MUM: a Middleware for the Provisioning of Continuous Services to Mobile Users

Paolo Bellavista, Antonio Corradi, Luca Foschini
Dipartimento di Elettronica Informatica e Sistemistica - Università di Bologna
Viale Risorgimento, 2 – 40136 Bologna – ITALY
Phone: +39-051-2093001; Fax: +39-051-2093073
{pbellavista, acorradi, lfoschini}@deis.unibo.it

Abstract

Advances in wireless solutions and portable devices are enabling new challenging service scenarios where mobile users are willing to access ubiquitous and continuous services. This calls for novel middleware capable of tailoring service contents to client characteristics and of following client movements at provision time. The paper proposes MUM, a dynamic and flexible middleware to support continuous services to mobile users in ubiquitous scenarios. MUM performs service configuration by dynamically distributing middleware components to intermediate nodes along the client-server path and provides service session continuity by automatically migrating the session state in response to user movements during service provisioning. MUM exploits mobile agents to move both middleware components and session state, where and when needed, while it allows service developers to continue using the traditional client/server model for MUM-based application components. In addition, the paper presents the implementation of a Video-on-Demand service on top of MUM, with the goal of verifying the feasibility of our approach when applied to the challenging multimedia application area. First experimental results show that, notwithstanding the application-level approach, the MUM configuration/session migration times are compatible even with the strict requirements imposed by multimedia distribution over the best-effort Internet.

1. Introduction

The diffusion of networked computing environments at home/office/open public spaces and the proliferation of wireless-enabled portable devices, e.g., palm-sized computers, cellular phones and pagers, identify new scenarios of service provisioning where mobile users are willing to have ubiquitous and continuous access to both traditional and novel context-aware Internet services. This suggests not only to provide mobile users with seamless service accessibility independently of their movements and of their current access points to the network, but also to consider the strict limitations on the hardware/software characteristics of their current access terminals (together with their wide heterogeneity) [1].

The above issues are motivating novel distributed infrastructures to support service provisioning and tailoring at runtime, by involving the active participation not only of service end-points but also of intermediate nodes along the path between clients and servers, especially on the nodes at the wireless-wired network edges [2]. Middleware support over intermediate nodes is a strategy that have demonstrated to be capable of flexibly providing monitoring, control and adaptation of service flows, without leaving the whole management burden to either client or server end-nodes [3].

An active service is the result of the cooperation of interworking components distributed along the Service Path (SP), between client and server, and traversed by service flows. For instance, in the multimedia application domain, some recent research projects are proposing solutions where service components are distributed on the client, on the server and along the SP, to downscale Video on Demand (VoD) flows at the optimal node of the multimedia distribution tree, depending on both network resource availability and the profiles of served clients [4].

In particular, the paper claims the relevance of active middleware infrastructures in ubiquitous computing scenarios, mainly to support *service configuration*, i.e., the tailoring of provided levels of Quality of Service (QoS) depending on resource availability and client profile of characteristics, and *service session continuity*, i.e., the maintenance of session state notwithstanding user movements (also between different access points, connectivity technologies and access ter-

0-7803-8623-X/04/$20.00 ©2004 IEEE

minals) during service provisioning. We call *continuous service* an application service which supports both service configuration and session continuity.

Design and implementation of continuous services in ubiquitous environments is extremely complex, since it must address many unsolved issues such as provision-time client mobility and service reconfiguration in response to environment variations, in a very heterogeneous (for client characteristics, employed wireless technologies, changing availability of local resources) deployment scenario. This complexity can significantly slow down service development and diffusion, thus reducing the potential grow of the ubiquitous service market. For the above reasons, we claim the need of application-level middleware to simplify the design and deployment of continuous services. The application level is recognized as suitable to provide flexible solutions to some crucial mobility issues, such as application-specific caching/filtering, security, and interoperability. In fact, an application-level middleware can relevantly benefit from the availability of standard mechanisms, solutions and tools at this abstraction layer [5].

In particular, the paper proposes the **M**obile agent-based **U**biquitous multimedia **M**iddleware (MUM*), a dynamic and flexible infrastructure to transparently support service configuration and session continuity for ubiquitous environments. MUM performs service configuration by dynamically distributing middleware components to intermediate nodes along the SP, and provides session continuity by automatically migrating the reached session state where corresponding users move to during service provisioning. MUM is built on top of the SOMA platform and exploits Mobile Agents (MA) to move both middleware components and session state, where and when needed [6]. However, MUM allows service developers to continue using the traditional client/server model in the design and implementation of MUM-based application components.

Finally, the paper presents a VoD service for ubiquitous environments built on top of MUM. We verify the MUM support in a very challenging application domain, the multimedia distribution, to extensively check the feasibility of the application-level approach. The first experimental results coming from VoD service deployment show that MUM configuration/session migration times are compatible even with the requirements imposed by VoD distribution over the Internet.

* Additional details and the code of the MUM prototype are available at: http://www.lia.deis.unibo.it/Research/MUM/.

2. Service Configuration and Session Continuity Issues

For several reasons middleware for continuous services should include service configuration and session continuity as core functionalities. In the following we will discuss and analyze in detail these reasons together with some associated issues, presenting then the related work in the field.

Service configuration is the first fundamental step to address the heterogeneity of different clients as well as user mobility. For instance, consider the case of one user, Alice, watching the lesson, which she has missed in class in the morning, on her workstation at home. She suddenly remembers her appointment at the library to finish the group work. In the library a WiFi connection is available, but the video playing component is not installed on her PDA for memory sake. Hence, the appropriate player must be downloaded and installed on the PDA. In addition, depending on her PDA profile, it is necessary to include another component in the SP to suitably downscale the multimedia flow. This example shows that in highly heterogeneous domains not all components can be assumed already present and configured on all the nodes participating to a certain SP. In this sense, the service configuration facility saves time-consuming and error-prone set up operations by automating component downloading and configuration depending on client profiles.

Service session continuity permits to keep up and move service state when nomadic or roaming movements happen. Nomadic users access their services from wired terminals and, when they decide to log off, the infrastructure records their session state. Later, logging again on the same/another terminal, the middleware transparently re-establishes their session. For instance, Bob could open a session on his workstation at the office (departure node) working on his documents and accessing his services; when Bob goes back home, the infrastructure support will be able to find and restore the session on his personal computer (target node). Roaming users**,** instead, moving around with their wireless devices, want to keep their sessions, when they cross different localities, as it happens continuously for cell phone services. In both cases, when session movement is required, the middleware must perform the management operations for session handoff. Session handoff includes service component rebinding (disconnection and reconnection) and state movement; session handoff should be transparent to user and possibly to application components.

Service configuration and session continuity issues become particularly crucial for multimedia service

provisioning. In fact, this kind of services has strict QoS requirements to be respected even when provisioning over the best-effort Internet. For this reason, only complex design allows QoS tailoring to achieve resource booking at negotiation time and resource monitoring at runtime. In addition, multimedia service continuity is particularly critical during session handoff management.

All above issues call for novel approaches to service and infrastructure design [7]. So far some solutions have been proposed to assist nomadic users rather than device roaming, or code downloading rather than SP configuration, but to our knowledge none considers these issues together. This section presents some related research projects and prototypes without any pretence of being exhaustive: it sketches only a partial state of the art in middleware for user mobility and service continuity.

The $2K^{Q+}$ is a programming environment for developing and deploying component-based continuous services with QoS requirements [8]; recently its core has been extended adding nomadic and roaming mobility [9]. In *follow me Desktop*, an MA-based infrastructure, nomadic users can move their opened sessions, but there is no definition of SPs and of service configuration along that path [10]. Another system that explores the MA usage to support roaming mobility is [1] that proposes a general architecture for session handoff, by focusing on the tailoring/adaptation operations necessary when delivering service to access devices with very limited capabilities. The Telecommunications Information Networking Architecture (TINA) consortium has promoted the specification of novel middleware for personal mobility, by integrating TINA access sessions with MA [11]. Finally, specifically for code distribution and service configuration, there is an MA-based solution exploited in the distributed system for the live broadcast of NASA's Mars Pathfinder mission [12].

3. MUM Overview and Architecture

We have developed MUM, an MA-based middleware to support continuous services in ubiquitous environments. MUM simplifies service design by performing all the operations required for service configuration and session continuity.

MUM has been designed and implemented at the application layer to better face some aspects related to continuous service management, from dynamic un/installation of infrastructure/service components to configuration management, from resource accounting to security and interoperability [4]. We claim that operating at this level increases portability and flexibility. In addition, there is no need of modifying existing operating systems and traditional services already implemented can continue to execute along with the new ones.

The *location awareness* requirement guides the MUM design; it offers visibility of node positions at the infrastructure level. This awareness lets the middleware act transparently when SP management is required and makes possible informed choices at both configuration and session handoff time.

We have followed also two development guidelines: *simplicity* and *separation of concerns*. The first one means that MUM should provide the application developers with means that facilitate service development. In particular, service configuration, session handoff and resource management are demanded to the middleware. In addition, we would like to offer a framework for service component development that should follow the well-known C/S model, thus simplifying the application logic design. The second one calls for separation of concerns between service and infrastructure deployment. We decided to treat functional aspects, such as streaming control, at the service layer while non-functional ones, like resource booking/monitoring, configuration/reconfiguration management and mobility management, are performed at the infrastructure layer. Following these guidelines we divide components in two categories as explained in Section 3.1.

3.1. Application and Middleware Components

Within MUM we define two kinds of components: application components (ACs) and middleware components (MCs). The service developers implement the ACs that encapsulate the application logic for the specific application. MUM instead provides them with MCs that manage different aspects treated at the middleware layer, from QoS management to ACs mobility support.

For ACs development MUM supports three general kinds of ACs for continuous service design: *Client*, *Proxy* and *Server*. While the Client and the Server respond to the well known C/S model, the Proxy does not. The Proxy participates in service delivery as an entity that acts as both client and server, by passing the service data received from the previous entity in the SP to the next one. First, it is important to monitor the resource situation not only on the client and server end-nodes, but also in the middle of the SP. Secondly, its introduction lets the session handoff to be achieved near the user position as it will be explained in Section 4.2. Finally, proxies add flexibility and modularity to the overall service architecture. For instance, in the

future proxies could encapsulate the adaptation/transformation operations necessary when client profiles require it as well as data filter operations in response to user preferences.

3.2. The MUM Layered Architecture

The MUM architecture, see Figure 1, consists of two layers, the *Mechanisms Layer* that realizes basic services, and the *Facilities Layer* that encapsulates the strategies and implements advanced middleware services. Multimedia applications are built on top of the Facilities Layer.

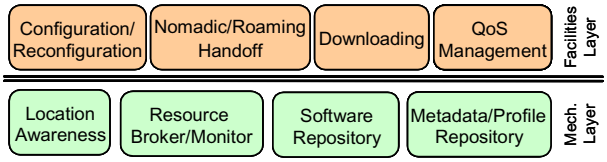

Figure 1: MUM layered architecture

The Mechanisms Layer realizes the basic middleware mechanisms offered to the more complex Facilities Layer:

- *Location Awareness* provides location visibility of the nodes to the upper layer, by using the location abstractions of SOMA organized in a tree [13];
- *Resource Broker and Monitor* perform resource monitoring and admission control;
- *Software Repository* stores downloaded code and component descriptors;
- *Metadata and Profile Repositories* maintain all the titles available within the system, the descriptors of presentations (i.e., presentation metadata), and client profiles.

The Facilities Layer includes the strategies and the most dynamic parts of MUM:

- *Configuration/Reconfiguration* configures the SP, by including also all QoS negotiation aspects, and by reconfiguring it at runtime in response to variations in the provisioning environment;
- *Nomadic/Roaming Session Handoff* realizes application handoff for nomadic/roaming users;
- *QoS Management* encapsulates the strategies for QoS Management at the middleware layer by using the above Resource Broker/Monitor;
- *Downloading* searches and downloads the needed code.

As observed in the previous section, the MA programming paradigm was already proposed and applied in some existing projects because of its properties of dynamicity, asynchronicity and autonomy [14] [1]. MUM proposes a mixed approach based both on Client/Server (C/S) and MA models to achieve the best of the two approaches. The mechanisms use only standard C/S solutions while the facilities mix the two approaches. That is motivated by the fact that mechanisms should be available on all nodes and are not expected to change frequently, while facilities encapsulate strategies that could modify frequently. In fact, MA-based technologies make it fairly simple to change and distribute Facilities Layer action strategies.

4. The Configuration and Nomadic Session Handoff Facilities

The main effort put so far in designing the MUM architecture strives for flexibility and generality. We have already developed all Mechanisms Layer modules described in [15] [16] [17]. The facility core modules implemented in the first MUM prototype are the *Configuration* and the *Nomadic Session Handoff Facility*. This paper presents them by explaining their crucial relevance and detailing their design.

The Configuration Facility is important because SP establishment is fundamental for continuous services and because Reconfiguration Facility should subsume it. The Nomadic Session Handoff is fundamental because it permits session state movement; it has been designed without dealing with other Roaming Session Handoff related issues, e.g., hopping connectivity.

4.1. Configuration Facility

This module offers functions for the configuration of distributed services and the QoS negotiation/booking at configuration time. This service is based on two MCs: the Decision Maker (DM) and the Plan Visitor Agent (PVA). Figure 2 presents the scenario considered for the configuration process.

The DM is an object without moving capabilities that encapsulates the strategies to decide how to configure the SP. By using the services offered by the Mechanisms Layer the DM gets the user profile, the position of the current node within the distributed system, and the presentation metadata for the title required by the user. The information permits to produce a *Plan* that contains one or more solutions corresponding to different QoS levels according to the adopted strategy. For instance, to take into account proximity, the Plan would be built by choosing, for the required title, the N presentations with the best QoS (one for each QoS level) nearest to the client. Note that one title may correspond to several presentations at different QoS levels: any solution states the actions to configure the SP for the specific presentation, i.e., the needed components and the necessary resources.

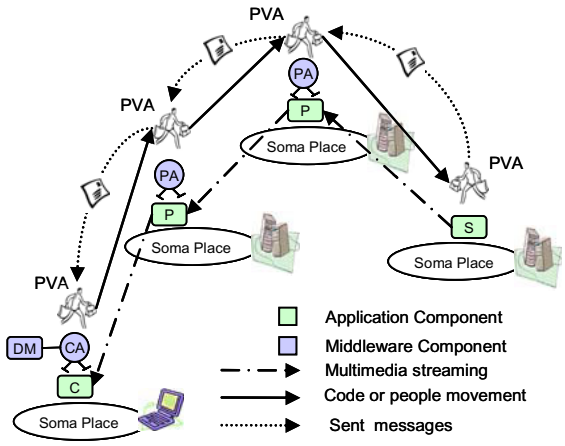

Figure 2: Configuration service scenario

The PVA is an MA that moves along the SP to configure it. Once the configuration service has obtained the Plan from the DM, it passes it to the PVA that goes through the path from the Client (C) to the Server (S) to negotiate and book resources, by downloading the components code if necessary, to initialize and configure all ACs and corresponding MCs. These MCs are the ClientAgent and the ProxyAgent (CA and PA in Figure 2) that control and manage the Client and the Proxy at runtime. The PVA design followed the Visitor Pattern, being the PVA the visitor and the Plan the visited structure, thus providing flexibility and easy extendibility [18]. In each participating node the PVA asks if there are enough resources for the first available solution in the Plan and, in that case, it books them and instantiates another PVA, to be sent forward. Arriving PVAs remain on the nodes participating in the service delivery to finish the component configuration. In fact, each PVA has to wait the endpoint of the next application component along the path (the envelope in Figure 2) sent back by next node PVA.

If there are enough resources in the system, the SP is established and service delivery can start, otherwise a configuration trace back is required. The configuration trace back begins as soon as one PVA does not succeed in booking required resources. In that case it checks if there is another configuration to explore (at a lower QoS level) in its plan. If there are enough resources for the second solution the configuration process can continue, otherwise a message is sent back to the previous PVA, by asking it to trigger an alternative solution.

4.2. Nomadic Movement Facility

This support is based on several MCs: ClientAgent, ProxyAgent and PVA. We assume that the user requires moving the opened session from the departure node to the target node. The locality-based design allows to keep session handoff cost-effective even when the distance between Client and Server grows. In fact, by adopting this approach, the required modifications impact only the last part of the SP, namely the Client and its nearest Proxy node, as Figure 3 shows in general and Figure 4 visualizes in its most relevant phases.

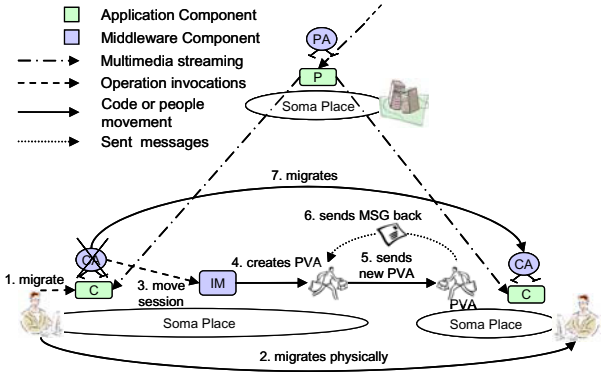

Figure 3: Nomadic movement scenario

The user request is caught by the ClientAgent and passed to the InitManager (IM in Figure3) along with the Proxy endpoint. Hence, the InitManager, a MC present on each node, obtains from the DM a Plan with the actions to take for the session movement. The Plan is passed to the PVA that sends a new PVA to the target node. Once arrived there, the new PVA downloads the Client code if necessary, initializes the Client and binds it to the Proxy. The session handoff terminates when the new PVA sends back a message triggering the ClientAgent movement to the target node. The designed protocol ensures session continuity, in fact when the ClientAgent migrates the Client is already present and initialized on the target node and the streaming has already begun. In other words, we succeed in non stopping the presentation. Let us observe that step 2 and steps 3 to 7 happen concurrently.

This proposed architecture is general enough to apply also to roaming. In fact, the migration process is rather similar with the main difference that it should deal with wireless connections and, instead of moving clients, should move proxies (the first in the SP from the client to the server). Of course more work is needed to decide how to detect device movements and to treat disconnections that can frequently occur in a wireless network.

4.3. MUM Implementation Insights

The main technologies involved in the implementation of the MUM are Java, the Secure and Open Mobile

Agent (SOMA), and the Java Media Framework (JMF).

Java has been chosen for several different reasons. First for its portability over different platforms. Moreover, Java is the typical implementation language employed by most MA environments because it permits an easy support to weak mobility, i.e., it is possible to move the application state, but not the execution stack of the MA [19].

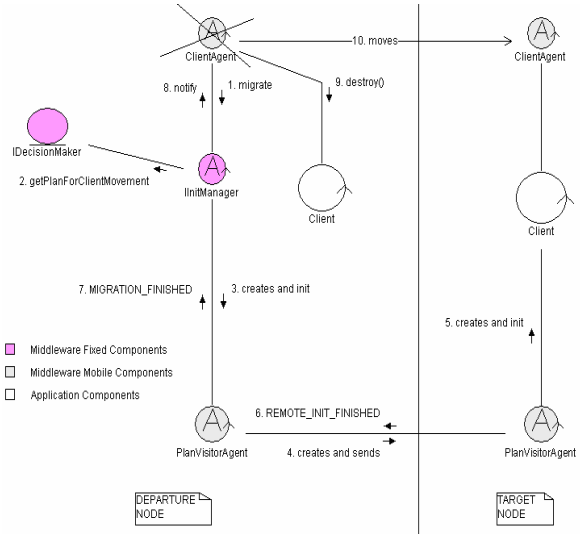

Figure 4: Nomadic movement service design

SOMA is a Java-based MA system with weak mobility, and MUM implementation is built on top of it. SOMA offers locality abstractions to describe any kind of interconnected system, from simple LANs to the Internet and these abstractions called places can be organized in a tree, to identify univocally resource positions. In other words, SOMA offers a good naming support to location of users, devices and presentations within the system, by offering the visibility needed to develop a location-aware middleware like MUM.

JMF is the SUN framework proposed for multimedia object management within Java environment [20]. JMF supports streaming, by adopting the RealTime Protocol (RTP) for video streaming, and by using the RealTime Control Protocol (RTCP) to monitor network status. The JMF implementation that uses only pure Java code solutions could incur in efficiency loss due to the fact that multimedia data management is CPU intensive. That is the reason why JMF provides the possibility of applying native plug-ins, even differently implemented for different platforms, such as Windows, Linux and Solaris, as long as accessible by the Java environment.

Other implementation details and the code of the MUM proposal are available at: http://www.lia.deis.unibo.it/Research/MUM/.

5. Experimental Evaluations

This section presents one typical Video on Demand (VoD) application developed within MUM and the experimental results collected from its execution. We have measured system performance for the realized service to see how MUM behaves in a very challenging scenario.

5.1. The MUM-based VoD Service

The realized application is a video streaming service with the possibility of commanding the presentation delivery. We are targeted to an Internet scenario where many users may desire to share their favorite videos. Each node participating in the coordinated architecture decides the presentations to make available for downloading, by registering them to the Metadata Repository. Then, the VoD service makes possible to remotely play the presentations commanding their delivery.

The service designer is involved only in designing the three application components: Client, Proxy and Server. In particular, they implement the GUI used to show the presentation and to command the video playing, realize the necessary protocols to pass the commands through the SP and manage the multimedia flows delivery. Let us stress again that the whole execution of the tasks required by service configuration, resource management and session movement is delegated to the MUM support. In particular, MUM is invoked to download the needed code, to book/monitor the resources, to initialize ACs, to bind them without any programmer intervention, and to move session state when needed.

5.2. Evaluation Methodology

The testbed used to run the application consists of a set of nodes connected by a 100 Mbps Ethernet LAN. Each node is a Sun Blade 2000 workstation equipped with a 900 MHz processor and 1024MB RAM. The operating system running on each machine is SunOS 5.9, and the rest of the software installed is the Java Virtual Machine (JVM) version 1.4.0_00_b05, and the Java Media Framework (JMF) Performance Pack for Solaris version 2.1.1e. The test done can be split into two parts, namely: the Service Configuration and the Nomadic Session Movement.

Service configuration. At the beginning of the test the Server, the Proxy and the Client are all not working. When one user logs to one node asking the beginning of the streaming, the MUM configures the SP by initializing, connecting and starting a server on the node

that stores the multimedia presentation, one or more Proxies and a Client.

Nomadic session movement. After a specified time, the user requires the nomadic session handoff to another node in the set, moving then to that terminal. In response MUM initializes a Client on the target node and carries the session movement off following the protocol introduced in 4.2. This closes the experiment.

5.3. Experimental Results and Discussion

Table 1 reports the average times required to complete the two processes described above.

Table 1. Average times for service configuration and nomadic session movement

Operation	Required time (msec)
Service Configuration	N x 366 + 456
Nomadic Session Movement	1011

As one can expect, the average time for initializing the distributed system is longer than the time required for the session movement. This is due to the fact that the first operation involves several ACs (corresponding to N nodes), while the latter only two. The collected times include the overhead spent for GUI creation, that is not negligible for a video streaming application. Table 2 and 3 report the experimental performance measured with a finer granularity.

Table 2. Service configuration

Operation	Time msec	Notes
Video Frame Configuration (local)	179	Time for the creation of the Video Frame that is the frame where the video is rendered.
RTP Session Configuration (distributed)	N x 366	Time for the configuration of the RTP Session over the distributed system. In particular this time takes into account the initialization time of all the components along the path, and the time to connect them all and begin the streaming.
JMF Player Initialization time (local)	121	Time for the initialization of the JMF player. The Player initialization begins only when the Client has received the stream.
Video Frame Activation (local)	72	Time for the activation of the video frame, in particular here the panel obtained by the JMF Player is added to the video frame.

Table 2 refers to service configuration. Most of the time is spent for configuration of RTP sessions. Hence the results mainly depend on the time required by JMF libraries to instantiate RTP session and the component used for streaming (namely the Processor), as shown clearly by the testing results.

Table 3 reports average times referred to nomadic session movement. Similarly to the service configuration case, most of the time is spent in the configuration of RTP sessions. Nonetheless, we stress that the time in this case is shorter because the nomadic session movement requires only to reconfigure RTP sessions between the terminal node running the Client and the Proxy, while the rest of the SP remains the same.

Table 3. Nomadic session movement

Operation	Time msec	Notes
Plan Visitor Agent Migration (distributed)	179	Time for PVA migration.
Video Frame Configuration (local)	179	Time for the creation of the Video Frame that is the frame where the video is rendered.
RTP Session Configuration (distributed)	341	Time for the activation of the Client on the target node and reconfigure the RTP Session between the Client and the Proxy.
JMF Player Initialization time (local)	121	Time for the initialization of the JMF player on the target node.
Video Frame Activation (local)	72	Time for the activation of the video frame, in particular here the panel obtained by the JMF Player is added to the video frame.

We have tested MUM for SP of different lengths. At the increasing of the node/components number, the service configuration time grows linearly and depends on the node number while the nomadic session movement maintains almost constant. This is due to the fact that the middleware reorganizes by adopting exclusively local strategies. Most of the time is caused by JMF libraries to create and initialize Processors, Players and to establish the RTP session, while the overhead introduced by the execution of MUM itself seems to be acceptable.

In conclusion, the obtained times are promising: an interval lower than 2 seconds is good for session migration if nomadic session mobility is considered within most common applicative scenarios. Indeed, these results are particularly appealing for soft real-time service provisioning such as VoD distribution at the currently usual Internet transmission rate. In fact, we can observe that the nomadic session movement time is comparable with the time usually required for switching between terminals.

6. Conclusions and Future Work

The paper has presented an MA-based middleware for supporting configuration and session movement management operations in ubiquitous environments. The

integration of the MA and the client/server paradigms in MUM has demonstrated how to achieve a good balance between middleware dynamicity and service development simplicity. In addition, the first experimental evaluations show that the configuration and session migration management operations can be performed at the application level with overhead and response times that are compatible even with the strict requirements of VoD distribution.

These encouraging results are stimulating our further investigation and middleware extension. On-going research activities include the development of different strategies for both the initial choice of the VoD source and for the SP runtime reconfiguration. In addition, we are completing the testing and the integration of the MUM session mobility module for users who roam between different access network localities during service provisioning. Finally, experiments have given important feedbacks and future work will include also the re-engineering of JMF-based MUM parts, trying to identify and remove the bottlenecks in the library execution in order to improve the middleware performance.

Acknowledgements

We thank Prof. Klara Nahrsted at the University of Illinois at Urbana Champaign for encouragement in this research. The work is partially supported by the Italian MIUR within the FIRB WEB-MINDS Project and by the Italian CNR within the Strategic IS-MANET Project.

References

[1] P. Bellavista, A. Corradi, C. Stefanelli, "The Ubiquitous Provisioning of Internet Services to Portable Devices", *IEEE Pervasive Computing*, Vol. 1, No. 3, 2002.

[2] D.L. Tennenhouse et al., "A Survey of Active Networks Research", *IEEE Communications*, Vol. 35, No. 1, 1997.

[3] P. Bellavista, A. Corradi, "How to Support Internet-based Distribution of Video on Demand to Portable Devices", *IEEE 7th Int. Symp. on Computers and Communications (ISCC'02)*, 2002.

[4] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, "Dynamic Binding in Mobile Applications: a Middleware Approach", *IEEE Internet Computing*, Vol. 7, No. 2, 2003.

[5] R. Oppliger, "Security at the Internet Layer", *IEEE Computer*, Vol. 31, No. 9, 1998.

[6] SOMA, Secure and Open Mobile Agent, Home Page available at: http://www.lia.deis.unibo.it/Research/SOMA/.

[7] K. Geihs, "Middleware Challenges Ahead", *IEEE Computer*, Vol. 34, No 6, 2001.

[8] K. Nahrstedt, D. Wichadakul, X. Gu, D. Xu (2001), "2Kq+: An Integrated Approach of QoS Compilation and Reconfigurable, Component-Based Run-Time Middleware for Unified QoS Management Framework", *IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware 2001)*, 2001.

[9] K. Nahrstedt, D. Xu, D. Wichadakul, B. Li, "QoS-aware Middleware for Ubiquitous and Heterogeneous Environments", *IEEE Communications*, Vol. 39, No. 11, 2001.

[10] K. Takashio, G. Soeda, H. Tokuda, "A Mobile Agent Framework for Follow-me Applications in Ubiquitous Computing Environment", *IEEE Distributed Computing Systems Workshop*, 2001.

[11] A. Kupper, A.S. Park, "Realizing Nomadic Communication with Mobile Agents: Strategies and Their Evaluation", *TINA Telecommunications Information Networking Architecture*, 1999.

[12] F. Kon, R.H. Campbell, K. Nahrstedt, "Using Dynamic Configuration to Manage a Scalable Distribution System", *Elsevier Computer Communication*, 2000.

[13] P. Bellavista, A. Corradi, C. Stefanelli, "Mobile Agent Middleware for Mobile Computing", *IEEE Computer*, Vol. 34, No. 3, 2001.

[14] A. Fuggetta, G.P. Picco, and G. Vigna, "Understanding Code Mobility", *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, 1998.

[15] P. Bellavista, A. Corradi, C. Stefanelli, "Java for On-line Distributed Monitoring of Heterogeneous Systems and Services", *Oxford University Press The Computer Journal*, Vol. 45, No. 6, 2002.

[16] P. Bellavista, A. Corradi, "Mobile Middleware Solutions for the Adaptive Management of Multimedia QoS to Wireless Portable Devices", *IEEE Workshop on Object-oriented Real-time Dependable Systems*, 2003.

[17] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, "Context-aware Middleware for Resource Management in the Wireless Internet", *IEEE Transactions on Software Engineering*, Vol. 29, No. 12, 2003.

[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[19] G. Cabri, L. Leonardi, F. Zambonelli, "Weak and Strong Mobility in Mobile Agent Applications", *Conf. and Exhibition on The Practical Application of Java*, 2000.

[20] Java Media Framework, Home Page available at: http://java.sun.com/products/java-media/jmf/.