Mobile Agents for QoS Tailoring, Control and Adaptation over the Internet: the ubiQoS Video on Demand Service

Francesco Baschieri, Paolo Bellavista*, Antonio Corradi
Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna
Viale Risorgimento, 2 – 40136 Bologna - Italy
Ph.: +39-051-2093001; Fax: +39-051-2093073
{fbaschieri, pbellavista, acorradi}@ deis.unibo.it

Abstract

Service provision over the Internet has to address the issues of differentiated Quality-of-Service (QoS) and ubiquitous accessibility. Internet services should take into consideration the user QoS desiderata together with the various properties of servers providing replicated/partitioned services and of different access devices/points, from workstations over high-capacity ATM links to personal digital assistants over packet-switched 3G mobile phone. The major paper claim is that the provision over best-effort networks of services with negotiated and controlled QoS requires a distributed support infrastructure consisting of intermediate active nodes along the path between clients and servers. The paper deals with the Mobile Agent (MA) technology as suitable for implementing this active infrastructure and, in particular, presents the MA-based design and implementation of the ubiQoS Video on Demand (VoD) middleware. At negotiation time, ubiQoS establishes an active path between the requesting client and the proper VoD server to tailor the quality of VoD flow depending on user profile and device characteristics. At provision time, ubiQoS dynamically controls the offered QoS level to adapt locally when and where network resource availability changes.

Keywords: Quality of Service Management, Middleware, Mobile Agents, Video on Demand, Mobile Computing, Adaptability, Heterogeneity.

Areas of Interest: Multimedia Computing, Agents on the Internet, Mobile Computing.

* Please consider **Paolo Bellavista** as the author responsible for paper correspondence.

1 Introduction

A constantly growing number of users tend to access Internet services from ubiquitous points of attachment via an enlarging set of heterogeneous devices. Users tend to require differentiation and tailoring of Quality of Service (QoS), based on personal preferences and class of usage, by considering accounting aspects such as business/economic/free-of-charge QoS levels. The diffusion of mobile telecommunications and of mobile access to the Web, such as the NTT DoCoMo i-Mode and the Wireless Application Protocol (WAP) [1], widens further the heterogeneity of Internet client devices. Terminals span from traditional workstations and PCs, to laptops, personal digital assistants and smart phones, with wired/wireless continuous/intermittent connectivity. Any category of users/devices requires its proper QoS level and should be charged differently for received services.

In addition, the commercial competition among service providers and the necessity to achieve scalability over global networks force to consider service scenarios where it is common to have a group of servers capable of answering client requests. The choice of a specific server should consider client QoS requirements, but also the current conditions of server load and of network resource congestion. This is basic to estimate at negotiation time the possibility to achieve and maintain a specified QoS level at provision time over a best-effort network infrastructure [2].

Both service providers and network operators recently call for technologies, mechanisms, and tools to support Internet services with differentiated QoS, and to record, control and grant the QoS level provided at runtime. In the last years, several research efforts have investigated ad-hoc protocols at the network layer [3]. These solutions achieved interesting results for limited networks, but tend to clash with the best-effort model of the Internet. In addition, they require that intermediate routers traversed by service flows implement specific ad-hoc protocols. This constraint is likely to produce a long process of acceptance and diffusion. There are also other specialized low-level solutions that take into account QoS for specific targets, such as for ATM and for satellite connections [4]. As a general consideration, network-layer solutions work at a level of abstraction where it is difficult to embed some properties required for Internet service provision, such as application-specific adaptation and secure billing [2].

Recent research has pointed out the suitability of a distributed infrastructure consisting of intermediate nodes that play an active role along the path between clients and servers [5, 6]. Service provision should involve not only server hosts able to coordinate themselves to answer to client requests, and not only clients capable of both proposing profile information, such as device properties and user preferences, and enhancing service interactivity by offering local computing resources, as in the case of Java applets. Internet services should have the possibility to exploit also intermediate nodes that take part actively in service provision by performing operations on traversing service flows. For instance, some intermediate nodes should offer their storage resources to realize distributed caches of popular VoD contents for all clients and intermediate nodes of their locality, thus permitting to decrease overall traffic and service response time.

In addition, intermediate node participation is necessary for a scalable and decentralized middleware to deal with service provision and management in the open and global Internet environment [2]. Scalability imposes management decisions locally to the involved network resources and autonomous adaptation/recovery operations on service components when and where there are modifications in resource availability.

Mobile Agents (MAs) are a suitable middleware technology to design, develop and deploy the above described active services, i.e., distributed services where any involved node can play an active role by operating on service flows [6]. MAs can reallocate dynamically on the intermediate nodes of the service distribution paths and can exploit code mobility to achieve the dynamic deployment required in such open and global scenarios. In this way, MAs can control network resources and perform adaptation locally to the dynamically determined critical points of the infrastructure, e.g., where there is the need to overcome discontinuities in bandwidth due to either variations of communication resources or congestion situations.

The paper describes the design and implementation of an MA-based active service infrastructure, called ubiQoS[*], for the QoS tailoring, control and adaptation of VoD flows over standard best-effort IP networks. The name ubiQoS refers to the twofold ubiquity dimension of our middleware approach:

[*] The MA-based ubiQoS middleware is available at `http://lia.deis.unibo.it/Research/ubiQoS/`.

- *ubiquitous accessibility.* ubiQoS allows the reception of VoD flows anywhere, by tailoring VoD flows to user preferences, client device characteristics and network bandwidth at negotiation time, and by monitoring the QoS offered at provision time to perform corrective adaptation operations in response to possible dynamic changes in resource availability;
- *ubiquitous middleware.* ubiQoS tends to diffuse its components in the system. At negotiation time, middleware components autonomously distribute on the network hosts along the paths from receivers to sources of VoD flows. When new path segments are needed during service provision, e.g., in case of fault recovery, ubiQoS components can also migrate to the required locations without imposing restarting the service.

ubiQoS experimental results show the feasibility of the MA-based middleware approach in terms of achieved performance. Notwithstanding the Java-based implementation, ubiQoS can dynamically tailor, control and adapt the offered QoS levels by respecting the time constraints typical of VoD flows currently exchanged over the Internet.

In the following, the paper supports the claim that MAs are a viable technology to support active services with controlled QoS over the Internet. Section 3 describes the architecture and the main components of the MA-based ubiQoS middleware for QoS tailoring, control and adaptation of VoD flows. ubiQoS is built on top of the SOMA programming framework whose design guidelines are briefly sketched in Section 4 to provide the needed background for understanding the ubiQoS implementation insights given in Section 5. Comparisons with related research activities, conclusions and future work follow.

2 Mobile Agents to Support QoS-aware Active Services

The development, deployment and management of Internet services should meet the increasing user expectations and the growing requirements for QoS and dynamicity, and should face the flourishing heterogeneity in access devices and the globality of provision. In this context, the traditional end-to-end model of interaction is showing its limitations, thus suggesting the proposal of alternative scenarios. The network infrastructure should play an active execution role: for instance, in programmable networks, intermediate nodes operate on transmitted data and can be programmed by dynamically injecting service/user-specific code [6]. The approaches and technologies enabling the transition to programmable networks can

be roughly classified on the basis of their abstraction level: the terms active networks and active packets usually identify the approaches that achieve programmability by working mainly at the network layer, whereas active services tend to pursue network programmability by adopting solutions at the application layer [5].

Several research activities start to recognize that MAs are a suitable technology for the implementation of programmable networks, especially when adopting an application-layer approach [7, 8, 9]. MAs should be considered in the implementation of an active service infrastructure because MAs are autonomous entities with capacity of coordination, able to dynamically move to where resources are located, and able to adapt to current system conditions in a completely asynchronous way with regard to their launching user. The MA adoption simplifies the achievement of the following active service properties:

- ***control decentralization***. Cooperating MAs can migrate during service provision and can take autonomous management decisions based on local resource information. In addition, MAs can move dynamically to modify service distribution paths, e.g., in case of link failures or by following possible movements of both users and client devices;
- ***user asynchronicity***. Agent autonomy permits asynchronicity between user actions and MA-performed tasks. For instance, MAs can operate service negotiation and active path establishment also when users/access devices are temporarily disconnected. Differently from a traditional client/server model, stable connections are needed only for the limited time required to inject MAs into the network and to receive results. In this way, agent mobility significantly reduces connection times. This is particularly relevant in mobile computing where wireless connections exhibit strict constraints on available bandwidth and communication reliability [10];
- ***tailoring***. MAs provide an effective mechanism to support service tailoring in response to dynamically specified user requirements, access device characteristics, and resource availability at negotiation time. Dedicated agents can retrieve profile information, can propagate this information to current user access points and can customize service flows, in dependence of the current access devices and of already admitted service sessions. For instance, for accesses ranging from a laptop to a light PDA, an active service can decide to include/discard attachments in downloading e-mail messages;

- ***adaptability***. MAs simplify the adaptation of services in response to modifications in the availability of involved network resources at provision time [2]. For instance, MAs can exchange monitoring information and can migrate to obtain a global view of the system state. This MA coordination builds the shared knowledge that can trigger possible management operations to correct the achieved QoS (re-negotiation, additional communication channels, …). QoS control and adaptation work according to strategies either required by user profiles or decided by administrators depending on user roles.

In addition, MA solutions tend to address new requirements and putting together infrastructure properties that can significantly facilitate the active service acceptance and can enhance its effectiveness, such as:

- ***location awareness***. MAs tend to maintain full visibility of the location of underlying system resources and to propagate this visibility to the service level. Location awareness is a basic property to optimize resource usage within a locality [11]. For instance, MAs can decide to switch to another VoD server if the current one is overloaded and another one is currently available for a better service either in the same locality or in a near one;
- ***security***. The MA paradigm introduces not only specific security mechanisms and policies to deal with untrusted incoming code, but also easily integrates, at the application level, widespread and standard solutions for secure services. For instance, MA operations on system resources are controlled depending on permissions associated with authenticated principals and their proper role [12]. Based on these security mechanisms, any operation can be allowed, recorded and accounted for responsible users;
- ***interoperability***. MAs work in an open global scenario and should interoperate with existing components, from legacy systems to standard Internet services. Many MA systems achieve interoperability via compliance with general standards, such as CORBA, and more MA-specific, such as the MA Systems Interoperability Facility (MASIF [13] - the OMG standard for interoperability between heterogeneous MA platforms) and the Foundation for Intelligent Physical Agents (FIPA [14] - the standard specification proposal for agent platforms and communication languages) [15, 16].

Notwithstanding the above properties, one may argue that active services ask only for code mobility and not for full state migration typical of MAs. In addition, active services usually

require a single-hop mobility pattern and not full MA expressive power of specifying and performing multi-hop migrations. This consideration applies to the simple services commonly proposed, but the relevance of state migration raises for more complex and connection-oriented active services that could take advantage of maintaining and moving sessions. This is evident in mobile computing scenarios where MA-based active nodes should have to work as proxy of possibly disconnected users/client devices [10]. A reasonable conclusion is that, as stated in [17], "while none of the individual advantages of mobile agents is overwhelmingly strong, we believe that the aggregate advantages of mobile agents is overwhelmingly strong".

As a killer application area, several research activities have applied the MA technology to network and system management because of the possibility of moving management entities locally to administered resources [18]. For this reason, not so tied to the property of full mobility, many MA platforms give agents the possibility to access network and system properties, i.e., to have a certain degree of QoS awareness. In particular, in network monitoring and management, most MA-based prototypes can interrogate network elements via standard management protocols such as SNMP and RMON [19, 20]. In case of more complex service management functions, MAs should have also visibility of system/application-specific indicators, such as the list of the current threads of an application and, for each of them, the CPU effective time and the allocated memory. This requirement is not easy to grant, because most MA platforms are based on Java and the Java Virtual Machine (JVM) tends to hide kernel-level system properties. However, some work has recently achieved interesting results in extending monitoring visibility of Java MAs, with/out modifying the standard JVM [21, 22].

2.1 MA-based QoS Management of VoD Flows

QoS awareness is a key requirement for VoD services and QoS visibility is the property the active service infrastructure should be built around, so to negotiate and dynamically control QoS levels over best-effort networks. Two phases can be distinguished:

1. QoS tailoring at negotiation time, and
2. QoS adaptation at provision time.

The preliminary negotiation phase, prior to any service real data flow, can be considered a static negotiation of the QoS level. Its main goal is to arrive to the choice of the best possible

engagement of resources, on the basis of the user profile, the access device and the current system situation. The first step, in fact, is to retrieve user preferences and access device characteristics: their transportation is taken care by ad-hoc MAs. Then, the active service infrastructure should choose the VoD server capable of providing the requested content that best satisfies QoS requirements according to user and terminal profiles. Once the server is identified, the infrastructure should lead to the establishment of a server-to-client network path. The path represents the target for a set of MAs that can be locally installed to negotiate from there the QoS level any path segment has to maintain and to decide for any required multimedia scaling operation. The VoD flow distribution is tailored to match client required QoS specifications, also depending on already admitted service flows and current resource availability. The active service is implemented in terms of MAs that coordinate among themselves for its deployment; they are in charge of application-level admission control and reservation of their local resources. Any MA component accepts new reservations for VoD flows (or for enhancing the QoS level of already established ones) only if enough resources are available.

The second dynamic phase is necessary during service provision and is even more constrained than the first one because of imposed strict deadlines on reaction. Any deviation from conformity makes the service ineffective and should be avoided because it clashes with the initially negotiated QoS level. In fact, over best-effort networks, the QoS levels of VoD flows should change depending on the state of system/network resources along distribution paths. Therefore, QoS should be controlled at provision time, and changes in resource availability should trigger adaptation operations to readjust QoS levels. Adaptation can affect transmitted VoD data (from transcoding to frame resizing, from merging/splitting multi-layered tracks to reducing frame resolution and rate) but can ultimately modify established VoD paths. In this case, a new negotiation phase takes place for a possible redistribution of active MA components. Corrective operations at provision time should take into account user/terminal profiles to assign different priorities to the possible adaptation alternatives. For instance, a personal digital assistant with limited display capabilities can suggest lowering frame resolution instead of decreasing frame rate.

2.2 Active VoD Services and Mobile Agents: a Usage Scenario

To show more concretely how MAs can tailor, control and adapt the QoS of VoD flows to implement VoD active services, Figure 1 presents a possible deployment scenario.

An active service infrastructure answers the scalability issue by permitting to organize clients, servers and networked resources in hierarchies of locality abstractions. Active service MAs (and their hosts) can be grouped into domains that usually correspond to (a set of) local area networks with common administration and management policies. In addition, domains permit to limit the visibility scope of specific middleware components and their complexity.

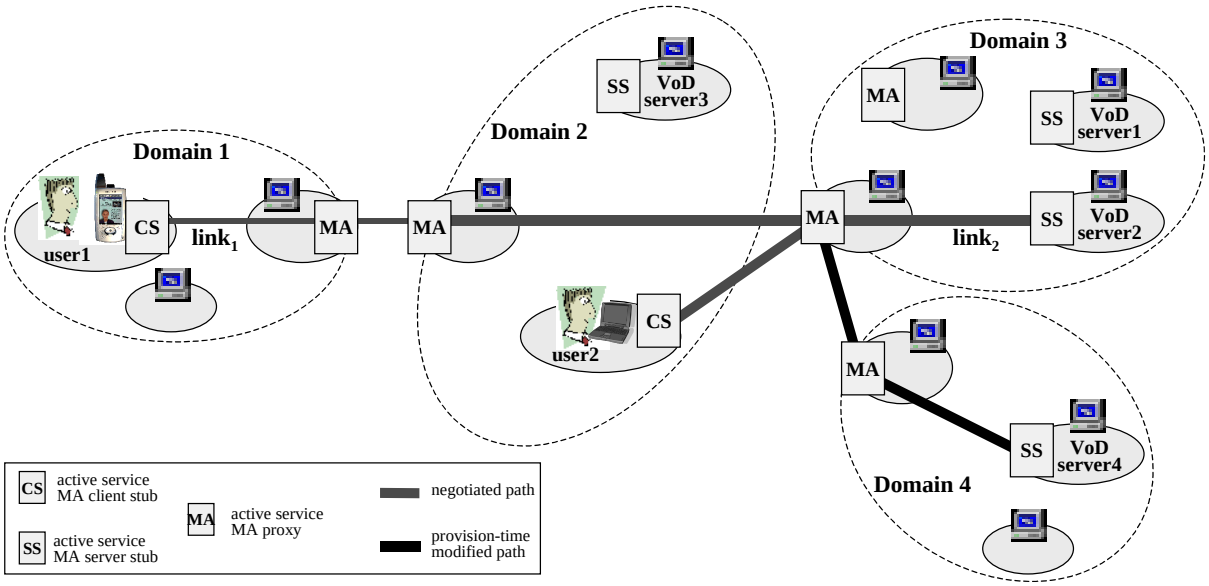

Figure 1. The MA-based active infrastructure performs service admission control and tailoring at negotiation time and VoD flow adaptation at provision time. The same VoD content with different QoS levels is transmitted to user1 and user2.

At negotiation time, MAs dynamically distribute on the hosts along the VoD path. The different QoS requirements of user1 and user2 (and of their access devices) have suggested a high-quality flow to the VoD server and its scaling down at the active service MA in domain2. At provision time, in case of degradation of $link_1$ bandwidth, after the coordination of active service MAs in domain1 and domain2, the same MA in domain2 adapts the VoD transmission for user1 by reducing the frame resolution according to the receiver preference profile. If there are no resources to adapt QoS by respecting negotiated requirements, e.g., in case of failure of $link_2$, a new VoD path segment can be established. The infrastructure in domain3 tries to identify a suitable VoD server in its near domains. Then, it negotiates with

new MA-enabled hosts, and finally restarts the flow transmission, from its interruption point (if server4 can support random-access to that VoD content). Apart from the time interval to establish the new VoD path, the server swap is transparent to receiver and to other intermediate hosting nodes.

3 The ubiQoS Active Service Infrastructure for VoD Services

The solution guidelines previously presented have driven the design and implementation of an MA-based active service infrastructure, called *ubiQoS*, for the support of QoS tailoring, control and adaptation of VoD flows over best-effort interconnections. ubiQoS is built on top of an MA framework, called Secure and Open Mobile Agents* (SOMA), with the double goal of testing the suitability of the MA technology in active services and also of verifying an a posteriori balance of achieved performance and service acceptance. The choice of SOMA is also motivated by the offered set of middleware facilities that can help the rapid development and deployment of MA-based Internet services. In particular, SOMA provides extensive facilities for QoS visibility and handling (SOMA monitoring and QoS facilities), for the definition of the most suitable trade-off between security level and performance (SOMA security facility), and for the interworking with other MA platforms, legacy resources/systems and available preexisting services (SOMA interoperability facility). Section 4 briefly gives some implementation insights of the SOMA middleware facilities.

Another important point preliminary to the ubiQoS architecture is the adopted basic protocol: our active service infrastructure exploits the Real-time Transport Protocol (RTP) for VoD flow transmission [23]. The choice of RTP stems from its complete diffusion in application-level approaches to QoS and from its relevance in the areas of mobile communications and multimedia distribution [24, 25]. As any application-level QoS solution, RTP attempts to meet QoS requirements without modifying the underlying best-effort network level. The RTP permits to monitor currently available QoS levels and to notify service components of any modification. RTP has its own control protocol for management operations, the Real-Time Control Protocol, in charge of handling control information. Most relevant information items are sender reports, generated by the sources of RTP-based multimedia flows, and receiver reports, filled by the target VoD clients. All reports include

either sender information such as RTP timestamps and the number of packets and bytes already transmitted or receiver statistics about the flow such as highest sequence number received, inter-arrival jitter, and fraction of lost packets since last report.

The ultimate goal of the ubiQoS active service infrastructure is to allow ubiquitous accessibility of VoD services from any device and from any Internet access point, with the opportune and negotiated QoS level. This goal requires the identification of some functions.

- Any client request is served after an initial negotiation phase that establishes an active path connecting the requesting client to a suitable VoD server. The server choice among the ones that could provide the requested VoD content is driven by a search for a QoS level at least equal to the required one. The QoS requirements usually stem from the profiles presented by client users and access devices, with a match operation that takes into account both. If the provided QoS level is greater than needed, some ubiQoS MAs on the path can perform down scaling. In this phase, ubiQoS MAs may migrate to intermediate nodes on the active path to install where needed operations are not yet available, thus dynamically building the service infrastructure. For instance, any node in the active path requires the local presence of an admission control agent in charge of accepting/refusing new VoD flow requests and, in case of acceptance, of reserving local resources correspondingly. Another example is the necessity of distributing dynamically some code to operate special VoD flow transformations where needed.
- The provisioning of QoS-enabled VoD services over the Internet requires not only a static phase of admission control and reservation but also a dynamic control of resource availability at provision time and the consequent handling of adaptation operations. These control phases should be enforced on any segment of the active path, and the MA technology can be effectively exploited to perform QoS monitoring in any locality traversed by VoD flows in order to decide locally any corrective intervention. Any local QoS level degradation should trigger adaptation actions on the VoD served flow by the ubiQoS MAs adjacent to the congested segments. The active infrastructure can decide to adapt the transmitted flow, e.g., via format transcoding, by maintaining the path;

[*] SOMA is available at `http://lia.deis.unibo.it/Research/SOMA/`.

otherwise, control MAs should try to establish new path segments either connecting to the same VoD server or to a less loaded available one.

- Another guideline to achieve effectiveness and performance is to share VoD contents that are likely to be required by other clients. ubiQoS organizes distributed caches of frequently accessed VoD flows to reduce overall traffic and latency and to increase service scalability. In particular, intermediate active nodes can maintain local caches depending on the access patterns of the clients in their locality. The amount of space that an active node should devote to its local cache, the cache refreshing time and the replacement policy are all choices that strongly depend on the characteristics of the locality, of its available resources and of the usual clients accessing from its points of attachment. As a consequence, it is important that administrators have the possibility to control and modify cache parameters during service provision by specifying a proper management policy. ubiQoS caching is implemented by MAs that enforce locally the cache policy defined by the locality administrators.

3.1 The ubiQoS Architecture

There are four main types of ubiQoS MAs that distribute along the active path between the (possibly multiple) VoD clients and servers for flow provisioning:

1) ***ubiQoS proxies*** are in charge of admission control/reservation for incoming/outgoing flows. They monitor system- and application-level state of local resources and are able to trigger local QoS adaptation operations. They coordinate with their previous and next ubiQoS proxies in the active path both in the initial negotiation phase and at provision time for resource availability changes. In addition, they should organize a global visibility of neighbor domains and other ubiQoS components within the domain. This helps in scalability of naming resolution and host/domain specific management decisions.
2) ***ubiQoS processors*** are in charge of performing QoS tailoring and adaptation operations on VoD contents depending on the specific QoS requirements of established sessions. In response to a new client request, one of this processor is in charge of retrieving QoS requirements of the user and of her current access device. The processor should carry this information by migrating to the nodes hosting ubiQoS proxies to establish the active path.

The ubiQoS processor reaches one node and then duplicates itself to forward a clone to the next ubiQoS proxy in the upstream until a suitable VoD server is reached.

3) ***ubiQoS client stubs*** forward VoD client requests to ubiQoS proxies and redirect RTP flows to their local visualization tools in a transparent way, to integrate ubiQoS with legacy VoD players. At the moment, we have implemented ubiQoS client stubs for Java Media Framework (JMF) [26] and Mbone `vic` [27] players.
4) ***ubiQoS server stubs*** answer to service requests from ubiQoS components by encapsulating VoD flows from legacy servers into RTP flows transparently. Up to now, we have implemented ubiQoS server stubs for JMF data sources.

All above components are implemented as MAs to permit dynamic installation and updating of existing functions even while the ubiQoS infrastructure is operating. ubiQoS server stubs migrate and install when and where a new VoD server registers to the ubiQoS infrastructure. ubiQoS client stubs can move at the new connection of an access device to permit its local VoD player to receive ubiQoS VoD flows. Proxies are the principal ubiQoS infrastructure components that monitor and control the resources locally to the active nodes; they install permanently on new hosts taking part in active paths and their migration is typically single-hop. On the contrary, ubiQoS processors are session/flow-dependent and transient components that have to propagate from the client toward the server by carrying the QoS requirements of client user/device for that specific service flow. Let us note that resource reservation, adaptation operations and path decisions may depend on previously established path segments; that makes important the multiple-hop potential granted by MAs.

While client and server stubs mainly play a simple role of VoD flow encapsulation to facilitate the integration with legacy VoD players and servers, the complexity of ubiQoS proxies and processors deserves a more detailed description of how they operate to provide their functionality.

3.2 ubiQoS Components at Work

We can start the description of a typical scenario of ubiQoS operations by stating the roles of its two main MA components, ubiQoS processors and ubiQoS proxies, with regard to their protocols and separation of duties. ubiQoS processors play the main role in the initial admission phase working as carriers for QoS requirements; in provisioning, they directly

operate tailoring and adaptation transformations on VoD flows. ubiQoS proxies, instead, control the currently offered QoS levels and trigger processor operations. In terms of the adopted protocol, ubiQoS proxies exploit RTCP reports for control duties, while ubiQoS processors employ RTP to receive/transmit VoD flows.

Any client request for an active VoD service is taken care by one ubiQoS processor in charge of finding and carrying the full information about QoS requested parameters and related profiles. Once the whole needed data has been collected, the ubiQoS processor helps in establishing the active path, by involving all necessary proxies. ubiQoS processors migrate toward the available ubiQoS proxies in the locality and in close/connected domains to present there the carried request. It is up to proxies the decision phase by comparing requested QoS levels and local resource availability.

If the ubiQoS proxy has direct local access to the VoD content with the proper QoS level, it behaves as the final VoD server in a simple client/server architecture. After the proxies command the most suitable QoS tailoring operations to the ubiQoS processors on the path, the VoD active service starts to flow, without any further proxy intervention.

Let us recall that ubiQoS permits caching for performance sake. ubiQoS proxies may access to the same VoD contents with different QoS levels, e.g., in case a local ubiQoS processor performs down-scaling operations to tailor the flow to an access device with limited visualization capabilities. The ubiQoS proxy chooses to cache the flow with the high/low QoS level depending on the locally enforced cache management policy. These policies are expressed in a high-level specification language, integrated in the SOMA programming framework [28].

If the VoD content is not directly available to the local proxy, then the establishment of the proper active path is carried to a near ubiQoS proxy. The ubiQoS processor responsible for establishing the active path is cloned and forwarded to the next ubiQoS proxy. The forwarded processor can have knowledge of the previous path segments and can bring the history of previous choices. This propagation goes on until a successful match occurs, between requested information, QoS levels and locally offered VoD contents. At this point, the whole active path has been successfully established, and all intermediate nodes host the needed ubiQoS components. Similarly to the chain of ubiQoS processors traversed by the flow, there

is an analogous path of ubiQoS proxies that play only a control role and work mainly in background for caching purposes.

During service provision, ubiQoS processors furnish the necessary transcoding operations: even if a flow has to traverse a chain of processors before reaching its destination, the introduced extra latency can be computed initially and taken into account before the service takes place. In addition, this latency (and the difference in latency of multiple receivers in case of multicast distribution) is usually not critical in non-interactive multimedia services such as VoD. In any case, after the initial admission control and if there are no variations, the whole proxy chain is only devoted to caching.

Location awareness and knowledge of local monitoring information at provision time drive the adaptation that may occur when the agreed QoS level cannot be maintained. The processor-based distribution of QoS requirements throughout the whole active path permits optimal decisions avoiding further negotiations. ubiQoS proxies have the duty of monitoring currently offered QoS levels and of identifying possible deviations. We claim that locality is the key for prompt identification: as soon as a proxy ascertains a problem, i.e., any QoS parameter can no longer be granted, it commands a corrective action to the processor. The most common situation is a network congestion of the local path segment. The easiest corrective action is to down-scale the VoD flow to pass around with a reduced quality provisioning. A more expensive countermeasure is to establish a new different sub-path to overcome the local problem situation. ubiQoS proxies are in charge of deciding the most suitable action by taking into account negotiated QoS parameters and user/terminal profiles of the receiver.

Figure 2 shows how the ubiQoS components cooperate in the service provision scenario proposed in Section 2. At negotiation time, ubiQoS proxies distribute on all the nodes of the active path that do not have one yet installed. Two different ubiQoS processors (for user1 and user2) establish two partially overlapping active paths by migrating and cloning on any involved active node. It is the user1 ubiQoS processor that performs the VoD flow down-scaling as specified in user1 terminal profile. At provision time, in case of degradation of $link_1$ bandwidth, ubiQoS proxies in domain1 and domain2 coordinate and command the user1 ubiQoS processor in domain2 to further reduce frame resolution of user1 VoD flow according to the receiver profile. In case of failure of $link_2$, a new VoD path segment is established. The

ubiQoS proxy in domain3 tries to identify a suitable ubiQoS server stub in close domains. Then, it starts a negotiation phase with the ubiQoS proxy of domain4 by cloning and migrating two new ubiQoS processors to this new domain.

Let us finally note that in multicast distribution of the same VoD content, the generated network traffic significantly decreases: the ubiQoS infrastructure, when ascertaining the possibility of multiple neighbor targets within a sub-tree of domain localities, can split packets late, only where necessary, by maintaining path sharing as much as possible.

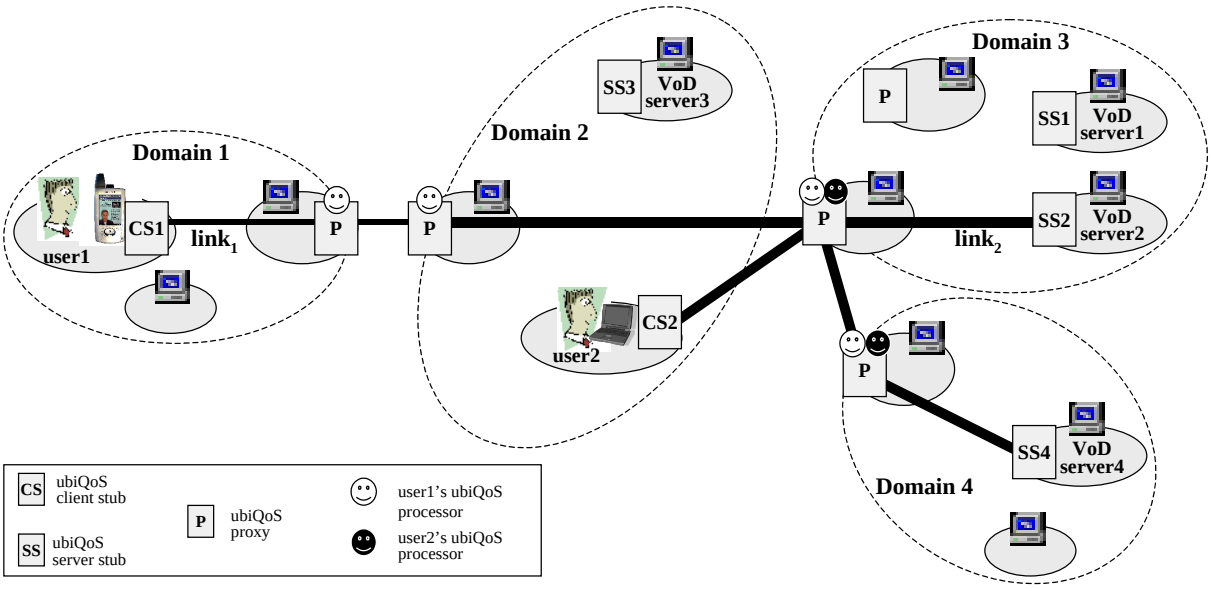

Figure 2. The ubiQoS infrastructure performing admission control, tailoring and adaptation when providing the same VoD content with different QoS levels to user1 and user2.

4 The SOMA Programming Framework

SOMA has been designed as a modular and flexible Distributed Processing Environment (DPE) with a wide set of middleware facilities. SOMA facilities include basic agent functions and more complex features suitable for the design and development of MA-based Internet services. The SOMA infrastructure is open and extensible because it permits to integrate dynamically the programming framework with new agent-based facilities possibly based on available functions.

Figure 3 depicts the SOMA infrastructure consisting of two facility layers. The Lower Layer of Facilities (LLF) provides SOMA agent basic functions:

- the ***naming*** facility. It maintains and permits to access any entity in the SOMA distributed middleware. A basic identification mechanism dynamically tags any local/distributed resource in the system (devices, objects, agents, service components and principals, i.e., users/organizations responsible for agent execution) with globally unique identifiers that do not change even after migration. They are the basis for the realization of the SOMA naming service that permits to bind to local/distributed resources via a set of different naming systems, e.g., LDAP-compliant and discovery-based [10];
- the ***communication*** facility. It provides tools for communication and coordination between possibly mobile entities. When hosted in the same execution locality, agents interact by means of shared objects, such as blackboards and tuple spaces for tight cooperation [29]. Otherwise, agents can perform coordinated tasks by exchanging asynchronous messages that are delivered also in case of migration of the target entity;
- the ***migration*** facility. It supports the transport of one entity that requests to change its allocation. Agents act on behalf of entities capable of reallocation and can move in the network either via SOMA native migration methods or via standard interfaces such as the OMG MASIF over the CORBA Internet Inter-ORB Protocol [13]. Reallocated entities can be traced also in their new locations by any entity in need of their services;
- the ***monitoring*** facility. It observes resource properties and provides information about them, from disk free space to currently available network bandwidth, from CPU usage to heap memory allocated by any agent thread [22]. This facility enlarges JVM visibility of kernel and application state by integrating with JVMPI [30] and with platform-dependent monitoring modules via JNI [31].

Upon this basic layer of middleware facilities, SOMA offers an Upper Layer of Facilities (ULF):

- the ***interoperability*** facility. It allows SOMA agents to interact with all local/distributed resources, in particular with legacy ones. This facility stresses compliance with common interoperability standards, such as CORBA for object systems [32] and JDBC for databases. In the specific MA area, SOMA supports the interoperation with other MA systems via compliance with the OMG MASIF interface and the FIPA specification;

- the ***security*** facility. It protects both MAs and hosting execution localities. Authentication is based on standard certificates and on the Entrust public key infrastructure. Authorization extends the Java standard mechanisms for access control. Secrecy is achieved by integrating the cryptographic libraries furnished by external providers. Integrity has required the development of MA-specific protocols for the protection of MAs from the execution environment [16]. Denial-of-service protection is considered a particular case of on-line QoS control and is enforced by the QoS facility introduced below;
- the ***QoS*** facility. It exploits the information collected and registered by the monitoring facility for control, adaptation and accounting of distributed operations performed by SOMA agents. Monitoring data are employed to control on-line the constraints on resource consumption imposed to SOMA agents and to adapt QoS-aware services to current resource availability, e.g., by transcoding multimedia flows in case of network bandwidth degradation. The security facility is in charge of granting non-repudiable association of agent bills to responsible users.

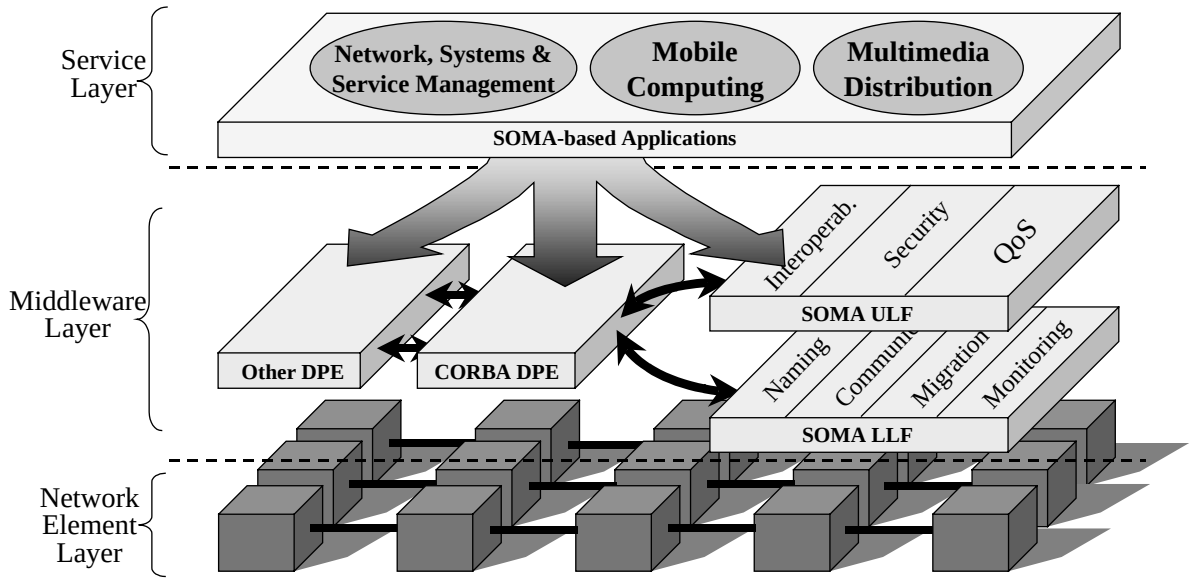

Figure 3. The SOMA layered architecture of middleware facilities.

In addition to these DPE facilities, SOMA offers locality abstractions to describe any kind of system interconnection, ranging from simple Intranet LANs to the Internet. Any node hosts at least one *place* for agent execution; several places are grouped into *domain* abstractions that

correspond to network localities. In each domain, a *default place* is in charge of inter-domain routing functionality and integration with legacy components via CORBA. The *mobile place* is the locality abstraction used to support mobile devices: it enhances the normal place with specific features for automatic reconfiguration when changing domain of attachment [10]. Further implementation details and performance of the SOMA programming framework can be found in [16].

5 The ubiQoS Implementation

The ubiQoS active service infrastructure requires mechanisms to retrieve dynamically all system lists: the VoD contents available in the global system, the ubiQoS proxies in the near ubiQoS domains and the user/device profiles to drive QoS tailoring and adaptation of provided VoD flows. This information is dynamically obtained by exploiting the SOMA naming facility that integrates discovery and directory servers. [10] reports a full description of the SOMA naming implementation, while the paper simply presents some elements necessary to fully understand how the components of the ubiQoS infrastructure interwork.

SOMA discovery and directory servers provide different naming solutions suitable for different goals. They differ in visibility scope (respectively, local vs. global), flexibility (rigidly predefined and simple structure vs. flexible content and organization), and performance (limited low-level efficient protocols vs. complete high-level searching/ registering operations). LDAP-compliant directory servers store the description of accessible VoD contents (title, associated keywords, multimedia format and QoS parameters), all profiles of ubiQoS registered users, and all profiles of recognized access devices, that is the information globally available in the ubiQoS system. On the opposite, Jini-based discovery servers permit to access the information visible in a single ubiQoS domain, usually the subset of VoD contents provided by the VoD servers in that domain, and the list of ubiQoS proxies either within or close to that domain locality.

User and terminal profiles are represented in a standard form to ensure the necessary interoperability over the open and heterogeneous Internet environment. In particular, ubiQoS adopts the Composite Capability/Preference Profiles (CC/PP), a World Wide Web Consortium standard proposal based on the XML Resource Description Framework (RDF), to represent profile information and to express exchange protocols. In the telecommunications

domain, WAP mobile phones are adopting CC/PP to tailor the provision of Internet services to their specific characteristics [33].

ubiQoS processors exploit JMF to perform tailoring and adaptation [26]. JMF, the SUN integrated framework for the management of acquisition, elaboration and visualization of multimedia flows offers a wide set of filter components to receive multimedia flows (acting as client receivers), to operate transformations on them, and to forward processed flows (acting as server sources). Typical transformations include compressions, e.g., reduction of frame size/rate and of single frame definition, and format transcoding, e.g., from MPEG-1 to H.263 for video. With regards to the transport and control of packet flows, JMF provides Application Programming Interfaces (API) for interacting with RTP and RTCP, and propagates visibility of RTCP transmission reports to application components. JMF components are portable on any platform that hosts a JVM. For performance sake, JMF processors often exploit local plug-ins available as native components, e.g., Dynamic Link Libraries for Windows platform and Shared Object libraries for Solaris and Linux. JMF integrates native plug-ins via the standard Java Native Interface technology to ensure compatibility of native code invocations for any JVM. Native libraries contain platform-dependent code and cannot be directly ported to different targets. To achieve portability, ubiQoS processors exploit the JMF API to retrieve dynamically the list of plug-ins installed on their hosting execution environment to bind to available native components.

The adoption of the SOMA technology simplifies the deployment of ubiQoS and the dynamic migration of its components wherever bottlenecks and critical points emerge during service provision. Bottlenecks can stem from heterogeneity in network characteristics, e.g., going from a 622Mbps ATM-based network to a 56Kbps modem link, and from heterogeneity in terminal capabilities, e.g., locally to WAP gateways that provide Web content to mobile phones. The default deployment choice is one ubiQoS proxy present (possibly newly installed at negotiation time) at any domain traversed by the VoD flow.

Our experience with the ubiQoS active service has shown that Java permits on-line QoS adaptation of multimedia flows at the usual Internet transmission rates. This is possible by exploiting Just-In-Time compilation techniques, native plug-in codecs, such as the ones distributed with the JMF and the commercial Cinax Design MediaPalette [34], and "pure Java" transcoders, e.g., the MPEG one implemented within the ubiQoS project. The

distribution of VoD contents encoded in the MJPEG format has exhibited interesting performance results. MJPEG compresses frames individually, without exploiting similarities between consecutive frames: this leads to a poor compression factor compared with other technologies, such as MPEG, but allows much more freedom in specifying encoding parameters (frame size, compression factor and frame rate) and makes VoD compression/transcoding operations less CPU-intensive. For instance, ubiQoS processors running on a 128MB PentiumIII700 host with Microsoft WindowsNT over a 10-Mbps Ethernet are able to manage up to a dozen of MJPEG flows with frame size up to 320*240 and with frame rate up to 20 Hz.

Figure 4 shows some GUIs for QoS monitoring and control provided to ubiQoS users and a JMF-based player while receiving an MJPEG flow. ubiQoS proxies usually decide the QoS level the ubiQoS processors should request to the following proxy in the active path towards the VoD server. The decision is taken on the basis of QoS requirements in user/terminal profiles, admission control reservation data, and a cost function with weighted QoS parameters.

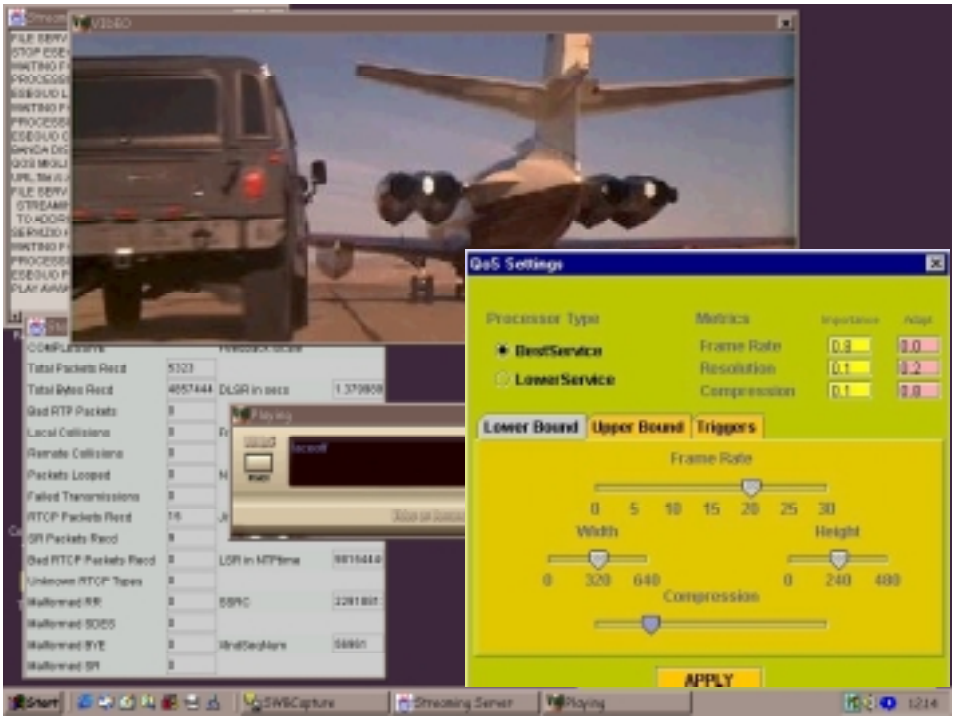

Figure 4. Some visualization and control GUIs provided by the ubiQoS infrastructure: RTCP sender/receiver reports and QoS monitoring information; QoS parameters and adaptation weights; a JMF-based player integrated with the ubiQoS client stub.

In fact, a whole interval for QoS parameters is usually permitted; the ubiQoS processor chooses the QoS point depending on the local resource consumption policy. In the current implementation, it is possible to choose between two simple policies: Best QoS and Lower QoS. The Lower QoS policy minimizes the local resource consumption, while the Best QoS policy chooses the QoS point that reserves the maximum local resource usage. In the latter case, new VoD flow requests can also dynamically modify QoS points of accepted flows by preempting previously committed resources.

It is also possible to force the ubiQoS processor decision by directly specifying low-level QoS parameters as depicted in Figure 4. The QoS Setting panel permits to change frame rate, size, compression factor, and their relative weight in the cost function. At provision time, the ubiQoS proxy can command the local processor to modify the provided QoS level by moving in the currently allowed interval by maximizing the cost function. For instance, a device with limited display capabilities can specify a frame rate weight larger than frame resolution to indicate a preference in degradation of image quality instead of frequency decrease.

6 Related Work

Many research groups have recently argued about the suitability of programmable networks for a wide spectrum of Internet services. Programmable networks can help in fast prototyping and deploying new network-layer protocols, e.g., for congestion control, topology-aware reliable multicast and virtual private networks [5, 6]. Most recent programmable network prototypes choose to work at the application-level and can be classified under the category of active services. In these approaches, network programmability is exploited to deal with application-specific requirements, as in distributed information filtering and Web caching [7, 35]. Most active service projects implement prototypes on top of the Java programming environment to facilitate code portability and mobility; some of them explicitly adopts the MA technology [8, 9].

In the specific application domain of multimedia services, there are a few notable approaches that exploit intermediate nodes for QoS tailoring, control and adaptation. Baldi et al. designed an active videoconference service by uploading Java mobile code in active network routers, thus adopting a network-layer approach [36]. Amir et al. implemented a Media Gateway service (MeGa) for the adaptive transcoding of multimedia flows [37]. Their

work, however, is more focused on algorithms for efficient down scaling and adaptive bandwidth allocation than on dynamic reconfiguration and code distribution.

The Reflector project is the research activity that has proposed the active infrastructure most similar to ubiQoS [38]. Reflector is an application-level multimedia distribution system implemented in C++. It has been designed and deployed mainly to test and verify the feasibility of distributing low and medium bandwidth VoD flows to thousands of simultaneous users over the Internet. The Reflector technology had a significant success in the live broadcast of NASA's Pathfinder mission. However, Reflector designers learned from wide-scale deployment experiences the necessity for a better support to dynamic reconfiguration, code distribution and adaptation to changes in network resource availability. It is interesting that they are addressing the observed limitations of Reflector by adopting the MA technology to enhance the system extensibility [39]. To our knowledge, there are no projects in literature that have addressed the issue of an active service infrastructure, designed and implemented in terms of Java-based MAs, for QoS-enabled VoD distribution over best-effort networks.

7 Conclusions and Future Work

The work accomplished within the ubiQoS project has shown the feasibility and the effectiveness of addressing the QoS issues of VoD services via an infrastructure of active nodes distributed along the path between VoD clients and servers. This choice seems suitable to enable QoS differentiation according to user/terminal profiles and to perform domain-specific flow tailoring, control and adaptation over a best-effort network infrastructure. The effectiveness of the ubiQoS implementation in terms of SOMA agents depends on operating close to controlled resources to take locality-dependent management decisions and on dynamically distributing middleware components at provision time. The experimental results indicate that the Java technology is mature to provide the basis for the implementation and dynamic deployment of portable middleware components for on-line QoS adaptation of multimedia flows at usual transmission rates.

Obtained performance results have encouraged us to continue the work in the ubiQoS project. In particular, we are focusing on how to extend the ubiQoS active service infrastructure with accounting functions. The information about resource consumption

provided by the SOMA monitoring facility is rich and articulated. However, the overhead imposed by the on-line control of MA operations can be significantly reduced in case of off-line consumption analysis for accounting [22]. Along this direction, we are refining the monitoring information to maintain concise and off-line logs on any SOMA place about user resource consumption; in addition, an MA-based service to collect and process the log data either at fixed intervals or when requested by SOMA administrators is under development.

References

[1] J. Krikke, "Graphics Applications over the Wireless Web: Japan Sets the Pace", *IEEE Computer Graphics and Applications*, Vol. 21, No. 3, pp. 9-15, May/June 2001.

[2] P. Bellavista, A. Corradi, C. Stefanelli, "An Integrated Management Environment for Network Resources and Services", *IEEE Journal on Selected Areas in Communication*, Vol. 18, No. 5, pp. 676-685, May 2000.

[3] X. Xipeng, L. M. Ni, "Internet QoS: a Big Picture", *IEEE Network*, Vol. 13, No. 2, pp. 8-18, Mar. 1999.

[4] N. E. Andersen, A. Azcorra, E. Bertelsen, J. Carapinha, L. Dittmann, D. Fernandez, J. K. Kjaergaard, I. McKay, J. Maliszewski, Z. Papir, "Applying QoS Control through Integration of IP and ATM", *IEEE Communications*, Vol. 38, No. 7, pp. 130-136, July 2000.

[5] K. Psounis, "Active Networks: Applications, Security, Safety, and Architectures", IEEE Communications Surveys, Vol. 2, No. 1, http://www.comsoc.org/pubs/surveys, 1st Quarter 1999.

[6] H. Yasuda (ed.), *Proc. 2nd Int. Working Conf. Active Networks (IWAN'00)*, Lecture Notes on Computer Science, Springer-Verlag, Japan, Oct. 2000.

[7] A. Ghosh, M. Fry, G. MacLarty, "An Infrastructure for Application Level Active Networking", *Computer Networks*, Vol. 36, No. 1, pp. 5-20, June 2001.

[8] D. Putzolu, S. Bakshi, S. Yadav, R. Yavatkar, "The Phoenix Framework: a Practical Architecture for Programmable Networks", *IEEE Communications Magazine*, Vol. 38, No. 3, pp. 160-165, Mar. 2000.

[9] S. Karnouskos, I. Busse, S. Covaci, "Agent Based Security for the Active Network Infrastructure", *1st Int. Working Conf. Active Networks (IWAN'99)*, Springer-Verlag, pp. 330-344, Germany, 1999.

[10] P. Bellavista, A. Corradi, C. Stefanelli, "Mobile Agent Middleware for Mobile Computing", *IEEE Computer*, Vol. 34, No. 3, pp. 73-81, Mar. 2001.

[11] J. Bolliger, T. Gross, "A Framework-Based Approach to the Development of Network-Aware Applications", *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998.

[12] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, "Security in Programmable Network Infrastructures: the Integration of Network and Application Solutions", in [6].

[13] GMD FOKUS and IBM Corp., *Mobile Agent Facility Specification (MASIF)*, Joint Submission supported by Crystaliz Inc., General Magic Inc., the Open Group, OMG TC Document orbos/97-10-05, ftp://ftp.omg.org/docs/orbos/ 97-10-05.pdf, 1998.

[14] *Foundation for Intelligent Physical Agents (FIPA)* – http://www.fipa.org/.

[15] IKV++, *Grasshopper 2: the Agent Platform*, http://www.grasshopper.de/

[16] P. Bellavista, A. Corradi, C. Stefanelli, "Protection and Interoperability for Mobile Agents: A Secure and Open Programming Environment", *IEICE Transactions on Communications*, Vol. E83-B, No. 5, pp. 961-972, May 2000.

[17] R. H. Glitho, "Emerging Alternatives to Today's Advanced Service Architectures for Internet Telephony: IN and Beyond", *Computer Networks*, Vol. 35, No. 5, Apr. 2001, pp. 551-563.

[18] M. Baldi and G. P. Picco, "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications", *Int. Conf. on Software Engineering (ICSE'98)*, IEEE Computer Society, pp. 146-55, Apr. 1998.
[19] B. Pagurek, Y. Wang, T. White, "Integration of Mobile Agents with SNMP: Why and How", *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2000)*, IEEE Press, USA, 2000.
[20] D. Gavalas, M. Ghanbari, M. O'Mahony, D. Greenwood, "Enabling Mobile Agent Technology for Intelligent Bulk Management Data Filtering", *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2000)*, IEEE Press, USA, 2000.
[21] G. Czajkowski, T. von Eicken, "JRes: a Resource Accounting Interface for Java", *Proc. ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'98)*, USA, 1998.
[22] P. Bellavista, A. Corradi, C. Stefanelli, "Monitor and Control of Mobile Agent Applications", *ACM OOPSLA Workshop on Experiences with Autonomous Mobile Objects and Agent Based Systems*, Minneapolis, USA, Oct. 2000.
[23] T. Braun, "Internet Protocols for Multimedia Communications - Part II: Resource Reservation, Transport, and Application Protocols", *IEEE Multimedia*, Vol. 4, No. 4, pp. 74-82, Oct. 1997.
[24] D. Chalmers and M. Sloman, "A Survey of Quality of Service in Mobile Computing Environments", *IEEE Communications Surveys*, Vol. 2, No. 2, http://www.comsoc.org/pubs/surveys, 1999.
[25] A. T. Campbell, "QoS-aware Middleware for Mobile Multimedia Communications", *Multimedia Tools and Applications*, Vol. 7, No. 1-2, pp. 67-82, 1998.
[26] SUN Microsystems, Inc. - Java Media Framework API, http://www.java.sun.com/products/java-media/jmf/
[27] UCL Networked Multimedia Research Group - *Mbone Conferencing Applications*, http://www-mice.cs.ucl.ac.uk/multimedia/software/.
[28] N. Dulay, R. Montanari, C. Stefanelli, "Flexible Security Policies for Mobile Agent Systems", *Microprocessors and Microsystems*, Elsevier Science, Vol. 25, No. 2, pp. 93-99, Apr. 2001.
[29] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications", *IEEE Computer*, Vol. 33, No. 2, pp. 82-89, Feb. 2000.
[30] Sun Microsystems - *Java Virtual Machine Profiler Interface (JVMPI)*, http://java.sun.com/products/jdk/1.3/docs/guide/jvmpi/jvmpi.html.
[31] R. Gordon, Essential Java Native Interface, Prentice Hall, 1998.
[32] P. Bellavista, A. Corradi, C. Stefanelli, "Middleware Services for Interoperability in Open Mobile Agent Systems", *Microprocessors and Microsystems*, Elsevier Science, Vol. 25, No. 2, pp. 75-83, Apr. 2001.
[33] W3 Consortium - *Composite Capability/Preference Profiles (CC/PP) Working Group*, http://www.w3.org/Mobile/CCPP/
[34] Ravisent Technologies Inc. - *Cinax MediaPalette*, http://www.cinax.com/Products/mp.html.
[35] W. Marshall, C. Roadknight, "Provision of Quality of Service for Active Services", *Computer Networks*, Vol. 36, No. 1, pp. 75-85, June 2001.
[36] M. Baldi, G. P. Picco, F. Risso, "Designing a Videoconference System for Active Networks", *Proc. 2nd Int. Workshop on Mobile Agents (MA'98)*, pp. 273-284, 1998.
[37] E. Amir, S. McCanne, R. Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding", *Proc. ACM SIGCOMM Conf.*, 1998.
[38] F. Kon, R. H. Campbell, S. Tan, M. Valdez, Z. Chen, J. Wong, "A Component-based Architecture for Scalable Distributed Multimedia", *Proc. 14th Int. Conf. Advannced Science and Technology (ICAST'98)*, pp. 121-135, 1998.
[39] F. Kon, R. H. Campbell, K. Nahrstedt, "Using Dynamic Configuration to Manage a Scalable Multimedia Distribution System", *Computer Communications*, Vol. 24, No. 1, pp. 105-123, Jan. 2001.