Vol. 43 No. 11 IPSJ Journal Nov. 2002

Regular Paper

A Mobile Agent-activated Middleware for Internet Video on Demand

Paolo Bellavista† and Antonio Corradi†

The widening of user requirements and the enlargement of terminal heterogeneity force to address the issues of differentiated Quality-of-Service (QoS) and ubiquitous accessibility in Internet services. The paper claims that the provision of services with negotiated and controlled QoS over best-effort networks is achievable via distributed support infrastructures that activate some of the nodes along the network path between clients and servers. The paper proposes Mobile Agents (MAs) as the activation technology to implement the needed active infrastructures and presents the MA-based design and implementation of the ubiQoS middleware for Video on Demand. At the negotiation time, ubiQoS establishes an active path of intermediate nodes capable of tailoring multimedia flow QoS depending on profiles of user preferences and of device characteristics. At the provision time, ubiQoS monitors the offered quality and promptly react to changes in resource availability by locally performing management operations, such as flow transcoding/downscaling and resource preemption, when and where needed.

1. Introduction

A constantly increasing number of users tend to access Internet services from ubiquitous points of attachment via a widening range of heterogeneous devices. Users tend to require differentiated and tailored Quality of Service (QoS), based on personal preferences and classes of usage, by considering also accounting aspects such as business/economic/free-of-charge QoS. The diffusion of wireless communications and of mobile access to the Web [1] widens further the heterogeneity of Internet client devices. Terminals span from traditional workstations and PCs, to laptops, personal assistants and smart phones, with continuous/intermittent ubiquitous connectivity.

Both service providers and network operators are calling for technologies, mechanisms, and tools to support Internet services with differentiated QoS, and to record, control and grant the QoS level provided at runtime. Several research efforts have recently investigated ad-hoc protocols at the network layer [2),3)]. These solutions achieved interesting results for limited networks, but tend to clash with the best-effort Internet model. In addition, they require that routers traversed by service flows implement specific ad-hoc protocols. This constraint is likely to produce a long process of acceptance and diffusion. As a general consideration, network-layer solutions work at a level where it is difficult to embed some of the functions required in QoS-enabled service provisioning, such as application-specific adaptation, secure billing and non-repudiable accounting [4)].

Some recent work has pointed out the suitability of distributed infrastructures where some intermediate nodes play an active role along the network path between clients and servers [5),6)]. Service provision involves not only a coordinated set of server hosts and not only clients capable of proposing profile information (user preferences and device properties) and of enhancing service interactivity by offering local execution resources, as in the case of Java applets. Internet services should also activate intermediate nodes for QoS-enabled service provisioning by operating on traversing data flows and reserving intermediate network resources. For instance, intermediate nodes should offer their storage resources to realize distributed caches of popular Video on Demand (VoD) contents for clients and intermediate nodes within their locality, thus permitting to decrease overall traffic and response time. In addition, the participation of intermediate nodes can achieve scalability and complete decentralization, crucial requirements for service provision and management in the open and global Internet environment [4)]. Scalability imposes management decisions locally to the involved resources and autonomous adaptation/recovery operations on service components when and where there are changes in available resources.

Mobile Agents (MAs) emerge as a middleware technology suitable to develop and deploy active services [6)]. MAs can exploit code mobil-

† DEIS — University of Bologna

ity to reallocate on the nodes of the distribution paths, thus allowing the needed dynamic deployment of middleware components. MAs can monitor/control network resources locally and autonomously, and can perform prompt management operations at the dynamically determined critical points of the network infrastructure, e.g., where there is the need to overcome discontinuities in bandwidth due to either variations of connection technologies or congestion situations. The paper describes the design and implementation of an MA-activated service infrastructure, called ubiQoS ☆, for the QoS tailoring, control and adaptation of VoD flows over standard best-effort networks. The name ubiQoS refers to the twofold ubiquity dimension of our middleware approach:

- *ubiquitous accessibility.* ubiQoS allows the reception of VoD flows anywhere, by tailoring multimedia content to user preferences, client device characteristics and available network bandwidth at negotiation time. In addition, it can monitor the provided QoS levels at provision time to perform corrective flow adaptation in response to modifications in available resources;
- *ubiquitous middleware.* ubiQoS tends to diffuse its components in the system. At negotiation time, middleware components autonomously distribute on the hosts along the paths from VoD receivers to VoD sources. When new path segments are needed at provision time, e.g., in case of fault recovery, ubiQoS components can migrate to the required locations without imposing any service restart.

The paper is organized as follows. Section 2 briefly describes the state-of-the-art of application-level technologies for Internet VoD services. Section 3 supports the claim that MAs are a feasible and effective technology to deploy active services with controlled QoS over the Internet. This gives the needed background to fully understand the design choices of the proposed ubiQoS middleware, presented in Section 4. Section 5 reports implementation insights and experimental results about the ubiQoS performance (path setup time, reaction delay, network traffic) and shows how its Java-based implementation can respect the typical time constraints of Internet VoD provisioning. Comparisons with related research activities, conclusions and directions of current work end the paper.

☆ The SOMA-based ubiQoS middleware is for download at http://lia.deis.unibo.it/Research/ubiQoS/.

2. Application-level Technologies for Internet VoD Services

Middleware solutions for VoD services can achieve a wide and rapid diffusion only by integrating with standard Internet mechanisms and technologies. There are three emerging and accepted application-level solutions that can help the deployment of QoS-aware VoD services over best-effort IP networks. We propose the ubiQoS middleware that exploits some of these technologies: the Real-time Transport Protocol (RTP) to transmit multimedia packets, the Java Media Framework (JMF) to process VoD flows, and the Composite Capabilities/Preference Profile (CC/PP) to manage user/terminal profiles.

2.1 Real-Time Transport Protocol

Recent research activities on QoS-enabled protocols have explored two different directions. At the network level, the research investigates the definition and standardization of new protocols based on the reservation of the needed amount of network resources [4]. However, the acceptance and deployment of new network-level protocols is long and difficult, mainly due to the large base of non-programmable and already installed network equipment. An application-level granted QoS is especially significant in the areas of mobile communications and multimedia distribution [6),7]. Application-level solutions try to meet QoS requirements without modifying the underlying best-effort IP network level: the guideline is to monitor the currently available QoS and to notify service components of quality modifications to trigger suitable management operations.

RTP, developed by the Internet Engineering Task Force (IETF) Audio/Video Transport working group, is the most widespread example of the application-level approach [8]. RTP supports communication with real-time constraints by defining the basic packet format for audio/video data transfer, e.g., sequence numbers of packets to permit packet loss detection or to determine the position of a video frame within a flow. Different encoding schemes (Motion-JPEG, H.261, ...) are supported.

RTP has its own control protocol, the Real Time Control Protocol (RTCP), which does not transfer data but handles only control informa-

tion. The most relevant information elements are sender reports, generated by the sources of RTP-based multimedia flows, and receiver reports, filled by the target VoD clients. Each sender report includes sender information such as RTP timestamps and the number of packets and bytes already transmitted; each receiver report contains receiver statistics about the flow such as interarrival jitter and fraction of lost packets since last report.

RTP is often integrated into application-level service components rather than implemented as a separate layer. Service components exploit RTCP to take service-specific corrective management operations on the served flows. According to RTP terminology, these components include mixers (when they generate a single new data flow out of several incoming data flows) and translators (when they operate audio/video format transcoding or multicast-to-unicast conversions). Finally, RTCP permits also to define service-specific control information elements.

RTP-based VoD service components can exploit sender and receiver reports to adapt the QoS level to the current conditions. However, the frequency of RTCP report transmission is determined in a rather static way, i.e., a fixed percentage of the maximum bandwidth available. This does not permit to exploit RTCP reports in taking prompt corrective operations in response to dynamic modifications in system/network conditions. The VoD community has identified this limitation and proposes the sending of reports based on events triggered by service-specific thresholds [9].

2.2 Java Media Framework

The Java technology plays a central role in the design, implementation, and deployment of Internet services. In addition to Java portability, dynamic class loading, and easy integration with the Web, the main motivation stems from its Java Virtual Machine (JVM) that hides the underlying platform by presenting a uniform vision of available computing resources.

In the VoD provision scenario, SUN proposes JMF, an integrated framework for the acquisition, elaboration and visualization of multimedia flows [10]. JMF considers crucial the concept of filter components, called processors. A wide set of processors permits to receive multimedia flows (acting as client receivers), to operate transformations on them, and to forward processed flows (acting as server sources).

Typical processor transformations include compressions, e.g., reduction of frame size/rate, and format transcoding, e.g., from MPEG-1 to H.263. In addition, JMF provides service developers with APIs to encapsulate source devices, such as heterogeneous acquisition hardware, stored VoD files, and incoming network flows, and to encapsulate receiver devices, such as encoding software for visualization, target VoD files, or outgoing network flows. With regards to the transport and control of flows, JMF provides APIs for interacting with RTP and RTCP, and propagates visibility of RTCP transmission reports to application components.

JMF proposes a layered architecture where service components exploit its APIs for presentation and processing. In their turn, these APIs may invoke native modules via the JMF Plug-in API layer. Plug-ins include:

- *multiplexers* combining multiple tracks of input data into a single output flow;
- *demultiplexers* parsing multimedia flows to extract separate tracks;
- *codecs* encoding/decoding the media information of a track;
- *renderers* processing the media information in a track and delivering it to a destination, such as a screen or speaker;
- *effects* performing special effects on the media information of a track.

JMF components are portable on any platform hosting the JVM. For the sake of performance, however, JMF processors often exploit plug-ins locally available as native components, e.g., Dynamic Link Libraries for Windows platform and Shared Object libraries for Solaris and Linux. JMF permits the integration with native plug-ins via the standard Java Native Interface (JNI) technology that ensures compatibility of native code invocations for all JVM implementations. Native libraries contain platform-dependent code and cannot be directly ported to different targets. However, JMF-based applications can exploit the JMF APIs to retrieve dynamically the list of installed plug-ins to try the binding only to available native components.

The performance of JMF-based VoD services can suffer from the Java implementation. Nevertheless, the integration with native plug-ins makes the JMF performance acceptable for a wide variety of VoD flow formats for different QoS levels. The main limitation is the rapid version evolution and prototypical state of JMF. For instance, JMF still provides a very

limited set of transcoders working properly over the different JVM implementations for different operating systems.

2.3 User and Terminal Profiling

The differentiation of user requirements and the wide heterogeneity of client devices impose to represent and maintain information on user preferences and terminal characteristics, i.e., user/terminal profiles. Service components should have visibility of profiles to drive the QoS tailoring and adaptation processes, both in the negotiation phase and during service provision. User profiles include desktop interface information, the default language, the required security level, and the subscribed services with the corresponding QoS requirements. Terminal profiles include hardware capabilities of devices and user-specified information on how to adapt service provision to the currently used terminal. For instance, if users are connected via handheld devices with limited audio capabilities, they are likely to receive only GSM-quality audio flows and not MPEGIII ones. On the basis of the profile, service providers or network operators can decide to discard MPEGIII packets addressed to these devices or to transcode them into a supported format.

The Internet openness and heterogeneity impose standard formats to represent, maintain and retrieve user/terminal profiles. Recent work proposes the adoption of extensible and open description languages, such as the eXtensible Markup Language (XML). For instance, the Resource Description Framework (RDF) exploits XML to express metadata describing Web-accessible resources in an interoperable way [11]. A variety of application areas can take advantage of RDF: resource discovery to provide better search engine capabilities, cataloguing to describe the content and content relationships among Web documents, security to express the privacy policy of a Web site. The World Wide Web Consortium promotes CC/PP, a standard proposal based on RDF, to represent the profile information and to express the exchange protocol. Mobile phones that support the Wireless Application Protocol (WAP) are adopting CC/PP to tailor the provision of Internet services to their specific characteristics [12]. Other research areas start recognizing the importance of user and terminal profiling. For instance, the Foundation for Intelligent and Physical Agents (FIPA) is defining an agent interoperability framework for nomadic support to deal with user profile management and mobile device capabilities [13].

Let us finally note that several proposals integrate in the same user profile both user preferences and current terminal capabilities, e.g., preferred language and screen resolution [12]. Even if service adaptation needs metadata about both user and terminal, we claim that the two dimensions should be cleanly separated. User profiles should contain only user-related information; devices should have their own profiles to inform the middleware of their characteristics independently. Only this separation achieves the flexibility and reusability requested by real scenarios where, for example, the same user can exploit a set of different terminals with different characteristics.

3. Mobile Agents to Support QoS-aware Active Services

The development, deployment and management of Internet services should face the challenging issues related to the increased QoS requirements and to the wide heterogeneity of access devices in the global provision scenario. It is within this context that the traditional end-to-end model of interaction shows its limits, 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]. Several research activities start to recognize the suitability of MAs in this activated scenario where active services exploit intermediate nodes typically programmed at the application layer [5),7),14),15)]. MAs are autonomous entities with capacity of coordination, able to dynamically move (together with their code and the reached execution state) 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 active service properties, such as:

- ***control decentralization***. Cooperating MAs can migrate during service provision and take autonomous management decisions based on local resource state. MAs can modify dynamically service distribution paths, e.g., in case of link failures or by following possible movements of users and

client devices. In addition, agent autonomy permits asynchronicity between user actions and MA-performed tasks. For instance, MAs can operate service negotiation and establish the active path also when users/access devices are temporarily disconnected;

- ***tailoring***. MAs provide an effective mechanism to tailor services to user requirements and resource availability at negotiation time. Dedicated agents can retrieve profile information, can propagate this information to current user access points and customize service flows, depending on the current access devices and the 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 network resources at provision time [4]. For instance, MAs can locally monitor network resources and dynamically migrate where needed to obtain a global view of the system state. This awareness permits to trigger management operations to correct the achieved QoS (re-negotiation, additional communication channels, ...) by exploiting locality to congested resources.

In addition, MA solutions tend to address novel requirements and to provide infrastructure properties that significantly enhance the effectiveness of active services, thus facilitating their acceptance and diffusion. The most relevant property in this context is 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 crucial to optimize resource usage within a locality [16),17]. 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.

In addition, MA-based middleware solutions facilitate the achievement of security and interoperability. On the one hand, MA systems not only introduce specific security mechanisms and policies to deal with untrusted incoming code, but also easily integrate standard solutions for secure services at the application level. For instance, MA operations can be controlled depending on permissions associated with authenticated principals and their role; based on these security mechanisms, any operation can be allowed, recorded and accounted to responsible users [18]. On the other hand, many MA systems achieve interoperability via compliance with general specifications, such as CORBA, and more MA-specific standards, such as the MA Systems Interoperability Facility and the Foundation for Intelligent Physical Agents specification [19),20].

Apart from the above properties that MAs can grant, one may argue that active services ask only for code mobility and that they do not require the full state migration typical of MAs: active services usually can take advantage of single-hop mobility patterns and not of multi-hop migrations. This consideration applies only to very simple services and commonly proposed case studies. The opportunity of state migration emerges in more complex and connection-oriented active services that require maintaining and moving sessions. This is evident in mobile computing scenarios where MA-based active nodes work as proxy of possibly disconnected users/devices [16]. A reasonable conclusion is "while none of the individual advantages of MAs is overwhelmingly strong, we believe that the aggregate advantages of MAs is overwhelmingly strong", as stated in Ref. 21).

Finally, since the beginning, MAs are considered a suitable technology for network and system management because of the possibility of moving management entities locally to administered resources [22]. 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, most MA-based prototypes can interrogate network elements via standard management protocols such as SNMP and RMON [23),24]. When used for higher-level service management functions, MAs should also have 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 hard to grant because most MA platforms are implemented in Java and the Java Virtual Machine (JVM) tends to hide kernel-level system properties. However,

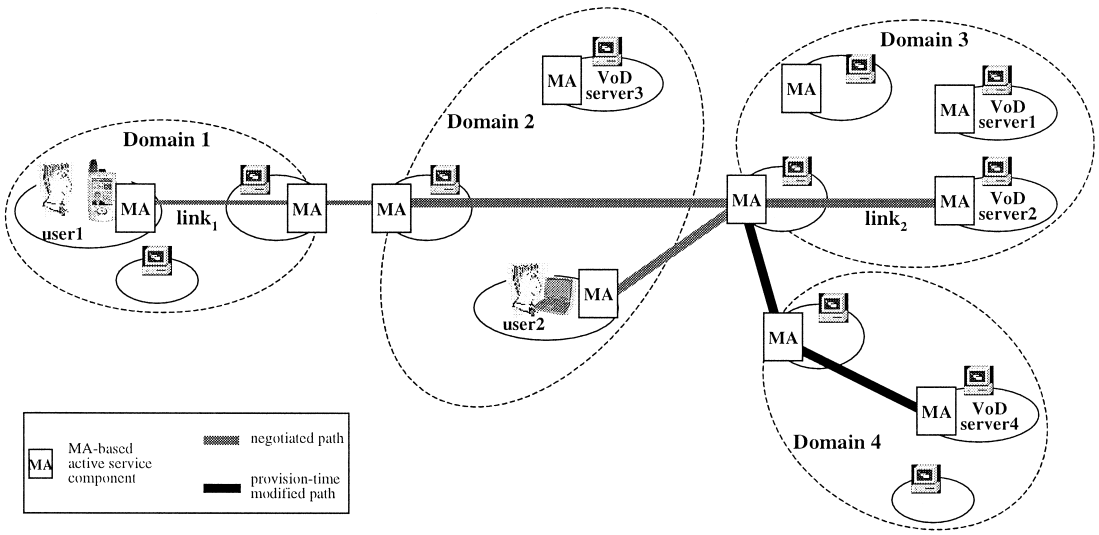

Fig. 1 A deployment scenario of the MA-based active infrastructure.

some work has recently achieved interesting results in extending the monitoring visibility of Java MAs, with/without modifying the standard JVM [25),26)].

3.1 MA-based QoS Management of VoD Flows

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;
(2) ***QoS adaptation*** at provision time.

The first phase precedes any real service flow and is necessary to negotiate the initial suitable QoS level. Its main goal is to determine the optimal engagement of resources, on the basis of the user preferences, of the characteristics of her current access device and of the currently available network resources. Ad-hoc MAs could retrieve user/terminal profiles and transport this information where needed. Then, the active service infrastructure could choose the VoD server capable of providing the requested content best satisfying the QoS requirements depending on the profiles. Once identified the server, the infrastructure could establish an active server-to-client network path. MAs can dynamically install along this path, to negotiate from there the QoS level that any path segment has to maintain and to decide for any required multimedia scaling operation. The VoD flow distribution is tailored also depending on already admitted service flows and current resource availability: MAs are in charge of application-level admission control and reservation of local resources.

The second phase is necessary during service provision and requires prompt reaction times. 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 can change depending on the state of system/network resources along distribution paths. Therefore, QoS could be controlled dynamically and possible modifications in available resources could promptly trigger adaptation operations. Adaptation operations include transformations on served VoD flows (from transcoding to frame resizing, from merging/splitting multi-layered tracks to reducing frame resolution and rate) and ultimately also the modification of the established active path. In this case, a new negotiation phase takes place for a possible redistribution of active MA components.

To show more concretely how MAs can operate to tailor, control and adapt the QoS of VoD flows, **Fig. 1** presents a possible active service deployment scenario. Clients, servers and networked resources are organized 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. 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) produce the distribution of a high-quality flow from the VoD server and the downscaling of the flow at the MA in domain2. At provision time, in case of degradation of link1 bandwidth, the MA in domain2 can adapt the VoD transmission to user1 by reducing the frame resolution according to the receiver preference profile. If there are not enough resources to adapt QoS by respecting negotiated requirements, a new active path segment is established. The MA 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 needed to establish the new path, the server swap is transparent to both receivers and intermediate nodes.

4. The ubiQoS Active Service Infrastructure

The above solution guidelines 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 networks. ubiQoS is built on top of an MA framework, called Secure and Open Mobile Agents (SOMA☆). The choice of SOMA is motivated by its middleware facilities for the rapid development and deployment of MA-based Internet services. SOMA provides facilities for QoS awareness and QoS management (Monitoring and QoS facilities), for the definition of suitable trade-offs between security level and performance (Security facility), and for the interworking with other MA platforms, legacy systems, resources and services (Interoperability facility) [16),20)]. In addition, ubiQoS exploits RTP for VoD flow transmission, due to the RTP diffusion in application-level approaches to QoS.

The ubiQoS ultimate goal is to allow ubiquitous accessibility of VoD services from any device and from any Internet access point, with the proper and negotiated QoS level. Any client request is served after an initial negotiation phase that establishes an active path connecting the requesting client to a suitable VoD server, i.e., a server that could provide the requested VoD content with a QoS level greater or equal to the required one. At the moment, the QoS level is expressed as a tuple including frame rate, frame size, compression factor (for MJPEG flows), and jitter. This tuple is obtained by combining the requirements stored in the profiles of the user and her current access device. If the QoS offered by the chosen VoD server is greater than needed, some ubiQoS MAs on the active path can downscale the flow. In this phase, MAs may migrate to intermediate nodes to install where needed operations are not yet available. For instance, any node in the active path requires the local presence of an admission control MA in charge of monitoring on-line local resource availability and of performing application-level reservation of local resources.

The provisioning of QoS-enabled VoD services over the Internet requires 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 help in performing QoS monitoring in any locality traversed by VoD flows in order to decide locally any corrective intervention. Any local QoS degradation triggers adaptation operations on exchanged VoD flows at the ubiQoS MA adjacent to the congested segment. The middleware can locally decide how to work on the flow, e.g., via format transcoding, by maintaining the path, or to establish new path segments, either connecting to the same VoD server or to a less loaded one.

To reduce overall traffic and latency and to increase service scalability, ubiQoS organizes distributed caches of frequently accessed VoD flows. 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 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 local usual clients. As a consequence, it is important that administrators can control and modify cache parameters during service provision by specifying a proper management policy. At the moment, ubiQoS offers domain administrators the possibility to choose which percentage

☆ The SOMA platform is available for download at http://lia.deis.unibo.it/Research/SOMA/.

of disk free space has to be exploited for cache storage and to adopt either a least-frequently-used or least-recently-used replacement policy.

4.1 The ubiQoS Architecture

To better detail how the ubiQoS active infrastructure provides the above functions, this section presents the main points of the ubiQoS architecture. Four types of ubiQoS MAs 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. They monitor system- and application-level state of their local resources and are able to trigger local adaptation operations. They coordinate with previous and next proxies in the active path both during the initial negotiation phase and at provision time when resource availability changes.

(2) ***ubiQoS processors*** are in charge of tailoring and adaptation operations on VoD contents depending on the QoS levels required in the currently provided sessions. In addition, in response to new client requests, new processors retrieve profile information and migrate to the involved proxies in order to establish the needed active path depending on client/server location.

(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 JMF [10] 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 server stubs for JMF data sources.

All above components are implemented as MAs to permit dynamic installation and updating of existing functions even while ubiQoS is operating. Suitable server stubs migrate and install when and where a new VoD server registers to the ubiQoS infrastructure. Suitable player-specific client stubs move at the new connection of an access device to permit its local VoD player to receive ubiQoS flows. Proxies install permanently on new hosts taking part in active paths and their migration is typically single-hop. On the contrary, processors are session/flow-dependent and transient components that 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, thus making definitely relevant the multiple-hop potential granted by MAs. While client and server stubs mainly play a simple role of flow encapsulation to integrate with legacy applications, the complexity of proxies and processors deserves a more detailed description, given in the following.

4.2 ubiQoS Proxies and Processors at Work

ubiQoS processors play the main role in the initial admission phase working as carriers for QoS requirements; during 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 adopted protocol, proxies exploit RTCP reports for control duties, while processors mainly employ RTP to receive/transmit VoD flows.

Any client VoD request is served by one initial session-specific processor in charge of finding and carrying the associated profile information. In addition, the processor operates to establish the active path, by involving all necessary proxies: first, it interrogates all proxies known in its current and close localities. If one of the proxies has direct local access to the requested VoD content and its local resource availability is compatible with the requested QoS level, the path is established. The processor migrates to the proxy, which behaves as the final VoD server in a simple client/server architecture.

Otherwise, the processor duplicates itself and forwards its clones to the known proxies. Forwarded processors carry the whole knowledge of previously established path segments and bring the history of previous choices. This propagation goes on until a successful match occurs between requested QoS and locally offered VoD contents. At this point, the whole active path is completed, other processors working to path establishment are notified and killed by exploiting the SOMA facilities for communication and coordination [16),20)], and all intermediate nodes host the needed ubiQoS components. Let us

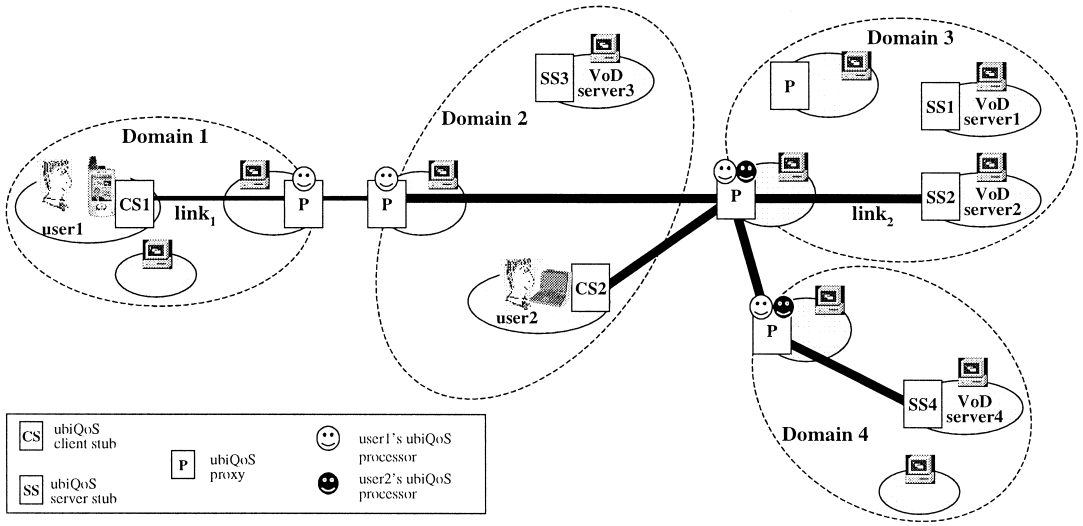

Fig. 2 The ubiQoS infrastructure providing different QoS levels to user1 and user2.

note that also proxies are MA-based and can install dynamically where needed. Once the active path is established, the involved proxies coordinate to command the most suitable QoS tailoring operations (see the following section) to their local processors and the active service flow starts, without any further proxy intervention.

At provision time, processors furnish the necessary transcoding operations. After the initial admission control phase and if there are no variations in resource availability, all proxies are only devoted to caching. 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 not critical in non-interactive multimedia services such as VoD.

Location awareness and on-line local monitoring 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. Proxies have the duty of monitoring currently offered QoS 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 downscale the flow to continue the service with reduced QoS. A more expensive countermeasure is to establish a new different subpath to overcome the local congestion. Section 5 sketches how proxies decide the most suitable service management operation by taking into account system state and client profiles.

Figure 2 shows ubiQoS components while cooperating in the provision scenario proposed in Section 3. At negotiation time, the proxies distribute on all the nodes of the active path that do not have one yet installed. The default deployment choice is to have at least one ubiQoS proxy for any participating network locality, but additional proxies can dynamically install where network bottlenecks emerge during service provisioning. Two different processors (for user1 and user2) establish two partially overlapping active paths by migrating and cloning on any involved active node. It is the user1 processor that performs the VoD flow downscaling as specified in user1 terminal profile. At provision time, in case of degradation of $link_1$ bandwidth, the proxies in Domain1 and Domain2 coordinate and command the user1 processor in Domain2 to further reduce frame resolution of user1 flow according to the preferences specified in the receiver profile. In case of failure of $link_2$, a new VoD path segment is established. The proxy in Domain3 tries to identify a suitable server stub in close domains. Then, it negotiates with the proxy of Domain4

by cloning and migrating two new processors to the new domain.

Let us finally note that in case of multicast distribution of the same VoD content, the ubiQoS infrastructure permits to decrease significantly the network traffic. When ascertaining the possibility of multiple neighbor targets within a sub-tree of localities, ubiQoS split packets late, by preserving path sharing as long as possible.

5. Implementation Insights

The ubiQoS active service infrastructure requires mechanisms to retrieve dynamically all system lists: the VoD contents available in the global system, the proxies in near network localities and the user/device profiles to drive QoS tailoring and adaptation. The SOMA naming permits to collect this information dynamically, by integrating discovery and directory servers. Reference 16) reports a full description of the SOMA naming implementation, while here the paper only sketches some elements to permit to fully understand how the ubiQoS components interwork.

SOMA discovery and directory servers provide different naming solutions suitable for different goals. They differ in visibility scope (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 registered users, and all profiles of recognized access devices. Profile information about new access devices may be described dynamically as an extension of already included profiles that exploit the CC/PP composition capabilities. On the opposite, Jini-based discovery servers permit to access the active infrastructure information visible in a single locality: the subset of VoD contents provided by intra-domain servers and the list of locally known proxies (usually, the ones within or close to that locality).

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 622 Mbps ATM-based network to a 56 Kbps 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.

To focus on the process of determining the QoS of the VoD flow exchanged on any active path segment, any ubiQoS processor autonomously decides the QoS level to request to the following processor towards the server. A whole interval for QoS parameters is usually permitted; the processor chooses the QoS point to enforce in the permitted QoS space interval depending on the local resource consumption policy. In the current ubiQoS implementation, system administrators can choose between two simple policies: Best QoS and Lower QoS. The Lower QoS policy aims at reducing the resource consumption by reserving only the set of resources that minimizes a specified local cost function. On the contrary, the Best QoS is a greedy policy that chooses the QoS point reserving the maximum local resource usage. While enforcing Best QoS, new VoD flow requests can also dynamically modify QoS points of accepted flows by preempting previously committed resources. In addition, ubiQoS provides administrators with GUIs to force the processor decision by directly specifying low-level QoS parameters, e.g., by changing frame rate, size, and compression factor.

At provision time, the proxy can trigger the local processor to modify the provided QoS by moving in the currently permitted interval of the QoS space according to the preferences expressed in the client profile (via relevance weights associated with the different QoS parameters [28]). For instance, the profile of 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. In other words, the weights determine the preferred directions of correction actions in the QoS space when the proxy detects a modification in local resource availability. Only when the allowed correction region is null, the proxy triggers the search of a new active (sub-) path and starts a new negotiation phase.

5.1 Experimental Results

To evaluate the feasibility and effectiveness

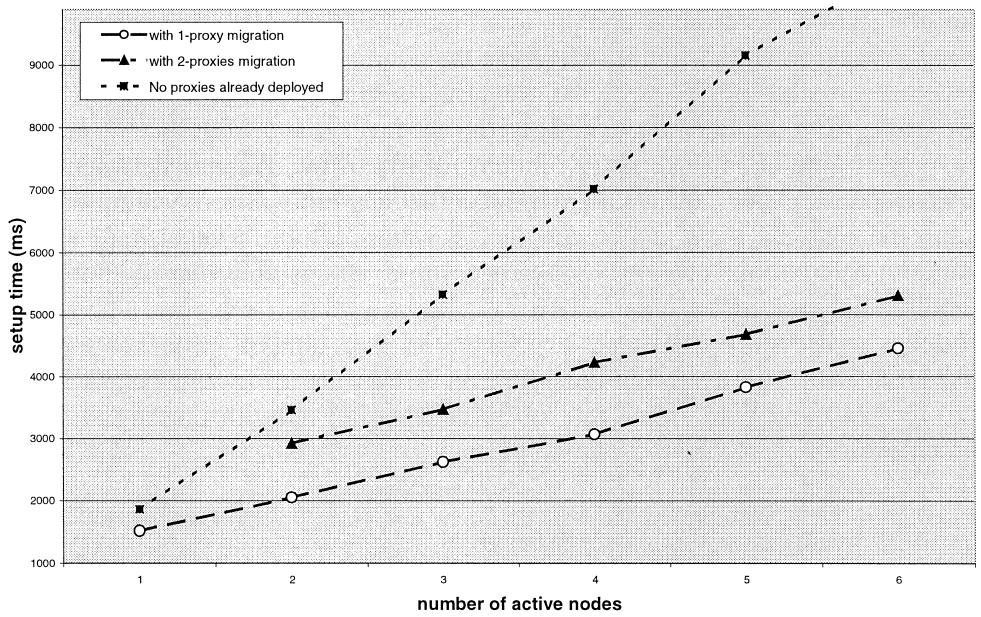

Fig. 3 ubiQoS path setup time.

of our approach, we have deployed the ubiQoS infrastructure over a set of geographically distributed networks, with heterogeneous bandwidth (10/100 Mb/s) and interconnected via GARR, i.e., the Italian Academic and Research Network. Any local network is modeled by one ubiQoS locality and includes heterogeneous hosts (SUN Ultra5 400 MHz workstations with Solaris 7, 128 MB PentiumIII700 PCs with Microsoft WindowsNT and 128 MB PentiumIII700 PCs with SuSE Linux 7.1). In this deployment scenario, we have measured several performance figures, listed in the following, to estimate the overhead and the reaction time of the ubiQoS middleware.

The initial phase of path establishment and negotiation in ubiQoS involves not only the client and the dynamically retrieved server, but also some active intermediate nodes. The establishment of any active path segment requires the interrogation of the local discovery service, the creation of an RTP connection, the migration of one processor MA, the resource admission control/reservation, the negotiation of the tailored QoS, and, when needed, the migration of one ubiQoS proxy MA. **Figure 3** shows an almost linear dependence of the path setup time on the number of intermediate active nodes. Let us specify that the migration of ubiQoS components is not required by any service request. In fact, ubiQoS proxies are infrastructure components that dynamically install where needed in response to a service request and that can persist there to serve future requests for VoD flows. For this reason, the figure also reports how the number of needed proxy migrations affects the path setup time: the delay significantly reduces in the usual case of a service request that triggers the installation of ubiQoS proxies only over a small set of new active nodes.

Let us additionally observe that, even in case of large-scale networks involved in Internet-wide service distribution, the number of active nodes along the server-to-clients path tends to be very small because it does not coincide with the number of traversed routers/gateways. In fact, ubiQoS proxies and processors need to operate only where there are either strong bandwidth discontinuities or forking of the VoD multicast tree. In any experimented usage scenario, with some dozens of geographically and randomly distributed clients, the dynamically determined ubiQoS active paths never included more than 4 intermediate active nodes; we needed to force the infrastructure decisions by adding ad-hoc bottlenecks to have longer active paths in order to measure the setup times, shown in Fig. 3, for the cases with 5 and 6 active nodes. However, the strong dependence between setup time and number of active nodes pointed out by the experimental results moti-

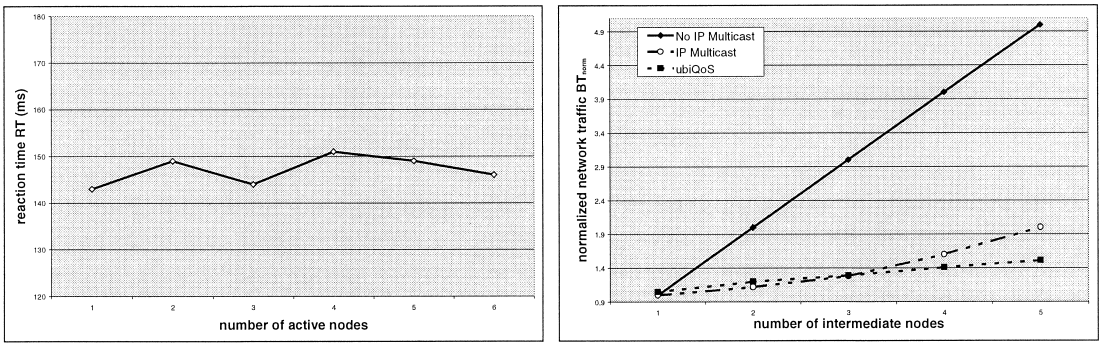

Fig. 4 ubiQoS reaction time (a) and multicast network traffic (b).

vated a slight modification of the ubiQoS path creation algorithm. In the current version, the middleware does not terminate the research of alternative paths by ubiQoS processors until a path with less than 6 active nodes is found. This obviously does not impact on path creation time but significantly reduces the following proxy negotiation time, which has demonstrated to be the slowest phase of the overall setup process.

The path setup time usually does not exceed 5 s, with an average number of active nodes less than 5 and an average number of required migrations less than 3. This interval is significantly larger than the one necessary to establish a single RTP connection between one client and one server, but is acceptable in most categories of multimedia services, i.e., non-interactive VoD services, since it affects only the delay with which the visualization starts at the client side. The overhead in path setup is largely counterbalanced by the possibility of performing prompt reactions at provision time in response to dynamic changes in network resources. ubiQoS proxies integrate standard RTP report transmissions with event-triggered exchanges of monitoring information provided by the QoS Monitoring module. This permits to overcome the RTP limit related to the frequency of reports, which is statically determined in RTP as a fixed percentage of the maximum network bandwidth available at the time of connection establishment [9]. **Figure 4 a** reports the ubiQoS Reaction Time (RT) when the bandwidth reduction over an active path segment triggers the downscaling of the transmitted VoD flow. RT is measured as the time interval between the congestion occurrence and the starting of the adapted flow transmission from the downscaling processor. In the figure, we report the average results obtained by measuring RT for all the different possible positions of the congested segment in the path (from the server to the first active node, from the first active node to the second, ..., and from the N^{th} node to the client). The RT average value is 148 ms, almost independently of the distance between the client/server and the congested segment. This is due to the fact that any ubiQoS proxy autonomously monitors, controls and manages its local path segment. RT values exhibit a very small variance and very limited random fluctuations around the average value, exclusively due to runtime variations in the non-ubiQoS network traffic over GARR.

About local transcoding, we have experimented that ubiQoS processors on 128 MB PentiumIII700 hosts with Microsoft WindowsNT can downscale up to 15 MJPEG flows with frame size up to 320*240 and with frame rate up to 20 Hz [28] notwithstanding their portable Java-based implementation.

In case of multicast distribution of the same VoD flow to multiple clients, the ubiQoS middleware can significantly reduce the network traffic by exploiting the location awareness typical of the MA programming paradigm. In traditional VoD systems over networks that do not support IP multicast, the VoD server is forced to generate N flows, one for each requesting client. The ubiQoS processors can ascertain whether there are several receivers interested in the same VoD flow within their served localities, can split VoD flows only when necessary and can downscale the VoD quality depending on the maximum QoS requirements in their distribution sub-tree. In a usage scenario with clients that requested the same QoS level, ubiQoS achieves the same traffic reduction of network-layer multicast support, without re-

quiring compliant hardware, e.g., IP-multicast routers. In case of different QoS level requests, ubiQoS can even generate less network traffic than IP-multicast solutions. In fact, instead of distributing one multicast flow for any different QoS request, the ubiQoS infrastructure uses one unified VoD flow and dynamically downscales it at the proper node in the active path, by exploiting its awareness of the QoS levels requested in any distribution sub-tree.

A significant estimate of the network traffic generated by multicast distribution is the overall traffic BT, defined as the total number of bytes exchanged between any pair of adjacent nodes (client stubs-processors-server stubs in ubiQoS, clients-routers-servers in traditional provision scenarios) along the VoD distribution tree. We measured BT for different values of the number N of intermediate nodes and with different locations of 9 multicast clients requesting the same VoD content. Our testbed assumed that 3 clients requested the VoD flow with gold QoS (original VoD format), 3 with silver QoS (50% bandwidth saving via frame rate reduction), 3 with bronze QoS (90% bandwidth saving via reduction of both frame rate and resolution). Figure 4 b shows the average results for BTnorm = BT / BTnoMulti where BTnoMulti is the BT value for distributing the flow over a single path segment with no multicast support. The figure reports the experimental results in case of VoD distribution with/without IP-multicast and by exploiting the ubiQoS infrastructure. On the average, for N greater than 3, ubiQoS reduces BT more than three times if compared with non-IP-multicast distribution.

6. Related Work

Many research groups have recently claimed 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)]. Among the active service approaches, network programmability is exploited to deal with application-specific requirements, as in distributed information filtering and Web caching [7),29)]. 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 [14),15)].

In the specific application domain of QoS and multimedia services, there are a few notable approaches that exploit intermediate active nodes. Baldi et al. designed a videoconference service by uploading Java mobile code in active routers, thus adopting a network-layer approach [30)]. Amir et al. implemented a Media Gateway (MeGa) for the adaptive transcoding of multimedia flows [31)]. Their work, however, is more focused on algorithms for efficient downscaling and adaptive bandwidth allocation than on dynamic reconfiguration and code distribution.

The Reflector project has proposed the active infrastructure most similar to ubiQoS [32)]. 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 that it is crucial to adopt technologies to facilitate the 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 extensibility of their system [33)].

7. Conclusions and On-going 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 is 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 MAs 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 adap-

tation of multimedia flows at usual transmission rates.

First ubiQoS performance results are encouraging and stimulate refinements and extensions of the implemented infrastructure. In particular, we are extending the middleware to include VoD client stubs based on the Java 2 Micro Edition and VoD multimedia players such as the TealMovie one [34], targeted to Palm portable devices with very limited visualization capabilities. In addition, we are focusing on how to extend ubiQoS with accounting functions. The SOMA monitoring facility already provides rich and articulated data about resource consumption for MA on-line control [26]. We are working on effectively processing this monitoring information to maintain concise off-line consumption logs at any ubiQoS locality and on developing an MA-based service to globally collect the log data at fixed intervals or when requested by ubiQoS administrators.

Acknowledgments Work supported by the Italian Ministero dell'Istruzione, dell'Universita e della Ricerca in the Project "MUSIQUE: Infrastructure for QoS in Web Multimedia Services with Heterogeneous Access".

References

1) Krikke, J.: Graphics Applications over the Wireless Web: Japan Sets the Pace, *IEEE Computer Graphics and Applications*, Vol.21, No.3 (2001).
2) Xipeng, X. and Ni, L.M.: Internet QoS: a Big Picture, *IEEE Network*, Vol.13, No.2 (1999).
3) Andersen, N.E., et al.: Applying QoS Control through Integration of IP and ATM, *IEEE Communications*, Vol.38, No.7 (2000).
4) Bellavista, P., Corradi, A. and Stefanelli, C.: An Integrated Management Environment for Network Resources and Services, *IEEE Journal on Selected Areas in Communication*, Vol.18, No.5 (2000).
5) Psounis, K.: Active Networks: Applications, Security, Safety, and Architectures, *IEEE Communications Surveys*, Vol.2, No.1 (1999).
6) Yasuda, H. (Ed.): *2nd Int. Working Conf. Active Networks* (*IWAN'00*) *Proceedings*, Japan, Springer-Verlag Lecture Note on Computer Science (2000).
7) Ghosh, A., Fry, M. and MacLarty, G.: An Infrastructure for Application Level Active Networking, *Computer Networks*, Vol.36, No.1 (2001).
8) Braun, T.: Internet Protocols for Multimedia Communications—Part II: Resource Reservation, Transport, and Application Protocols, *IEEE Multimedia*, Vol.4, No.4 (1997).
9) Wenger, S.: RTCP-based Feedback: Concepts and Message Timing Rules, IETF Internet Draft, http://search.ietf.org/internet-drafts/draft-wenger-avt-rtcp-feedback-01.txt (2000).
10) Sun Microsystems, Inc.: Java Media Framework (JMF) API, http://www.java.sun.com/products/java-media/jmf.
11) Decker, S., Mitra, P. and Melnik, S.: Framework for the Semantic Web: an RDF Tutorial, *IEEE Internet Computing*, Vol.4, No.6 (2000).
12) W3 Consortium: Composite Capability/Preference Profiles (CC/PP) Working Group, http://www.w3.org/Mobile/CCPP.
13) The Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org.
14) Putzolu, D., Bakshi, S., Yadav, S. and Yavatkar, R.: The Phoenix Framework: a Practical Architecture for Programmable Networks, *IEEE Communications*, Vol.38, No.3 (2000).
15) Karnouskos, S., Busse, I. and Covaci, S.: Agent Based Security for the Active Network Infrastructure, *1st Int. Working Conf. Active Networks* (*IWAN'99*) *Proceedings*, Germany, Springer-Verlag Lecture Notes on Computer Science (1999).
16) Bellavista, P., Corradi, A. and Stefanelli, C.: Mobile Agent Middleware for Mobile Computing, *IEEE Computer*, Vol.34, No.3 (2001).
17) Bolliger, J. and Gross, T.: A Framework-Based Approach to the Development of Network-Aware Applications, *IEEE Trans. Softw. Eng.*, Vol.24, No.5 (1998).
18) Bellavista, P., Corradi, A., Montanari, R. and Stefanelli, C.: Security in Programmable Network Infrastructures: the Integration of Network and Application Solutions, in Ref. 6).
19) IKV++, Grasshopper 2: the Agent Platform, http://www.grasshopper.de.
20) Bellavista, P., Corradi, A. and Stefanelli, C.: Protection and Interoperability for Mobile Agents: A Secure and Open Programming Environment, *IEICE Trans. Comm.*, Vol.E83-B, No.5 (2000).
21) Glitho, R.H.: Emerging Alternatives to Today's Advanced Service Architectures for Internet Telephony: IN and Beyond, *Computer Networks*, Vol.35, No.5 (2001).
22) Baldi, M. and Picco, G.P.: Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications, *Int. Conf. on Software Engineering* (*ICSE'98*) *Proceedings*, IEEE Computer Society (1998).
23) Pagurek, B., Wang, Y. and White, T.: Integration of Mobile Agents with SNMP: Why

and How, *IEEE/IFIP Network Operations and Management Symposium* (*NOMS 2000*) *Proceedings*, USA, IEEE Press (2000).
24) Gavalas, D., Ghanbari, M., O'Mahony, M. and Greenwood, D.: Enabling Mobile Agent Technology for Intelligent Bulk Management Data Filtering, *IEEE/IFIP Network Operations and Management Symposium* (*NOMS 2000*) *Proceedings*, USA, IEEE Press (2000).
25) Czajkowski, G. and von Eicken, T.: JRes: a Resource Accounting Interface for Java, *ACM Conference on Object-Oriented Programming, Systems, Languages and Applications* (*OOPSLA'98*) *Proceedings*, USA, ACM Press (1998).
26) Bellavista, P., Corradi, A. and Stefanelli, C.: Monitor and Control of Mobile Agent Applications, *ACM OOPSLA Workshop on Experiences with Autonomous Mobile Objects and Agent Based Systems*, USA, ACM Press (2000).
27) UCL Networked Multimedia: Mbone Conferencing Applications, http://www-mice.cs.ucl.ac.uk/multimedia.
28) Baschieri, F., Bellavista, P. and Corradi, A.: Mobile Agents for QoS Control, Tailoring and Adaptation over the Internet: the ubiQoS Video on Demand Service, *2nd Int. Symp. on Applications and the Internet* (*SAINT'02*) *Proceedings*, Japan, IEEE Computer Society Press (2002).
29) Marshall, W. and Roadknight, C.: Provision of Quality of Service for Active Services, *Computer Networks*, Vol.36, No.1 (2001).
30) Baldi, M., Picco, G.P. and Risso, F.: Designing a Videoconference System for Active Networks, *2nd Int. Workshop on Mobile Agents* (*MA'98*) *Proceedings* (1998).
31) Amir, E., McCanne, S. and Katz, R.: An Active Service Framework and its Application to Real-time Multimedia Transcoding, *ACM SIGCOMM Conf. Proceedings*, ACM Press (1998).
32) Kon, F., et al.: A Component-based Architecture for Scalable Distributed Multimedia, *14th Int. Conf. Advanced Science and Technology* (*ICAST'98*) *Proceedings* (1998).
33) Kon, F., Campbell, R.H. and Nahrstedt, K.: Using Dynamic Configuration to Manage a Scalable Multimedia Distribution System, *Computer Communications*, Vol.24, No.1 (2001).
34) TealPoint Software: TealMovie, http://www.tealpoint.com.

(Received September 2, 2002)
(Accepted September 5, 2002)

Paolo Bellavista is a research associate of computer engineering at the University of Bologna. His research interests include mobile agents, mobile computing, network and systems management, location/context-aware services, and adaptive multimedia systems. He received a Ph.D. in computer science engineering from the University of Bologna. He is a member of the IEEE, the ACM, and the Italian Association for Computing. Contact him at pbellavista@deis.unibo.it.

Antonio Corradi is a full professor of computer engineering at the University of Bologna. His research interests include distributed systems, object and agent systems, network management, and distributed and parallel architectures. He received an M.S. in electrical engineering from Cornell University. He is a member of the IEEE, the ACM, and the Italian Association for Computing. Contact him at acorradi@deis.unibo.it.