobility and active networks has been exploited also in terms of user customization which can be strategic in the videoconference application domain. Also, the aforementioned work is mainly concerned with mechanisms needed to probe and monitor resources. In our work, we focus on the exploitation of code mobility; however, devising and comparing different strategies for relocation to be embedded in the Locator component of the system is the subject of ongoing research. This encompasses individuating and defining which mechanisms and information must be provided at the application level and which information can be assumed to be provided by the active network device—still an open issue in active network research.

The performance of a videoconference system depends strongly on the real-time properties of the transmission services it is based on. This work has not tackled issues related to the provision of quality of service guarantees over an active network and to locator mobility in such a scenario (e.g., the impact on resource reservation of the reflector position changing over time). These topics represent an interesting and broad area for future studies.

Another open issue is a better integration of the Real-time Transport Protocol (RTP) [13] in our prototype. RTP is presently the protocol for multimedia transport most widely used on the Internet, and is engineered for conferences whose attendance ranges from a few clients up to 10,000; more clients cannot be managed due to implosion of control messages. In fact, each participant sends periodically to all the others a report containing its identity and possibly information about the quality of the stream it receives. This information can be used by the sources as a feedback of the quality of service perceived by the receiver. Clearly, this solution is impractical for a very large conference. In the cloning variant of our architecture, control message implosion is reduced because each reflector acts as a virtual participant for other reflectors, and thus "hides" the clients directly attached to it at the same time providing direct feedback about the quality of transmission.

Finally, we are in the process of analyzing quantitatively the benefits and the tradeoffs of our design, along the lines of [2]. This encompasses the definition of an analytical model for comparing the performance (e.g., in terms of traffic or latency) of our architecture with respect to conventional solutions, as well as the validation of such a model through direct measurements on the actual implementation.

Acknowledgments This work has been partially supported by Centro Studi e Laboratori Telecomunicazioni S.p.A. (CSELT), Italy. The authors wish to thank Valerio Malenchino from CSELT for his insightful comments during the development of the work described in this paper. Also, the authors wish to thank Margarita Millet Sorolla for her work on the implementation of the prototype.

References

1. The MBone Information Web. Web page `http://www.mbone.com`.
2. M. Baldi and G.P. Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In *Proc. of the 20th Int. Conf. on Software Engineering*, pages 146–155, April 1998.
3. S. Bhattacharjee, K. Calvert, and E. Zegura. On Active Networking and Congestion. Technical Report GIT-CC-96/02, Georgia Institute of Technology, 1996.
4. Cornell University. CU-SeeMe. Web page `http://cu-seeme.cornell.edu`.
5. A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Trans. on Software Engineering*, 24(5):342–361, May 1998.
6. J. Hartman et al. Liquid Software: A New Paradigm for Networked Systems. Technical Report 96-11, Univ. of Arizona, June 1996.
7. M. Hicks et al. PLAN: A Programming Language for Active Networks. Technical report, Univ. of Pennsylvania, November 1997.
8. ITU-T Recommendation H.323. *Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service*, November 1996.
9. M.A. Millet Sorolla. Realizzazione di un'applicazione su rete attiva. Master's thesis, Politecnico di Torino, Italy, February 1998. In Italian.
10. G.P. Picco. μCODE: A Lightweight and Flexible Mobile Code Toolkit. In *Proc. of the 2nd Int. Workshop on Mobile Agents (MA'98)*, September 1998.
11. M. Ranganathan, A. Acharya, S. Sharma, and J. Saltz. Network-Aware Mobile Programs. In *Proc. of the USENIX 1997 Annual Technical Conf.*, January 1997.
12. J. Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
13. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, January 1996.
14. Sun Microsystems. The Java Language: An Overview. Technical report, Sun Microsystems, 1994.
15. Sun Microsystems. *Java Media Framework*, January 1997. Available at `http://java.sun.com/products/java-media/jmf`.
16. D. Tennenhouse et al. A Survey of Active Network Research. *IEEE Communications*, 35(1):80–86, January 1997.
17. D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: A Toolkit for Building an Dynamically Deploying Network Protocols. In *Proc. of IEEE Open Architectures and Network Programming (OPENARCH'98)*, pages 117–129, April 1998.
18. Y. Yemini and S. da Silva. Towards Programmable Networks. In *IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management*, October 1996.