UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica Informatica e Sistemistica
Dottorato in Ingegneria Elettronica e Informatica
XIII Ciclo

Mobile Agent Models and Technologies for Distributed Coordinated Applications in Global Systems

Candidato:
Ing. Paolo Bellavista

Coordinatore:
Chiar.mo Prof. Ing. Fabio Filicori

Relatore:
Chiar.mo Prof. Ing. Maurelio Boari

Anno Accademico 1999-2000

1 INTRODUCTION 1

2 MOBILE AGENT-BASED INFRASTRUCTURES FOR INTERNET SERVICES 7

2.1 EMERGING ARCHITECTURES IN STATE-OF-THE-ART MOBILE AGENT PLATFORMS 11
2.2 A FLEXIBLE AND MODULAR MIDDLEWARE ARCHITECTURE FOR INTERNET SERVICES 16
2.2.1 Properties 16
2.2.2 The Proposed Middleware Architecture 20
2.2.3 The Java Middleware: Editions, Configurations and Profiles 23
2.2.4 The CORBA Middleware: Broker, Services and Facilities 27

3 THE SOMA MIDDLEWARE CONFIGURATIONS 31

3.1 BASIC CONFIGURATION 34
3.2 COMPLETE CONFIGURATION 35
3.2.1 The Agent Naming Facility 35
3.2.2 The Agent Interoperability Facility 39
3.2.3 The Agent Security Facility 45

4 THE SOMA PROFILE FOR INTEGRATED MANAGEMENT 49

4.1 RELATED WORK 51
4.2 ON-LINE MONITORING 55
4.2.1 Java Technologies for Monitoring 56
4.2.2 The MAPI Component 60
4.2.3 The MAPI Implementation 63
4.2.4 The MAPI Overhead 67
4.3 ON-LINE DISTRIBUTED MONITORING AND MANAGEMENT 69
4.3.1 The On-line Distributed Control of MA Resource Consumption 72
4.3.2 The Performance of Distributed Monitoring 73
4.4 VIDEO ON DEMAND SERVICE MANAGEMENT 74
4.4.1 The MADAMA Performance 78

5 THE SOMA PROFILE FOR MOBILE COMPUTING 81

5.1 RELATED WORK 84
5.1.1 User Mobility 84
5.1.2 Terminal Mobility 85
5.1.3 Mobile Access to Resources 87
5.2 USER VIRTUAL ENVIRONMENT 87
5.3 MOBILE VIRTUAL TERMINAL 89
5.4 VIRTUAL RESOURCE MANAGEMENT 90
5.5 THE PERFORMANCE OF THE MOBILE COMPUTING PROFILE 91
5.6 AN EXAMPLE OF APPLICATION SCENARIO 93

6 CONCLUDING REMARKS 97

6.1 LESSONS LEARNED 97
6.2 FUTURE WORK 98

ACKNOWLEDGEMENTS 101

REFERENCES 103

1 Introduction

The Internet is transforming into an open and global distributed system to provide services to an increasing number of users, interconnected by very different and heterogeneous devices, e.g., PCs, personal digital assistants, and even cellular phones [Lewis, 98]. In addition, the diffusion of multimedia Web services and the competition among service providers stress innovative service properties more and more important for vendors, network operators and final customers. The key property is Quality of Service (QoS), defined as the possibility to grant and guarantee negotiated service levels independently of the dynamic conditions of resources in the involved networks and systems [Chalmers, 99] [Hutchison, 94]. The providers offering services with some controlled and differentiated QoS levels are interested in accounting users for consumed resources and in enforcing effectively the desired billing policies. Users request the possibility to specify and dynamically modify the desired QoS level of their applications, and to control the effective quality achieved during service provision before paying their service providers and network operators. In the untrusted Internet environment, another important service property is security, which permits to identify and face all forms of misuse and attack, such as denial-of-service obtained by overloading resources to cause service unavailability.

Whereas the standardization of communication protocols has made possible to interconnect geographically distributed resources in a unique global system, the extensive usage of the Internet for the implementation and provision of distributed services demands the additional availability of basic common facilities, provided at the infrastructure level, to simplify the interworking between distributed components [Bolliger, 98]. This facility infrastructure is crucial to help application developers in the realization of highly accessible and personalized Web services supporting differentiated levels of quality. In this scenario, applications should consist of and be accessed by interworking resources and service components, geographically distributed and intrinsically heterogeneous due to the openness of the Internet. For instance, any Internet service implemented in terms of cooperating components needs a flexible naming infrastructure to simplify the dynamic identification and location of required entities, even based on a partial knowledge of searched resources (e.g., by knowing their functionality but not their interface). Naming is probably the most evident example of a general-interest facility for distributed systems: single applications that implement their proper *ad hoc* naming service not only force their designers to a considerable implementation effort but, most important of all, lead to competing and incompatible solutions that are in contrast with the reuse and interoperability principles emerging in the Internet environment.

The provision of services in an open, global and mobile environment has significantly stimulated research work on new programming paradigms to enhance the flexibility of the traditional Client/Server model. All proposed paradigms focus on the support of code mobility at runtime. In particular, the Mobile Agent (MA) programming paradigm associates with location-aware computing entities (mobile agents) the possibility of migrating with code and reached execution state from one network host to another one while in execution [Fuggetta, 98]. The property of mobility makes the MA technology more flexible than the traditional C/S one because it can exploit locality in the access to distributed resources and to perform distributed operations in a completely asynchronous way with respect to both commanding users and originating hosts [Fuggetta, 98] [Rothermel, 98] [Kotz, 00].

While first research activities on mobile agents aimed at the definition of models, design principles and methodologies, as the focus has moved to the implementation of MA systems and MA-based complex services, one of the main objectives of state-of-the-art MA platforms is becoming the identification and provision of a common middleware of basic facilities. This middleware should provide application designers with a wide set of general-purpose (i.e. horizontal) facilities, possibly designed according to a flexible and modular layered architecture, to simplify the rapid prototyping of MA-based distributed services and to support their deployment and execution at run-time [Bellavista, 00a].

To summarize, the growing requirements of Web service provision are forcing distributed component technologies towards the definition of a common set of facilities ubiquitously accessible as an integrated part of the communication infrastructure. This trend involves both the traditional area of Client/Server distributed systems and the emerging sector of the MA technology. Middleware facilities not only provide the basis where developers design and deploy their applications, but also should be flexible and easily extensible, in order to accommodate evolving system/service requirements and expectations of final users. This common global infrastructure is the current vision of what is traditionally known as *middleware*.

While there is a general agreement on the necessity of providing middleware solutions for Internet services, researchers hardly agree on the precise definition of what middleware is, which facilities and services are part of it, and which middleware facilities have to be considered either at a lower layer (as components of the network infrastructure) or at a higher one (as application-specific components). In fact, the concept of middleware tends to depend on the different and subjective perspectives of the different implementers. It is even dependent on when the question is asked, since the middleware of yesterday (e.g., Domain Name Service, Public Key Infrastructure and Event Services) may become the fundamental network infrastructure of tomorrow. Final users and programmers usually see everything below the Application Programming Interfaces (API) as middleware. Networking people see anything above IP as middleware. Researchers and practitioners between these two extremes consider middleware somewhere between TCP and the API, and attempt further classification of middleware solutions into application-specific upper middleware, generic middleware, and resource-specific lower middleware [Aiken, 00] [NGI].

To briefly present a historical evolution of the middleware definition, some of its earliest conceptualizations originated within the distributed operating research of the late 1970s and early 1980s and were further advanced by the I-WAY project [Foster, 96]. The I-WAY linked high-performance computers over high-performance networks such that the resulting nation-wide environment worked as a single high-performance system. On the basis of this experiment, the involved researchers re-emphasized the fact that effective high-performance distributed computing required distributed common resources and facilities, including libraries and utilities for resource discovery, scheduling and monitoring, process creation, communication and data transport. In May 1997, the members of the Next Generation Internet (NGI) consortium extended the traditional definition of middleware by stressing requirements such as reusability and expandability [NGI]. Their definition includes a wide set of services at different layers of abstraction, such as traditional operating systems facilities (e.g., run-time support and libraries), distributed programming environments (e.g., the Java framework), and network infrastructure services (e.g., the Domain Name Service).

This thesis has neither the pretension nor the objective of presenting a conclusive definition of middleware. Its aim is, instead, to point out the convergence of research efforts in the MA area towards the provision of a set of horizontal facilities to help in the development and deployment of globally available services. In addition to simple basic facilities for agent identification, migration and communication, the research community starts to recognize that is crucial for MA platforms to provide advanced facilities to answer the increasing requirements of both users and providers in terms of service quality, dynamic personalization and tailoring, rapid application development and deployment.

After a thorough analysis of the related literature and after first experiences in deploying MA-based services in several application domains (e.g., systems management and distributed information retrieval), we have investigated the possibility to design a middleware for Internet services according to the following guidelines:

- *Adoption of the MA technology*. Mobile agents are suitable for the implementation of middleware services in the emerging Internet scenario because of their mobility, autonomy, asynchronicity with regards to their users.
- *Interoperability*. Several research efforts, both in traditional distributed computing and in the emerging sector of mobile agents, are defining different middleware infrastructures of general-purpose facilities. It is crucial to allow these infrastructures to interoperate the one with the other, and to interwork with the large base of installed systems and service components.
- *Flexibility*. Middleware should support very different requirements, possibly evolving during service provision. Middleware facilities should consist of different extensible and composable modules, possibly organized in hierarchical layers dynamically added only when and where specifically required.

This thesis presents the work done to develop, implement and deploy a modular MA-based middleware called Secure and Open Mobile Agents[*] (SOMA). We have

[*] SOMA, related papers and documentation are available at:
`http://lia.deis.unibo.it/Research/SOMA/`

implemented the SOMA middleware as a layered architecture of general-purpose horizontal facilities, defining SOMA configurations, and of domain-specific vertical facilities, defining SOMA profiles. These facilities can be installed dynamically depending on location-specific and service-specific runtime conditions.

With regards to existing middleware approaches, mobile agents are a suitable technology to enable a new perspective focused on the mobility of computing entities, service users and system resources, and require both developing new solutions from scratch and adapting existing components. We have tried to follow both directions, by obtaining very useful feedback by the direct exploitation of SOMA middleware facilities in the implementation of MA-based service prototypes in several application domains, from distributed monitoring and control of network equipment to QoS management of distributed multimedia services, from automatic personalization of Web services according to mobile users' preferences to re-qualification of bindings to needed resources and services in the case of terminal mobility.

Beginning with an analysis of emerging architectures of the state-of-the-art MA platforms, Chapter 2 defines the concepts of configuration and profile, and briefly describes the notable example of the Java middleware with its current different editions. Then, the chapter presents our proposed layered architecture of extensible and modular components, organized in two possible configurations (the basic configuration and the complete one) described in Chapter 3. In particular, Chapter 3 focuses on the design and implementation of the interoperability facility of the complete configuration, which has been the result of the first year of the doctorate work. Over the configuration layer, the SOMA middleware provides two different profiles of facilities that are oriented towards domain-specific scenarios.

Chapter 4 presents the SOMA profile for the integrated management of network elements, legacy systems and service components, either MA-based or not. The chapter presents the implementation of a Java-based on-line monitoring facility for distributed heterogeneous components. The monitoring facility is the basis to trigger management operations performed autonomously by mobile agents depending on runtime conditions. The ultimate goal is to permit the dynamic control of service quality and its adaptation in response to the current state of distributed resources and to the current user requirements.

Chapter 5 describes the SOMA profile for the support of the several different forms of mobility that are emerging in the Internet scenario. User mobility requires the collection and automatic migration of user preferences in order to provide users with a uniform interface to their usual working environments. Terminal mobility calls for the possibility to maintain network connectivity transparently to the current location of mobile devices. Resource mobility imposes to support the automatic re-configuration of needed bindings when networked resources and service components have the possibility to move during service provision. Both SOMA profiles are described together with the experience coming from the implementation of SOMA-based service prototypes on top of them, and with the presentation of the experimental measurements of the costs of the adopted solutions.

Finally, Chapter 6 summarizes the lessons learned from the accomplished work, and sketches guidelines for the further evolution of the SOMA middleware, in terms of both the extension/refinement of middleware facilities and the deployment of new service prototypes for the ubiquitous access to distributed heterogeneous information in virtual museums.

2 Mobile Agent-based Infrastructures for Internet Services

The appearance of the Mobile Agent (MA) concept can be derived mainly by a new technology called TeleScript developed by General Magic in 1994 [White, 94]. It was the period when scripting languages, such as the Tool Command Language (TCL) and its derivative SafeTCL [Borenstein, 94] gained much attention, since they enabled rapid prototyping and the generation of portable code. The concept of smart messaging [Kisielius, 97], based on the encapsulation of SafeTCL scripts within emails [Borenstein, 94], made new mail-enabled applications possible. In the same years, the concept of mobile computing, intended as the possibility of moving users and terminals in the Internet with no need to suspend service provision, has gained increasing importance and has further stimulated the research on mobile code technologies [Chess, 95]. Last but not least, it was the beginning of the Java age, and Java is the basis for the largest part of current MA systems.

In the last years, the MA technology has achieved a large and general interest, and has been the object of several development efforts, much of which has evolved from the platform independence of the Java language with its object serialization and network communication support. We can summarize that, based on this coincidence of the appearance of various agent concepts, agents become a fashion technology for both the academic and industrial development communities. However, this also created confusion, since there was a lack of common definitions and standards, resulting in various concepts, languages, architectures, technologies and terminology.

Mobile agents are based on the principle of code mobility. In the Client/Server (C/S) paradigm, the server is defined as a computational entity that offers some functionality: the client requests the execution of these services by interacting with the server; after the service is executed, the result is delivered back to the client. Therefore, the server provides the knowledge of how to handle the request as well as the necessary resources. Mobile code enhances the traditional C/S paradigm by performing changes along two orthogonal axes:

- Where is the know-how of the service located?

- Who provides the computational resources?

Three main programming paradigms based on the possibility of dynamic code migration have been identified [Fuggetta, 98]: *Remote Evaluation* (REV), *Code on Demand* (CoD), and *Mobile Agents*. These paradigms differ in how the know-how, the processor and the resources are distributed among the components S_A and S_B of a distributed system (see Figure 2.1). The know-how represents the code necessary to accomplish the computation. The resources (i.e., the file system where application-specific data are stored) are located at the machine that will execute the specific computation.

In the REV paradigm [Stamos, 90], a component A sends instructions specifying how to perform a service to a component B. For instance, the instructions can be expressed in the Java bytecode. B then executes the request on its local resources. Java Servlets are an example of REV [Gosling, 97]. In the CoD paradigm, the same interactions take place as in REV. The difference is that component A has resources located in its execution environment but lacks the knowledge of how to access and process these resources. It gets this information from component B. As soon as A has the necessary know-how, it can start executing. Java applets fall under this paradigm.

Paradigm	**Before**		**After**	
	S_A	S_B	S_A	S_B
Client/Server	A	Know-how Resource B	A	Know-how Resource B
Remote Evaluation	Know-how A	Resource B	A	Know-how Resource B
Code on Demand	Resource A	Know-how B	Resource Know-how A	B
Mobile Agent	Know-how A	Resource	---	Know-how Resource A

Figure 2.1: Taxonomy of programming paradigms based on code mobility [Fuggetta, 98]

It is possible to think of the MA paradigm as an extension of the REV one [Fuggetta, 98]. Whereas the latter primarily focuses on the transfer of code, the MA paradigm involves the mobility of an entire computational entity, along with its code and the reached execution state. In other words, if component A is a mobile agent, it has the know-how capabilities and a processor, but it lacks the resources where to perform its operations. The computation associated with the interaction takes place on component B that has a processor and the required resources. For instance, a client owns the code to perform a service, but does not own the resources necessary to provide the service. Therefore, the client delegates the know-how to the server where the know-how will gain access to the required resources and the service will be provided. An entity encompassing the know-how is a mobile agent. It has the ability to migrate autonomously to a different computing node where the required resources are available. Furthermore, it is capable of resuming its execution seamlessly, because it preserves its execution state.

This means that a mobile agent is not bound to the network host where it begins execution. The ability to travel permits a mobile agent to move to a destination agent system that contains the resources with which the agent wants to interact. Moreover, the agent may be interested in exploiting the services offered by the destination agent system. When an agent travels, its state and code are transported with it. The agent state can be either its execution state or agent attribute values that determine what to do when execution is resumed at the destination agent system. The agent attribute values can include the agent system state associated with the agent (e.g., time to live).

The MA paradigm is showing its suitability for the provision of middleware facilities for Internet services because it represents an alternate, or at least complementary, solution to traditional C/S models of interaction [Chess, 95]. MA solutions may contribute to a reduction of the overall communication traffic in network. For example, mobile code has the ability to engage with a server locally for searching large databases; the proximity to the server ensures high communication bandwidth.

The adoption of the MA technology is encouraged by many researchers in the distributed system area, by citing the following deriving benefits [Chess, 98]:

- *Asynchronous/autonomous task execution* – After the injection of an agent into the network environment, both the commanding user and the originating network host have no control duties on the launched agent and can perform other tasks. In addition, agents can exhibit an autonomous behavior and can have built-in itineraries which determine which tasks they have to perform and where.

- *Reduction of network traffic and client processing power* – Both massive and frequent data exchanges are handled locally at the nodes hosting the data, and client computers could concentrate on performing only limited local tasks.
- *Flexibility and extensibility* – Software can be distributed at run-time and only when needed (on-demand software distribution). Service software can be encapsulated within mobile agents, instantly downloaded to the client side, to the server side and to any intermediate node involved, and installed by transporting agents even when complex and time-/contect-dependent configuration operations have to be performed on target hosts.
- *Decentralized control and management* – Dynamic agent migration and the possibility to multiply the agent instances via cloning significantly simplify the automated distribution of formerly centralized programs for the control and management of network resources and distributed applications.
- *Increased robustness* – The reduction of dependence between interworking components allows MA-based applications to overcome temporary unavailability of both the network and the needed C/S resources. Once the agent arrived at a target system, the originating host may crash or the network may become unavailable without any drawbacks on the task processing.

This means that mobile agents provide flexibility in dynamically (re-)distributing intelligence inside a distributed network environment, in particular to reduce network load and to optimize service performance. The MA benefits listed above permit to overcome various problems and inefficiencies of traditional C/S architectures, especially when dealing with application scenarios intrinsically mobile, such as users that modify their terminal of attachment to the Internet and portable devices that roam from one hosting network to another one. In addition, QoS requirements imposes on-line adaptive operations that can significantly profit from the possibility to migrate agents locally to the resources to be managed, as extensively described in Chapter 4.

The possible drawback of the MA technology is represented by the security risks introduced by the necessity to host the execution of new computing entities that carry their own code. Furthermore, an agent may be attacked, modified or deleted by a hostile agent platform on a malicious network host. Another typically stated and obvious concern related to mobile agents is the question if agent migration is always of advantage if compared with message passing. For example, it is probably better to interact by message passing in case the agent code is bigger than the expected data volume to be exchanged.

In summary, it has to be stated that the MA technology has a lot of appealing advantages compared to traditional technologies for solving specific requirements that are emerging in the provision of distributed services in the Internet environment. But the agent support implies the introduction of middleware components in the target environment in order to enable mobility, local agent execution and inter-agent communication. In addition, to be effectively usable in the short term, mobile agents require mechanisms and tools to interact with existing services and legacy systems designed according to the traditional C/S programming paradigm.

2.1 Emerging Architectures in State-of-the-art Mobile Agent Platforms

Several research laboratories and industrial manufacturers were involved in the development of various MA platforms since 1996. Those platforms were built on top of different operating systems, and based on different programming languages and technologies. Even new languages have been realized, exclusively designed for the support of mobile agents (e.g., TeleScript).

However, what is more relevant from the middleware perspective is that common trends in MA platforms have started to emerge evidently within the last couple of years. Interpreter-based programming languages, particularly Java, are forming the basis for most of today's agent platforms, mainly due to their easy portability over heterogeneous platforms. In addition, Java is frequently chosen for the available support, directly at the language level, of fine-grained security policies and transport facilities via object serialization [Gosling, 97]. Moreover, even if coming from different experiences and domain-specific design constraints, the implementers of MA platforms are achieving a general agreement on the architecture of middleware services that are necessary for supporting mobile agents in open and global environments and for leveraging the diffusion of MA-based services in the Internet. Finally, several approaches have recently explored the possibility of integrating mobile agents and RPC-based middleware like the OMG Common Object Request

Broker Architecture (CORBA), possibly stimulated by the research work accomplished for the definition of agent interoperability standards [Zhang, 98].

Good examples of state-of-the-art MA systems are Aglets Workbench from IBM Japan [Lange, 98], Concordia from Mitsubishi [Concordia], Odyssey from General Magic [Odyssey], Voyager from ObjectSpace [Voyager], Ajanta from the University of Minnesota [Ajanta], TACOMA from Universities of Cornell and Tromso [TACOMA], Grasshopper from IKV++ GmbH [Grasshopper], and SOMA from University of Bologna [SOMA]. An extensive description and comparison of MA platforms is out of the scope of this thesis and can be found in [Perdikeas, 99].

MA platforms typically realize a distributed processing environment, consisting of several middleware components, and usually referred to as *Distributed Agent Environment* (DAE). DAEs usually support a hierarchy of locality abstractions (domains, agent systems and places) to model network physical resources, and two different types of agents (mobile and stationary). Figure 2.2 depicts an abstract view of these entities.

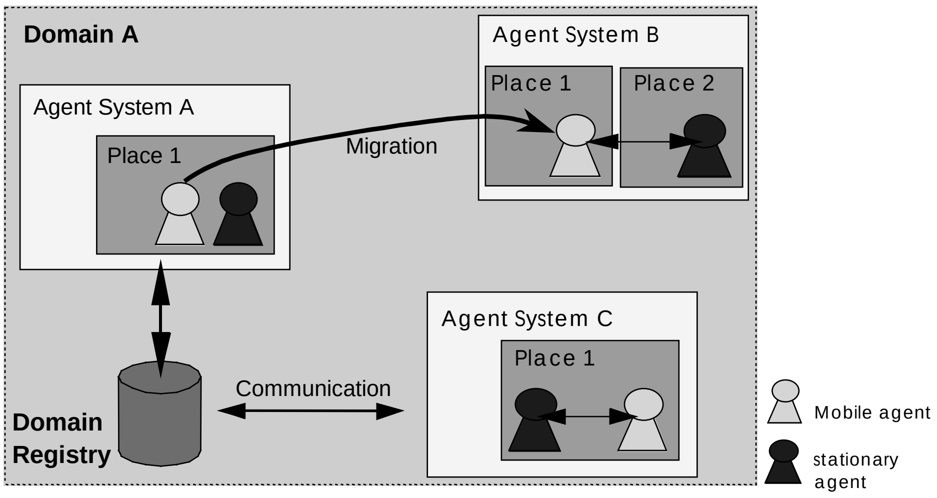


Figure 2.2: Structure of a Distributed Agent Environment (DAE)

The actual runtime environment for agents is called *agent system*: on each network host, at least one agent system has to run to support the execution of agents. Agent systems are structured by means of places, i.e., isolated execution contexts that are able to offer specific additional services. For example, there may exist a communication place offering sophisticated communication features, or there may be a trading place where agents offer/buy information and service access. Agent systems can be grouped into domains that usually model a physical local area network: each domain has an associated (domain) registry that maintains information about all registered agent systems, places and agents.

The domain locality abstraction facilitates the management of the distributed components (agent systems, places and agents) in the DAE and improves its scalability. The domain registry maintains information about all components that are associated to a specific domain. When a new fixed/mobile entity is created, it is automatically registered within the corresponding domain registry. While agent systems and their places are generally associated to a single domain for their entire lifetime, mobile agents are able to move between different agent systems of possibly different domains. The current location of mobile agents is updated in the corresponding domain registry after each migration. By contacting the domain registry, other entities (e.g., agents or human users) are able to locate agents, places, and agent systems residing in a domain.

Two different types of agents are defined. Mobile agents are able to move from one physical network location (agent system A in Figure 2.2) to another one (agent system B). Stationary agents, instead, are bound to the agent system where they have been installed and where they remain for their whole lifetime to provide a place persistent service according to the C/S model of interaction.

Several fundamental requirements have been identified due to the experiences made during research and development activities in the MA area. These requirements are fulfilled by any state-of-the-art MA platform, and their identification is the first fundamental step towards the definition of a common and interoperable distributed middleware to support mobile agents in the Internet scenario. The implementation of a modern MA system requires middleware components to support:

- *Agent execution*. An MA platform must provide the basic capability to put incoming mobile agents into execution, taking into account possible agent-specific requirements regarding the runtime environment (e.g., binding to specified local resources). The platform has to retrieve the agent code that may be either delivered with the migration request or downloaded separately from an external code base.

- *Transport*. A special mobility support must be provided by the platform, not only to facilitate the network transport of agent code and execution state, but also to permit MA system administrators to command remote execution and migration. Note that both agent execution and transport cannot be sufficiently handled without a strict interworking with the security support mentioned in the following.
- *Unique identification*. Mobile agents as well as agent systems have to be uniquely identifiable in the scope of the entire Internet environment. Thus, special support is required for the generation of unique agent and agent system identifiers.
- *Communication*. Agents should be able to communicate with each other as well as with platform services. Several mechanisms are possible, such as messages, method invocation, object sharing and tuple-spaces, with different levels of expressive power and of temporal/spatial coupling between coordinating entities [Cabri, 00]. Communication through messages may be done point-to-point, by multicasting, or by broadcasting.
- *Security*. Basic issues are authentication (i.e., the determination of the identity of an agent or an agent system), and access control of resources/services depending on the authenticated identity of the requesting entity. To guarantee privacy and integrity, crucial information such as code and state of a migrating agent should exploit public-key cryptographic encryption before transfer over an untrusted network.
- *Management*. It is necessary for agent administrators to be able to remotely monitor and control mobile agents and MA-based provided services. Control functions include temporary interruption of the execution of an agent task, agent premature termination, and modification of its task list. The monitoring of an agent is associated with its localization in the scope of the whole distributed environment. Regarding an agent system, all hosted agents as well as the occupied system resources have to be monitored and controlled, possibly to notice and avoid denial-of-service attacks.

Figure 2.3 shows the structure of a core agent system that includes several services in order to fulfil the basic functional requirements identified above. Note that some services provide remote interfaces in order to be accessible by external actors, such as other agent systems, agents, or human users.

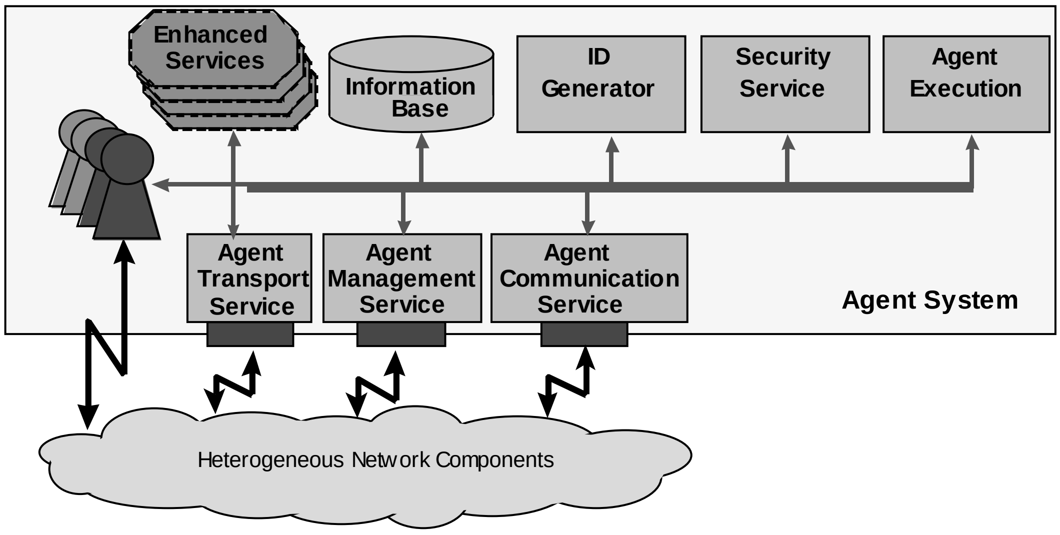


Figure 2.3: Architecture of basic facilities in mobile agent platforms

Apart from the basic capabilities shown in the figure, additional ones have been taken into consideration in some of the most recent MA platforms [Grasshopper] [SOMA]. For instance, an interoperability module is offered to permit the integration of heterogeneous agents and agent systems with already existing services and legacy components. Interoperability is obtained via compliance with emerging standards in the MA area, all based upon the CORBA middleware, and the implementation of the interoperability facility in SOMA will be described in Section 3.2.2.

Other capabilities have started to be accepted as fundamental and tend to be integrated in MA platforms. The persistency facility, for example, permits to temporarily suspend executing agents and to store them on a persistent medium. Persistency allows agents not to waste system resources while they are waiting for external events such as the reconnection of one user or terminal where they have to yield back the results of their operations. In addition, a facility for the mobile computing support is provided in some MA systems to accommodate the nomadicity of users, terminals and service components, which can move with no need to suspend offered/accessed services [Bellavista, 00b]. These additional features can significantly benefit from the modular organization of MA platforms and should be handled as add-ons that can be "plugged" into a core MA system to dynamically extend its basic functionality.

2.2 A Flexible and Modular Middleware Architecture for Internet Services

The adoption of new and more flexible technologies for distributed programming such as the MA paradigm is not sufficient in itself to face up with the complexity of providing Internet services capable of answering the increasing requirements of both users and providers. This complexity calls for the availability of rich and flexible general-purpose middleware facilities that provides service developers with a large set of APIs to support the design, implementation and deployment of ubiquitously accessible services with differentiated QoS levels in the global heterogeneous environment. These middleware facilities should permit service developers to abstract from the duties of horizontal infrastructure solutions, and to focus their attention and competence only on domain-specific issues. The key relevance of providing middleware facilities for Internet services is widely recognized in the distributed systems community [Sventek, 00] [Schmidt, 00] [Bellavista, 01b].

2.2.1 Properties

Middleware infrastructures should offer a wide set of facilities to develop and support distributed services at runtime in a uniform way, independently of the possible heterogeneity of distributed resources and systems involved. Middleware facilities should compose an integrated framework where they interwork to give service developers the possibility to choose, at design time and at execution time, the most suitable application-specific trade-off between contrasting requirements, such as completeness and efficiency of the support solution.

We claim that the main aim of a middleware infrastructure is to provide the widest set of the following properties, to face the new challenges of service provision in globally interconnected networks.

Dynamicity

New protocols and services should be introduced dynamically, in order to answer application-, session- or user- specific requirements; for instance, a customized multicast protocol can be deployed at runtime to support a teleconference service.

Adaptability
Services should adapt to current network situation and should evolve with user requirements with no need to suspend during their phase of tuning; for instance, a specific level of QoS can be renegotiated when congestion makes no longer possible its provision.

Interoperability
Services should take advantage of any other existing service and resource, from legacy systems to CORBA-compliant services, from diffused management components (based on standard protocols as SNMP and CMIP [Stallings, 98] [ISO, 92]) to any other resource that exploits possibly different emerging standards.

Distribution of service control
In general, services are not furnished by one predefined service provider, but by several providers allocated anywhere in the system. This motivates the need for distributing replicated service controllers to avoid the bottleneck of centralized management; for instance, a distributed control could be more efficient for a geographically distributed teleconference service.

Coordination of services and service providers
Any service can coordinate with other ones to negotiate any required strategy, and different providers can cooperate to offer coordinated services; for instance, only the coordination between network operators and multimedia providers can guarantee a specified bandwidth and latency in a video on demand service.

Security
Services are distributed in a global and untrusted environment, shared among a multitude of users and providers, in a scenario that imposes strong requirements for security; for instance, a specific service should be available only to recognized users, and any service should choose the proper balance between efficiency and required security level.

In addition, serving the wide market of Internet services calls not only for the availability of suitable middleware facilities, but also for a large measure of flexibility in how these facilities are organized in the middleware architecture. This flexibility is required because of:

- increasing and evolving expectations of users for service personalization and differentiated levels of QoS;
- increasing and evolving requirements of service providers for the monitoring, control and management of distributed service components, and for the accounting of effective resource consumption of their customers;
- the diverse range of the application domains involved;
- the large base of already installed systems and services (*legacy components*);
- the heterogeneity of the large range of device types and hardware configurations, especially when considering portable devices;
- the constant and rapid evolutions in device and communication technologies.

On the one hand, middleware facilities should evolve dynamically, possibly without imposing any suspension in service provision. On the other hand, the complexity and heterogeneity of service requirements impose to provide middleware solutions obtained by the composition of separated modules specialized in providing specific service properties and for specific domains of application. We have applied thoroughly the guidelines of middleware extensibility and modularity as solution strategies to the increasing complexity of service provision in the Internet scenario [Sventek, 00].

A similar trend towards flexibility in terms of dynamic extensibility and modularity can be also noticed in the users' approach to the market of e-services. Users tend to purchase economically priced devices/services with basic functionalities, and then expand them in increasingly complex solutions. In addition, the market of connected consumer devices such as cell phones, pagers, personal organizers, and set-top boxes, is going to become the dominant force in the Internet. These devices are highly heterogeneous in form, function and features, and can hardly support the access to services that have been designed and implemented for the computation capabilities and the network connectivity of standard desktop computers. This diversity imposes different versions of the same service, tailored to different device characteristics, or, more effectively, to design Internet services with different levels of accessibility and functionality. Each service level derives from the interaction of simpler and reusable components [Bolliger, 98].

We claim that all the above mentioned elements call for the availability of a middleware infrastructure built as the composition of small size facility modules that can be organized and configured in a widely customizable way. For instance, resource-limited portable devices can generally support minimal configurations of the middleware facilities. Service developers access APIs with only essential capabilities of each kind of device and the achievable subset of service properties. As device manufacturers develop new features in their devices, or service providers develop new applications, minimal configurations can be expanded with additional APIs and with the corresponding richer complement of the middleware facilities.

To support customizability and modularity, the design of middleware architectures can significantly benefit from an organization in terms of configurations and profiles.

A middleware ***configuration*** defines a minimum horizontal platform for a category of devices or a class of service features, each with similar resource and system requirements, e.g., processing power, memory budget and available network bandwidth. A configuration defines the middleware facilities and minimum APIs for service developers that a device manufacturer or a service provider can expect to be available on all devices of the same category.

A middleware ***profile*** is layered on top of (and thus extends) a configuration. A profile consists of middleware facilities that addresses the specific demands of a specific vertical segment of the service market or a specific device family. The main goal of a profile is to guarantee interoperability within a subset of devices/services by defining standard middleware facilities for that market. Profiles typically include facilities and corresponding APIs that are far more domain-specific than the facilities/APIs provided in a configuration. Whenever possible, the facilities included in one profile should not belong to any other profile, in order not to increase the dimension of the middleware with redundant components and functionality.

It is possible for a single device to support several profiles and for a service to require the support of several profiles. Some of these profiles can be very device-specific, while others can be more application-specific. Applications are written for their specific profiles and are required to use only the features defined by those profiles. The value proposition to the consumer is that any application written for a particular profile will run on any system that supports that profile. Thus, portability is achieved between the applications and the devices served by that profile. New devices can take advantage of a large and familiar application base. Most importantly, new Internet services (perhaps completely unforeseen by the original profile designers) can be dynamically downloaded to already installed and possibly heterogeneous systems, with the constraint to employ only the APIs supported by the profile.

To clarify the introduced concepts of middleware configurations and profiles, let us consider some examples of middleware facilities and their organization in a modular layered architecture. For instance, a naming service, which retrieves a needed resource on the basis of either its logical location-independent name or the values of some research attributes, is a typical general-purpose horizontal facility included in a middleware configuration. On top of this configuration, developers may require to deploy two different classes of services: e-commerce services and Computer Supported Collaborative Work (CSCW) services. The first class can benefit from the availability of middleware APIs including the support for transactions, negotiation protocols and auction protocols. The latter class requires the provision of facilities, for example, for event distribution and versioning management. The two service classes identify different domain-specific middleware facilities that are not so general-purpose to justify their inclusion in any distribution of the middleware. These facilities should belong to two different profiles, each one to be installed only when and where there are requests for provision of an e-commerce/CSCW service.

2.2.2 The Proposed Middleware Architecture

According to the guidelines presented in the previous section, we have defined a middleware architecture of facilities that should simplify the duties of service developers in the design, implementation and deployment of Internet services. The architecture is organized in layers, as depicted in Figure 2.4. Services at the application layer are implemented by exploiting the underlying layer of one or more domain-specific profile facilities. In their turn, profile facilities employ general-purpose facilities belonging to one configuration.

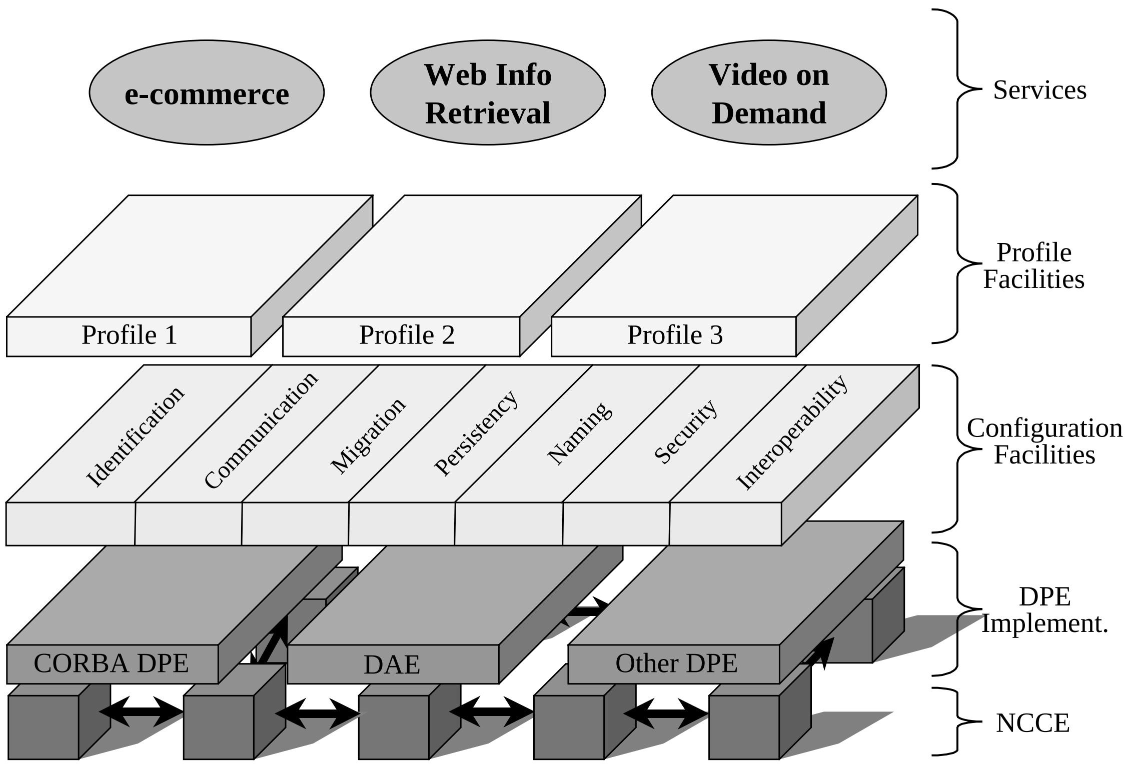


Figure 2.4: Our layered middleware architecture to support Internet services

Profile and configuration facilities are supported by the implementations of the underlying Distributed Processing Environments (DPE), but should not rely on a specific one to provide flexible solutions. For instance, a naming facility can be built on different naming systems provided by different DPE implementations, e.g, DNS-, CORBA-, and LDAP-compliant naming services [Albitz, 98] [OMG, 98] [Howes, 97]. Our architecture suggests several DPE implementations to coexist. For instance, the figure depicts the case with a CORBA DPE, a mobile agent-based DPE (i.e., a Distributed Agent Environment - DAE), and a generic other proprietary DPE. Any DPE, in its turn, abstracts and hides specific details of the underlying, possibly heterogeneous, Native Computing and Communication Environments (NCCE).

Each configuration facility answers specific general-purpose horizontal issues, and can also interact with other facilities, e.g., naming and security facilities interwork when the system has to authenticate an entity and to recognize its role [Lupu, 97]. Our architecture proposal identifies a single non-layered middleware configuration with the following set of facilities:

- the ***identification*** facility permits to tag resources, users and services, by assigning unique names to entities in the system;
- the ***migration*** facility is in charge of transporting one entity that should change its allocation from its sending node to the destination one. The reallocated entity, if it is active, should transparently restart its execution at the new location;
- the ***communication*** facility supports any exchange of information between service components, and is capable of eventually delivering messages to reallocated entities;
- the ***persistency*** facility permits to transparently save service components, whether MA-based or not, to stable storage, for the sake of minimizing the consumption of execution resources and of enabling fault-tolerant solutions;
- the ***naming*** facility accommodates and organizes all names of public entities and makes possible to search and trace them, also in case of their migration;
- the ***interoperability*** facility permits interoperation among different resources and different service components, designed with any programming style, by closely considering conformance with accepted standards;
- the ***security*** facility protects any entity in the system, by providing a wide range of mechanisms and tools for authentication, authorization, controlled access to all resources and services, privacy, and integrity.

Although some middleware solutions could neglect migration and the corresponding facility, we consider the possibility of reallocating entities as a basic feature when dealing with open and dynamic systems. Apart from the pioneer MbD work in the specific application domain of network and systems management [Goldszmidt, 95], the recent research activities that exploit both MA and AN technologies adopt this perspective [Chen, 98] [Bieszczad, 98].

The proposed architecture organizes all horizontal facilities in a single configuration. However, depending on the particular implementation of configuration facilities, it is sometimes preferable to split those facilities in subsets at different levels of abstraction. For instance, the naming facility can employ the identification facility to build more complex and user-level naming systems on the basis of the lower level identifiers. If and only if the facility subsets are disjoint and hierarchically organized (i.e., facilities in one subset are servers and not clients of facilities in another subset), the subsets identify different middleware configurations. In particular, Chapter 3 will describe how we have mapped our architecture proposal into an MA-based middleware; in that implementation, configuration facilities are split on two distinguished subsets that identify two different configurations, the SOMA basic one and the SOMA complete one.

Different profiles are intended to support service provision in different application domains. Profile APIs provide features that are either domain-specific or directed to a certain category of devices. For instance, this doctorate has mainly aimed at investigating the domains of integrated management and of mobile computing, and the work has leaded to the design and implementation of two different profiles for the SOMA middleware, which are presented, respectively, in Chapter 4 and Chapter 5. Even if in our architecture Internet services generally execute on top of a single profile, it is possible for different profiles to interwork in order to provide the support for classes of services that need the middleware APIs typical of different application domains. From another perspective, the idea is that profiles can be also composed in new profiles when required, to obtain maximum reusability of middleware components and rapid deployment of new services.

Let us end the chapter by clarifying the introduced concepts of middleware facilities, configurations and profiles via the notable examples of the widely diffused Java and CORBA middleware architectures, briefly sketched according to this perspective in the following two sections. The Java distributed programming environment is available in different editions, each one consisting of different configurations and profiles [SUN]. The CORBA distributed infrastructure provides service developers with a wide set of APIs (CORBA Services and Facilities) to support both horizontal features and vertical ones [OMG, 98].

2.2.3 The Java Middleware: Editions, Configurations and Profiles

Recognizing the increasing complexity and heterogeneity of the service provision scenario in the global Internet environment (“one size does not fit all”), SUN has decided to group its Java middleware technologies into three editions: the Java 2 Enterprise Edition (J2EE), the Java 2 Standard Edition (J2SE), and the Java 2 Micro Edition (J2ME), as depicted in Figure 2.5.

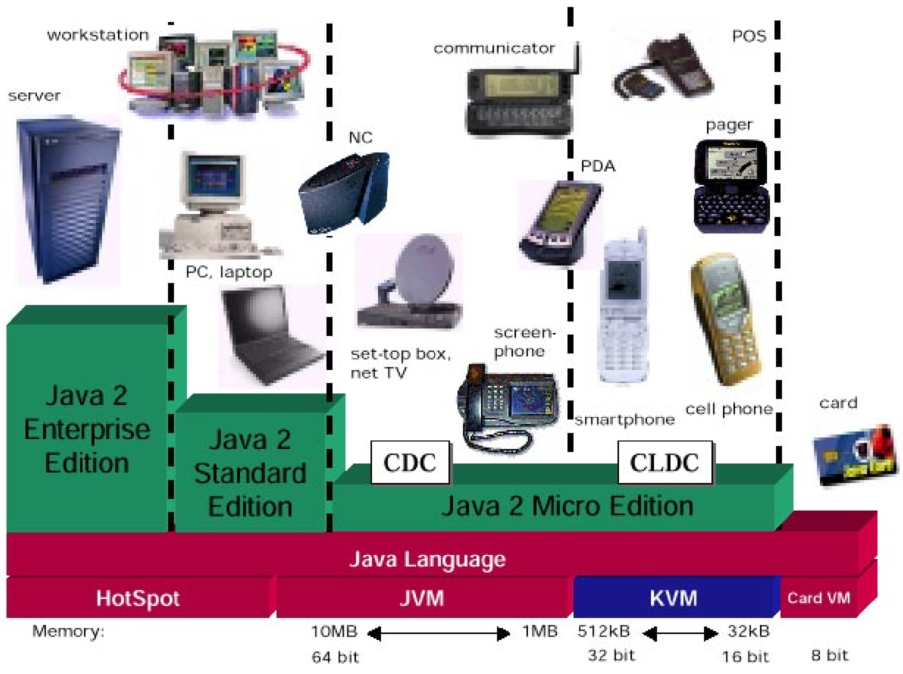


Figure 2.5: The three different editions of the Java middleware

Each Java edition is aimed at a specific category of devices and includes three layers of facilities built upon the host operating system:

1. ***Java Virtual Machine Layer***. This layer is an implementation of a Java Virtual Machine (JVM) that is customized for the characteristics of a particular device or host operating system, and supports a particular configuration.
2. ***Configuration Layer***. The configuration layer is less visible to users, but is very important to profile implementers. It defines the minimum set of JVM features and Java class libraries available on a particular category of devices representing a particular horizontal market segment. In other words, a configuration defines the lowest common denominator of the Java platform features and libraries that service developers can assume to be available on all devices.
3. ***Profile Layer***. The profile layer is the most visible layer to users and application providers. It defines the minimum set of APIs available on a particular family of systems representing a particular vertical market segment. Profiles are implemented upon a particular configuration. Applications are written for a particular profile and are thus portable to any device that supports that profile. A system can support multiple profiles.

The most interesting edition of the Java middleware in terms of flexibility and customizability is certainly the recently introduced J2ME, which extends the scope of the Java technology to meet the demand for information appliances in the consumer and embedded markets. The J2ME architecture currently has two configurations: the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLDC). They are the result of the standardization process of configuration specifications that has involved a large community of device manufacturers, network operators and service developers [CLDC].

The CDC runs on top of the classic JVM, a full-featured virtual machine residing on a desktop system. This configuration is intended for devices with at least a few megabytes of available memory.

For wireless devices and other systems with severely constrained memory environments, J2ME uses the CLDC [CLDC]. The CLDC configuration executes on top of a specific virtual machine (the SUN Kilobyte Virtual Machine - KVM) suitable for devices with 16-bit microprocessors/controllers clocked as low as 25 MHz, and with as little as 160 KB of total memory available. The CLDC provides a set of general-purpose middleware facilities appropriate for supporting an industry-defined profile for wireless devices, called Mobile Information Device Profile (MIDP), briefly presented in the following.

The J2ME middleware provides also the concept of profiles to make it possible to define Java platforms for specific vertical markets. Profiles can serve two distinct portability requirements. On the one hand, a profile provides a complete toolkit for implementing applications for a particular kind of device, such as pagers, set-top boxes, cell phones, washing machines, or interactive electronic toys. On the other hand, a profile may also be created to support a significant, coherent group of applications that might be hosted on several categories of devices. For example, while the differences between set-top boxes, pagers, cell phones, and washing machines are significant enough to justify creating a separate profile for each, it might be useful for certain kinds of personal information management or home banking applications to be portable to each of these devices. This could be accomplished by creating a separate profile for these kinds of applications and ensuring that this new profile can be easily and effectively supported on each of the target devices along with its normal more device-specific profile.

At the moment, the Java community has terminated the process of specification and standardization of one profile called MIDP in the J2ME [MIDP]. The MIDP APIs are tailored to target devices characterized by severe limitations in total memory, power (often battery-powered), connectivity (often with a wireless intermittent connection with 9600-bps bandwidth), and user interfaces. MIDP extends the facilities provided by the CLDC and the KVM with APIs mainly for user interfaces, persistent storage and networking. For instance, MIDP includes class libraries to support a subset of the HTTP protocol; these libraries work on top of the `GenericConnection` framework of the CLDC, which addresses the communication issues at the lower layers of the OSI protocol stack [MIDP].

Figure 2.6 shows the J2ME architecture proposal for the provision of Internet services to wireless devices. Service developers can deploy applications either based on the MIDP or dependent of the device operating system. MIDP-based applications can exploit the MIDP APIs but also have visibility of the facilities directly provided by both the KVM and the CLDC. These applications are portable (and dynamically downloadable) on any device that supports the MIDP. The interworking between MIDP-based portable applications and system-dependent ones in the context of a single device can be achieved via specific mechanisms for the JVM integration with native code, such as the Java Native Interface (JNI) described in Section 4.2.1.

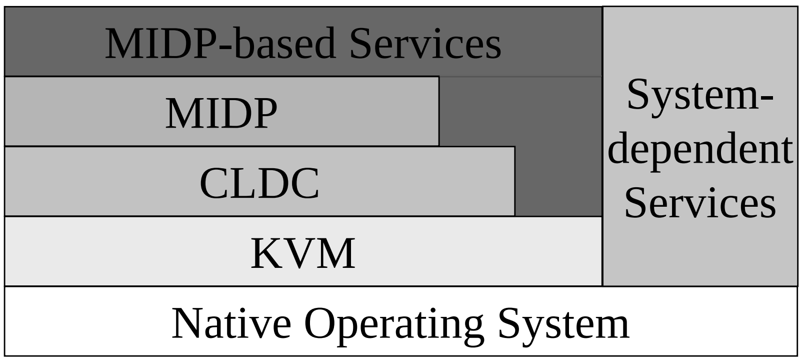


Figure 2.6: The CLDC-based architecture of the J2ME middleware

2.2.4 The CORBA Middleware: Broker, Services and Facilities

Among the middleware supports for distributed objects, CORBA is certainly the most widespread, complex and mature infrastructure, also because the major competing support, Microsoft Distributed Common Object Model (DCOM), has started later and with different aims in terms of openness and generality [OMG, 98] [Sessions, 97] [Chung, 98].

CORBA is a middleware that permits a very rich variety of interworking modalities between its components. The architecture is based on the concept of a common software bus allowing for distributed object interoperability and providing a wide set of bus-related facilities to interacting objects. The main aim is to release application developers from all the duties stemming from the possible heterogeneity and distribution of C/S components. CORBA client objects have no visibility of the location of the CORBA server objects they collaborate with, and their interaction is completely independent of both their implementation language and the platform where they are running. The most notable consideration in this context is that CORBA, apart from the variety of available policies for communication and coordination between components, has been designed from the beginning as a complex and layered architecture of facilities and services to support the implementation of distributed C/S applications.

Differently from the Java middleware, CORBA does not assume a homogeneous programming language environment based on a common virtual machine for service execution. The independence of both the implementation language and the execution platform is guaranteed in CORBA by the specification of a standard language to define object interfaces, the Interface Definition Language (IDL). CORBA server objects that publish their IDL interfaces can be invoked by any CORBA client object that has an a priori knowledge of server interfaces, by exploiting static mechanisms of interaction based on pre-compiled proxies both on the client side (stubs) and on the server one (skeletons). It is also possible to have more flexible and sophisticated modalities of interaction between CORBA objects, based on the availability of dynamic invocation mechanisms that permit to defer the knowledge of the IDL interfaces of the involved objects. A CORBA client can dynamically build an invocation request from scratch by exploiting the Dynamic Invocation Interface (DII). A CORBA server can analogously be unaware at compile-time of its skeleton by exploiting the Dynamic Skeleton Interface (DSI) functionality. Any possible combination of static/dynamic clients invoking static/dynamic servers is supported. In any case, object interactions in CORBA are mediated by run-time facilities for object activation on the server side (Basic/Portable Object Adapter - BOA/POA) and for dynamic retrieval of information on currently available CORBA interfaces (Interface Repository - IntR) and CORBA object implementations (Implementation Repository - ImpR).

All the above mechanisms are components of the CORBA core facilities, i.e., the software bus that realizes the transparency of allocation and implementation between CORBA objects and that is called Object Request Broker (ORB). Upon the ORB core, the OMG specifies the possibility to implement a layered architecture of additional modules to help in the design and deployment of distributed applications. These facilities are organized into a structured middleware architecture called Object Management Architecture (OMA), and classified as CORBA services, horizontal/vertical facilities and application objects.

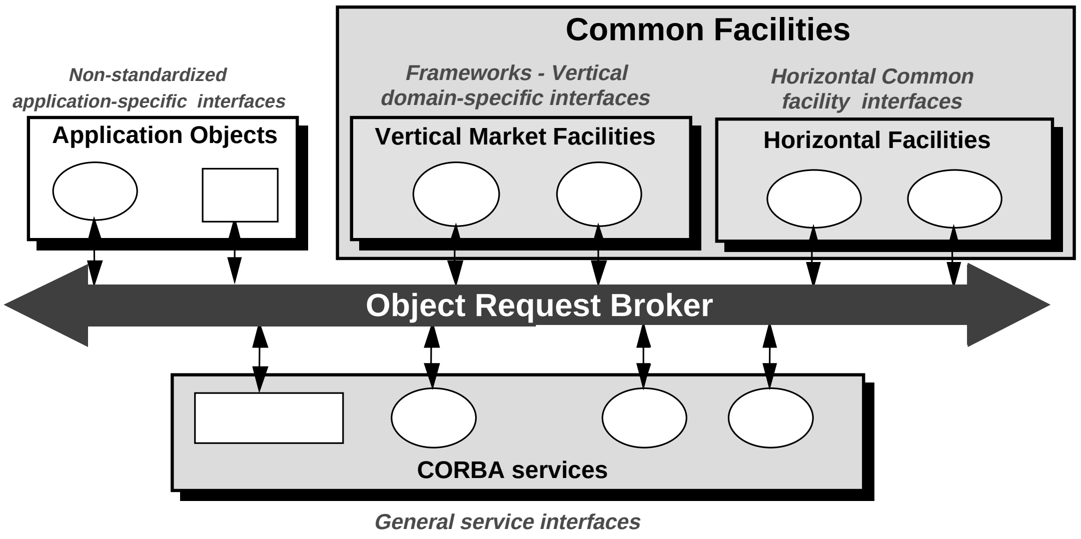

Common Facilities
Non-standardized application-specific interfaces
Frameworks - Vertical domain-specific interfaces
Horizontal Common facility interfaces
Application Objects
Vertical Market Facilities
Horizontal Facilities
Object Request Broker
CORBA services
General service interfaces

Figure 2.7: OMG Object Management Architecture

The OMA includes the ORB (also referred to as CORBA object bus), which supports object location transparency, server object activation and inter-object communication, and four categories of object interfaces (as depicted in Figure 2.7):

- *Object Services* are a collection of domain-independent low-level services that extend the ORB functionality with basic functions that are likely to be used in any program based on distributed objects (such as lifecycle management, naming, persistence, and transactions). These services provide the generic environment where single objects can perform their tasks. Together with the ORB, they are the modular components of what we have previously identified as a middleware configuration;
- *(Horizontal) Common Facilities* are interfaces for horizontal facilities that are oriented to final users and applicable to most application domains (such as user interfaces, information management, system management, and task management). When and where needed, they extend the basic configuration of the ORB and object services, in order to provide more complex middleware configurations with additional properties and APIs for service developers;
- *(Vertical) Domain Interfaces* are APIs tailored to specific domains and areas (such as electronic commerce, telecommunications, and tele-medicine), which may be based on object services and common facilities. According to the previously introduced terminology, they compose the profile layer of middleware facilities;
- *Application Interfaces* are non-standardized application-specific interfaces, which also allow wrapping existing interfaces into the CORBA framework (such as legacy switch control and management interfaces). Since the OMG is interested in the specification of middleware components and not of final applications, these interfaces are not subject to standardization.

3 The SOMA Middleware Configurations

The architecture presented in Chapter 2 has guided the design and the implementation of the SOMA integrated middleware, created for the provision of MA-based services in the Internet global, open and untrusted environment. SOMA is a Java-based MA platform that provides a layered infrastructure of horizontal and vertical facilities, organized into a modular framework of configurations and profiles. These facilities help service developers in the design, implementation and deployment of flexible Internet services with differentiated levels of QoS in scenarios where users, terminals and networked resources can be mobile, as extensively described in Chapter 4 and Chapter 5.

To achieve scalability, SOMA offers a hierarchy of locality abstractions to describe any kind of interconnected system, from simple intranet LANs to the Internet (see Figure 3.1).

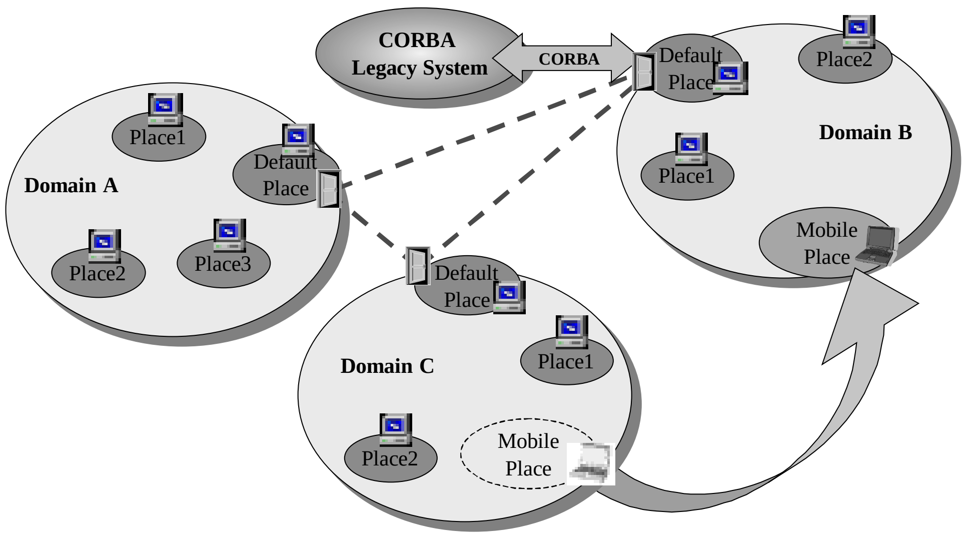


Figure 3.1: SOMA locality abstractions

Any node hosts at least one *place* for agent execution; several places are grouped into *domain* abstractions that correspond to network localities. In each domain, a *default place* is in charge of inter-domain routing functionality and integration with legacy components via CORBA (see Section 3.2.2). The *mobile place* is suitable for supporting mobile devices: it enhances the place locality abstraction with specific functions for automatic reconfiguration when changing domain, and will be presented in more details in Section 3.2.1. Locality abstractions permit also to introduce a scope when considering all other system policies, and help in granting a protected framework for the belonging entities.

SOMA provides weak mobility, i.e., agents migrate together with their code and state, and at the receiving host their execution starts from the method indicated as a parameter of the migration operation. This is achieved without any modification to the standard Java virtual machine, in order to enlarge SOMA usability in the open Internet environment.

The SOMA framework provides a layer of core facilities, i.e., the SOMA ***basic configuration***, which include agent services for migration, identification, communication and persistency. On top of the basic configuration, it is possible to have another set of general-purpose facilities that extend the basic set with advanced mechanisms and services for naming, interoperability and security. These facilities are the components of the SOMA ***complete configuration*** [Bellavista, 00c].

Figure 3.2 gives a general overview of the SOMA architecture, with the different configurations and profiles already implemented. SOMA-based Internet services can be built directly upon the basic/complete configuration, or can exploit all the facilities provided by one or both the SOMA profiles. For instance, a video-on-demand service that distributes multimedia flows to a dynamic set of receivers with heterogeneous characteristics requires middleware facilities for the on-line monitoring and management of the QoS offered and for scaling dynamically the distributed flows depending on current network and systems conditions. Consequently, the service is usually implemented on top of the integrated management profile. When the service is to be provided also to portable mobile devices with strict limitations on computational power, display capacities and network connectivity, it additionally requires middleware facilities for mobile terminal profiling and connectivity maintenance that are provided by the SOMA profile for mobile computing. In this case, as depicted for the Mobile Video-on-demand service in the figure, the service exploits the middleware facilities of both SOMA profiles.

The two configurations available in SOMA are the object of the remainder of this chapter, while Chapter 4 and Chapter 5 will present, respectively, the SOMA profile for integrated management and the SOMA profile for mobile computing, both built on top of the SOMA complete configuration.

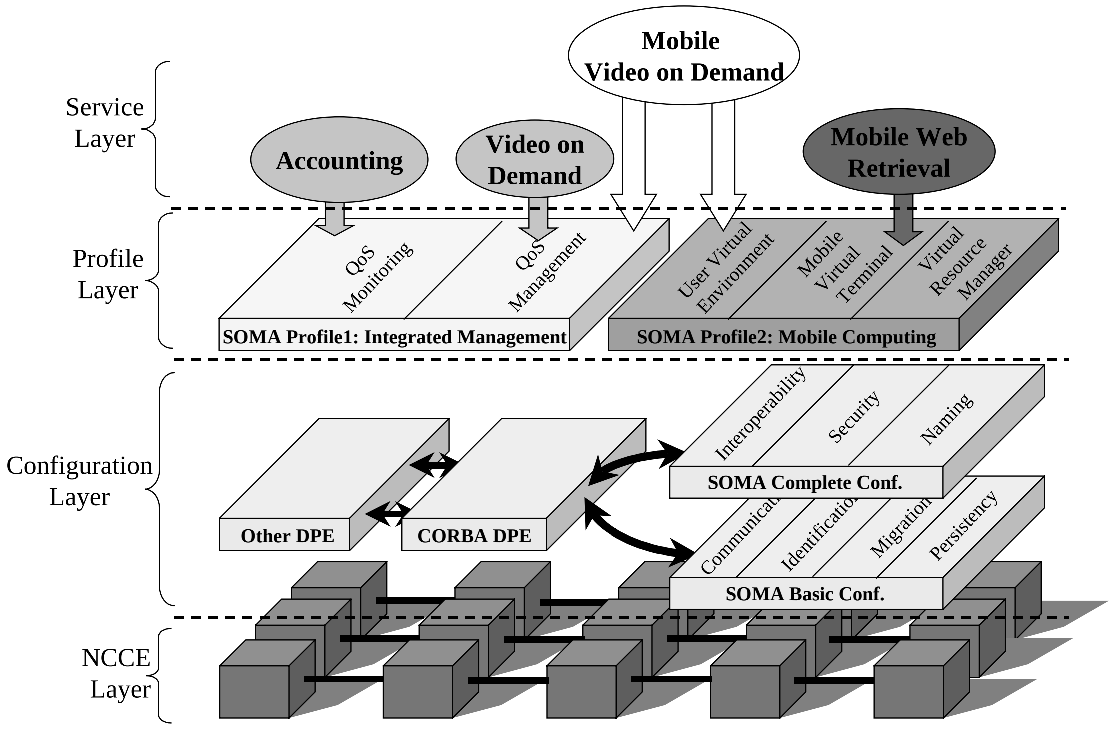


Figure 3.2: The architecture of the SOMA middleware

The two implemented SOMA profiles have not the objective of covering all needs of middleware support for any possible application domain, but simply try to provide a wide set of facilities for the integrated management and the mobile computing areas, which currently rise a great interest in the state-of-the-art research of both academies and industries. Other SOMA profiles are under investigation. For instance, a profile for autonomous information retrieval for virtual museums will include middleware facilities for heterogeneous database connectivity, XML-based interoperable data representations, and complex image-based search patterns. In addition, a SOMA profile for e-commerce will provide an e-marketplace support with facilities for auctions, transactions, and connectivity with enterprise resource planning systems.

3.1 Basic Configuration

The core part of the SOMA project is its architecture, that has been designed along the guidelines of Chapter 2, and offers a distributed infrastructure with a set of facilities for the design and the development of complex network-centric applications. This framework of facilities implement the SOMA Distributed Agent Environment (DAE), which, due to the openness property of the SOMA infrastructure, can be extended at runtime by dynamically adding new services, even built on the already provided functionality.

The SOMA basic configuration groups the basic and primary mechanisms. It includes:

1. ***Agent Migration Facility***. It gives application designers the possibility to simply reallocate network resources and service components at run-time. Entities capable of reallocation are encapsulated by agents that can move in the network by carrying both code and current state with themselves. The migration facility allows developers to choose between SOMA native migration methods and standard transport specifications, such as CORBA Internet Inter-ORB Protocol (IIOP) [OMG, 98] and MASIF [Milojicic, 98], thus permitting a different trade-off between contrasting requirements, such as minimization of transport costs and maximization of interoperability. Interoperable solutions for agent transport are more extensively described in Section 3.2.2.
2. ***Agent Identification Facility***. It permits to assign tags dynamically to any entity in the system. These tags are Globally Unique Identifiers (GUIDs), assigned in a completely decentralized way (see also Section 3.2.1 and [Bellavista, 00b]), and are independent of possible reallocations of entities at runtime. GUIDs are the basis for the realization of the multiple naming systems provided in the naming facility included in the SOMA complete configuration.
3. ***Agent Communication Facility***. It provides mechanisms and tools to simplify coordination and communication between entities. Agents in the same place interact by means of shared objects, such as blackboards and tuple spaces [Cabri, 00]. Any place hosts a Local Resource Manager module that regulates agent access to the node resources. This module controls the authorization of agents

and enforces the place security policy. Whenever one agent needs to share one resource with another agent that resides in a remote place, it is forced to migrate to that place. Outside the scope of the place, agents can perform coordinated tasks by exchanging messages delivered to agents even in case of migration.

4. ***Agent Persistency Facility***. It gives the possibility to suspend temporarily executing agents and to store them on a persistent medium. The facility allows agents not to waste system resources while they are waiting for external events such as the reconnection of one user or terminal to yield back the results of autonomously performed operations (see the SOMA profile for mobile computing in Chapter 5). In addition, it can be also exploited in providing fault-tolerant solutions by organizing and disseminating stored copies of agents [Assis-Silva, 98].

Further details about the facilities provided in the SOMA basic configuration are presented elsewhere [Bellavista, 99a] [Bellavista, 00c]. The short sketch of description reported here simply aims at permitting to understand the more detailed presentation of the advanced facilities of the complete configuration (and, in particular, the naming and interoperability ones), and of the domain-specific profiles, which are the main object of the work done in this doctorate.

3.2 Complete Configuration

The SOMA complete configuration presented in the following can make use, in its implementation, of the lower facilities of the basic configuration. For instance, the naming facility extensively exploits the underlying identification facility. As already said, the complete configuration includes three advanced facilities: the naming facility, the interoperability facility and the security facility, presented in the following.

3.2.1 The Agent Naming Facility

A mobility-enabled naming service, i.e. capable of tracing entities that move in the global Internet environment, requires basic mechanisms to assign GUIDs. MA platforms usually enforce identifier uniqueness by assigning GUIDs to their entities.

The simple solution with only one identification authority is usually not viable because of reliability and overload problems due to centralization. Frequently adopted solutions partition the global environment in non-overlapping regions, and any identification authority is in charge of serving its zone. GUIDs are usually very low-level identifiers; naming services can associate entities with several names that are GUID aliases, suitable for service developers and final users.

Naming services not only translate a high-level name into the corresponding GUID, but also maintain information about current entity location. As a consequence, mobility stresses to the limit the naming service because it should intervene at any migration and at any mobile entity search.

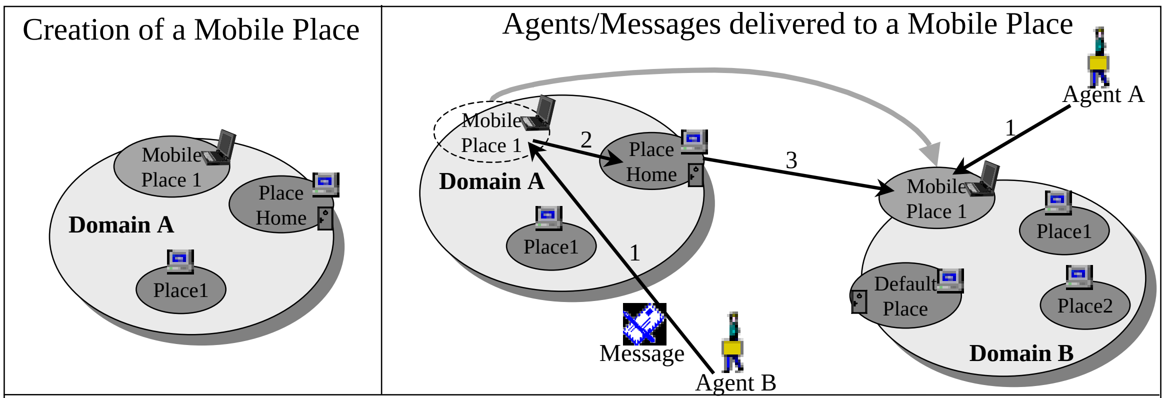
Different naming services can support mobility, with different properties in terms of flexibility, efficiency and scalability, and application developers should choose the most suitable service depending on their specific domain (see also Chapter 5). Mobility-enabled naming facilities can be classified in two main categories, *discovery* and *directory* services. Discovery services usually employ simple protocols to obtain information about entity location (address and simple configuration data) with a minimal knowledge of hosting environments [Perkins,99]. Directory services usually organize names and properties for registered entities in a very flexible way and provide operations to browse all registered information with complex search patterns [Howes, 97].

The discovery service generally provides a solution in a local scope without requiring specific knowledge to its clients: the service is typically requested with a broadcast in the local network. Discovery is usually implemented by a set of independent distributed servers, each one serving the information for its locality. Because of restricted visibility scope and limited query flexibility, many discovery services are lightweight and exhibit excellent performance.

The directory service permits entities to register in flexible hierarchies and can answer advanced queries with pattern matching on complex attributes. The directory service aims to give global visibility to all authorized clients. It is usually implemented in a scalable way by distributed servers that manage (partially) replicated copies of the name space and that coordinate to answer global requests, even by maintaining cached copies of frequently asked information.

Discovery and directory services differ in visibility scope (respectively, local vs. global), flexibility (rigidly predefined and simple structure vs. flexible content and organization), and performance (limited low-level efficient protocols vs. complete high-level searching/registering operations). All these properties can be useful in mobile computing (see the corresponding SOMA profile in Chapter 5), and suggest the provision of a naming service that integrates the different solutions. Directory services should register entities globally available and occasionally mobile, because of the overhead of registration/deregistration and update propagation. Discovery solutions, instead, provide access to entities that move often and are used mainly by components within the same locality. Both naming services require security solutions to permit the access only to authorized users.

SOMA naming is based on care-of mechanisms to locate mobile agents/places, as depicted in Figure 3.3. Any mobile agent to be traced should have its care-of (*agent home*) at the place of its first creation. Similarly, any mobile place has its care-of (*place home*) at the default place of the instantiation domain. The SOMA middleware transparently updates homes of agents at their migration, and homes of places at their connection/disconnection. SOMA mobile agents/places have GUIDs independent of their current position: GUIDs consist of the identifier of the corresponding home associated with a number unique in the home locality. For instance, a mobile place owns a GUID of the form (*DomainID, progNumber*) where *DomainID* is the address of its *place home* [Bellavista, 00c].



Agent A owns the update position of the mobile place and immediately reaches it.

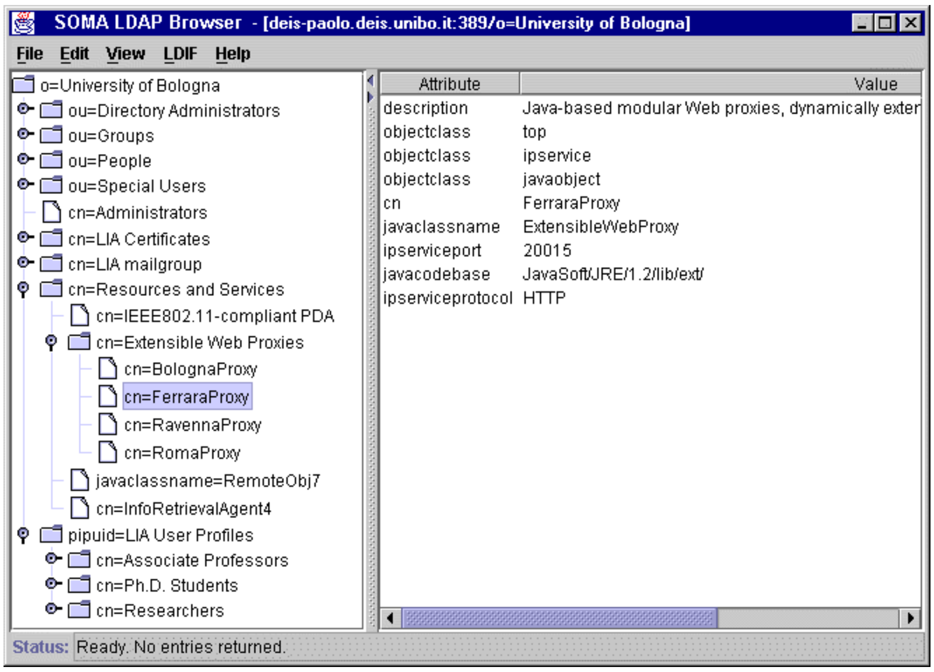
Agent B and **Message** try to reach the mobile place that has already moved; they are tunneled via the *place home*.

Figure 3.3: Agent and message delivery to a mobile place

In addition to basic naming mechanisms, the SOMA middleware provides a naming service that integrates a discovery protocol and a directory service based on the Lightweight Directory Access Protocol (LDAP) [Howes, 97].

The discovery service is the default solution for resource naming within one SOMA domain. We adopt a broadcast protocol to register/deregister resources at the discovery server located at the default place. The choice of the discovery service is suggested by the expected low frequency of resource migration and locality resource access. Up to now, SOMA uses a proprietary discovery protocol, but we are implementing a solution compliant with the Service Location Protocol (SLP) [Perkins, 99].

All entities in need of global visibility register to the LDAP-based SOMA directory service. Any LDAP directory server keeps its entity names and coordinates with other servers to maintain global consistency and to resolve external names. For instance, profiles of users at our department (LIA User Profiles in Figure 3.4) are registered at the directory service to provide mobile users with corresponding preferences from any location. Resources after migration can override the default discovery solution and register to the SOMA directory for wider accessibility.


SOMA LDAP Browser - [deis-paolo.deis.unibo.it:389/o=University of Bologna]
File
Edit
View
LDIF
Help

```
o=University of Bologna
  ou=Directory Administrators
  ou=Groups
  ou=People
  ou=Special Users
  cn=Administrators
  cn=LIA Certificates
  cn=LIA mailgroup
  cn=Resources and Services
    cn=IEEE802.11-compliant PDA
    cn=Extensible Web Proxies
      cn=BolognaProxy
      cn=FerraraProxy
      cn=RavennaProxy
      cn=RomaProxy
    javaclassname=RemoteObj7
    cn=InfoRetrievalAgent4
  pipuid=LIA User Profiles
    cn=Associate Professors
    cn=Ph.D. Students
    cn=Researchers
```

Attribute	Value
description	Java-based modular Web proxies, dynamically exter
objectclass	top
objectclass	ipservice
objectclass	javaobject
cn	FerraraProxy
javaclassname	ExtensibleWebProxy
ipserviceport	20015
javacodebase	JavaSoft/JRE/1.2/lib/ext/
ipserviceprotocol	HTTP

Status: Ready. No entries returned.

Figure 3.4: The LDAP-based SOMA directory service

3.2.2 The Agent Interoperability Facility

The large number of recently implemented MA platforms shows the interest of the distributed systems area in the MA programming paradigm. This variety, however, risks to endanger interoperability and to limit the growth of an MA applications industry. The only way to promote both interoperability and system diversity is to standardize some aspects of the MA technology.

In the area of the traditional C/S approach to OO distributed computing, the Common Object Request Broker Architecture (CORBA) [OMG, 98] is the universally adopted standard for object management, apart from the notable exception of Microsoft which has its own Distributed Component Object Model (DCOM) [DCOM]. CORBA puts together objects that can communicate to each other across boundaries such as network, different operating systems, and different programming languages. CORBA provides network transparency, openness and interoperability. In addition to that core functionality, it specifies an extensive set of object services, common facilities and application interfaces.

The MA model embodies a new programming paradigm, distinguished from the C/S one, and, consequently, different from the CORBA programming model under many points of view (e.g. location awareness vs. network transparency). However, we claim that CORBA can play a fundamental role in achieving interoperability also for the MA technology, working as a standard bridge among proprietary implementations.

The opportunity of an integration of CORBA and MA is also demonstrated by the standardization efforts that have emerged to achieve interoperability between heterogeneous mobile agents. Even if coming from different research communities and different scientific backgrounds, both the Object Management Group Mobile Agent System Interoperability Facility (OMG MASIF) and the Foundation for Intelligent Physical Agents (FIPA) adopt CORBA as the standard bridge to overcome heterogeneity [Milojicic, 98] [FIPA].

The OMG has worked on the specification of MASIF, an agent interoperability standard, built within the CORBA framework, to support agent mobility and management. The goal of MASIF is to achieve interoperability among existing MA platforms from different manufacturers, without forcing any radical modification, but simply by extending implementation with specific “add-on” modules.

MASIF does not suggest standardization of local agent operations such as agent interpretation, serialization, execution and deserialization, because these actions are application specific, and there is no reason to limit MA system implementations. MASIF proposes the standardization for agent and agent system names, for agent system types and for location syntax. It specifies two interfaces: the `MAFAgentSystem` interface provides operations for the management and transfer of agents, whereas the `MAFFinder` interface supports the localization of agents and MA systems in the scope of an administered locality. A `MAFAgentSystem` object should interact internally with MA system-specific services, and provides the associated CORBA interface to external users.

FIPA represents a different approach to interoperability. FIPA specifies the interfaces of the different components for agent interaction with other entities such as humans, other agents, non-agent software and the physical world. Being mainly proposed from the intelligent agent area, FIPA puts the emphasis on the standardization of agent communication, and a dedicated Agent Communication Language (ACL) is proposed for all communication between FIPA-compliant agents.

FIPA defines the concept of an agent platform offering three basic services. These services are namely the Agent Management System (AMS), the Directory Facilitator (DF) and the Agent Communication Channel (ACC). The AMS provides management functions that are similar to the `MAFAgentSystem` ones, except for the notable difference that the FIPA AMS does not address the possibility of migrating agents between heterogeneous MA platforms. FIPA agents may offer their services to other agents and make their services searchable in yellow pages by the DF. Registration on a DF is discretionary while registering on the AMS is mandatory on any agent platform. Finally, the ACC enables communication between agents on the same platform and between possibly heterogeneous platforms, by offering a message forwarding service. Reachability between platforms is obtained by making the forward service available over the CORBA ORB whose integration is considered mandatory for any FIPA-compliant MA platform. Agent messages are transferred on top of the CORBA IIOP.

While the AMS and DF services provide functionality similar to the MASIF `MAFAgentSystem` and `MAFFinder`, a specific characteristic of the FIPA standardization proposal is the agent communication via the definition of an interoperable ACL. In addition, FIPA agents acquire a predictable behavior being described by common semantics defined in the interpretation of a common language. This is achieved by the concept of communication acts [FIPA].

The SOMA interoperability facility takes into account both MASIF and FIPA specifications, as described in the following. Before giving details about the implementation of interoperability in SOMA, let us note that CORBA compliance imposes some additional costs, especially in terms of run-time performance, but it is worth the trouble because it ensures openness and stability to applications, saving the investments in MA-based programming. Our scenario puts together two models: we use proprietary and efficient solutions for internal operations among SOMA entities, but provide standard CORBA interfaces, both for exploiting the available CORBA services and for making SOMA itself a CORBA application object.

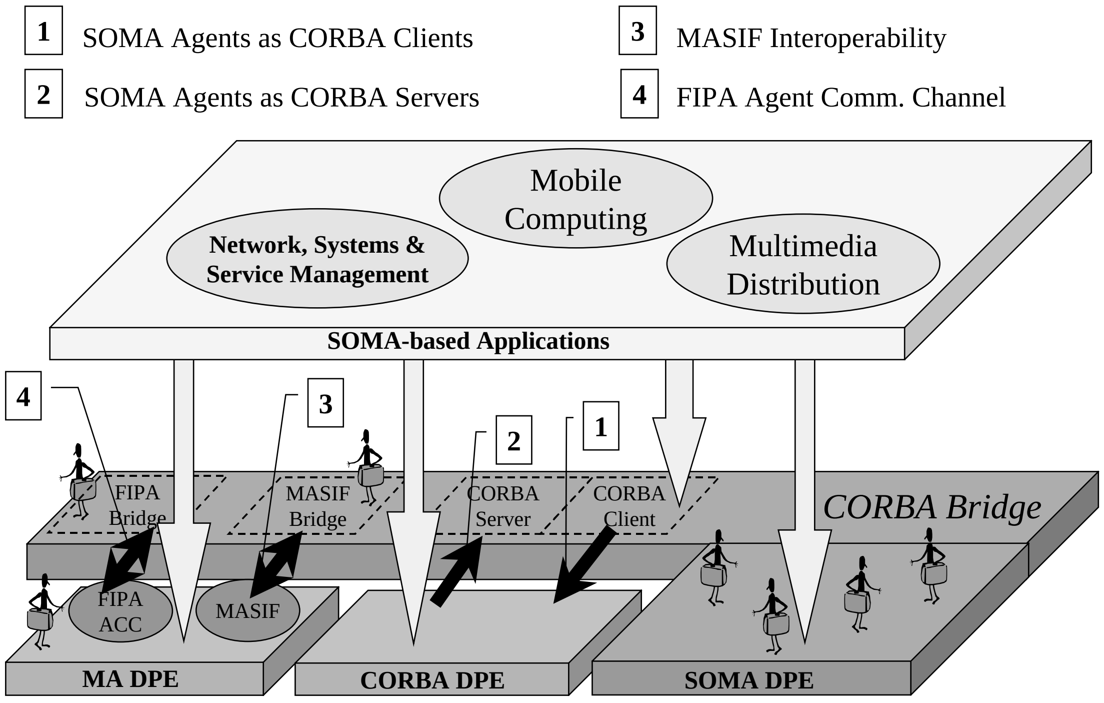


Figure 3.5: The four modules of the SOMA interoperability facility

SOMA faces four different challenges to provide interoperability (see Figure 3.5):

1. an agent may call external CORBA objects (SOMA agents as CORBA clients);
2. an agent may publish its interface to one ORB (SOMA applications as CORBA servers);

3. any external entity may access SOMA through the standard MASIF interface (interoperability between SOMA and other CORBA components);
4. an agent may send/receive messages to/from any agent platform via the FIPA ACC forward service (communication interoperability between SOMA agents and FIPA-compliant agents).

The first two features are provided by the *CORBA C/S* extension of SOMA: agents can play the role of CORBA clients and can also register themselves as CORBA servers to offer access points to an application outside the SOMA system. Even if there is no conceptual problem in a mobile agent registering as a CORBA server, we currently grant this possibility only to SOMA agents that do not migrate during their lifetime (stationary agents) to avoid the overhead of registering/unregistering with the CORBA Naming Service at every migration.

The third feature is a more complex issue and SOMA addresses it via MASIF compliance. Any external system can control remote agents of a MASIF-compliant MA system via the `MAFAgentSystem` interface: MASIF defines methods for suspending/resuming/terminating agents and for moving agents from one MA platform to another one. The interoperation is significant only when the two interworking systems present a compatibility base, that is the same implementation language, or compatible externalization mechanisms. Agent tracking functions permit the tracing of agents registered with `MAFFinder`, introduced to provide an MA name service, because the CORBA Naming Service is not suitable for entities that are intrinsically and frequently mobile.

About the fourth interoperability point, at the moment SOMA agents can communicate via proprietary mechanisms and protocols, but can also decide to exploit the CORBA middleware to coordinate via shared CORBA objects. Agent communication is outside the scope of MASIF: for this reason, we have decided and are now completing the integration of an additional module to provide full compliance with FIPA. SOMA mainly focuses on the implementation of the ACC because it provides interoperability for communication between heterogeneous agents that is not covered by the MASIF compliance. The SOMA ACC is available as a place facility that agents exploit to convert messages into the corresponding ACL format and vice versa, with an approach similar to the one of Jade [Jade]. The implementation of the AMS and DF facilities are mapped into the analogous functionality for agent management and registration already available in the SOMA `MAFAgentSystem` and `MAFFinder` modules.

The SOMA programming framework achieves interoperability by extending its basic functions in a modular way, as depicted in Figure 3.6. In particular, places in charge of interoperating are extended with the *CORBABridge* add-on that is composed by three modules: the first one (*CORBA C/S*) simplifies the design of SOMA entities as CORBA clients/servers; the second one (*MASIFBridge*) implements the MASIF functionality; the third one (*FIPABridge*) implements the FIPA ACC.

Since MASIF and FIPA compliance increases the code dimension of SOMA places, our default configuration does not extend all places with the *MASIFBridge/FIPABridge* modules, but only the default place of each domain. The *CORBA C/S* module instead is lightweight, and many places in the same domain may use it to access the CORBA bus, either for calling external services or for registering as servers.

Any SOMA agent, resident at a CORBA C/S extended place, is able to act as a CORBA client/server through static (IDL stub/skeleton) and dynamic (Dynamic Invocation Interface/Dynamic Skeleton Interface) invocations/registrations. Our implementation is based on the VisiBroker 4 ORB [Visibroker]. However, it is portable, with no modification at all, on any other ORB implementation compliant to the CORBA 2.2 specification. In fact, we have only used the portable functions provided by the Internet Inter-ORB Protocol and the Portable Object Adapter [OMG, 99a], introduced to overcome possible incompatibility between different ORB products.

The implementation work to achieve SOMA interoperability via full compliance with CORBA has been simplified by the fact that SOMA is completely written in Java. In our experience, CORBA and Java have demonstrated to integrate with one another in a synergic way. The former provides network transparency, while the latter achieves implementation transparency via the Java Virtual Machine common software layer. In addition, Java is deeply integrated with the Web, thus offering universal accessibility and a potentially wide user base to CORBA. For instance, a dynamic download of one applet with a CORBA client can produce the invocation of a CORBA server object from a CORBA-enhanced Web browser, such as Netscape Communicator 4.x.

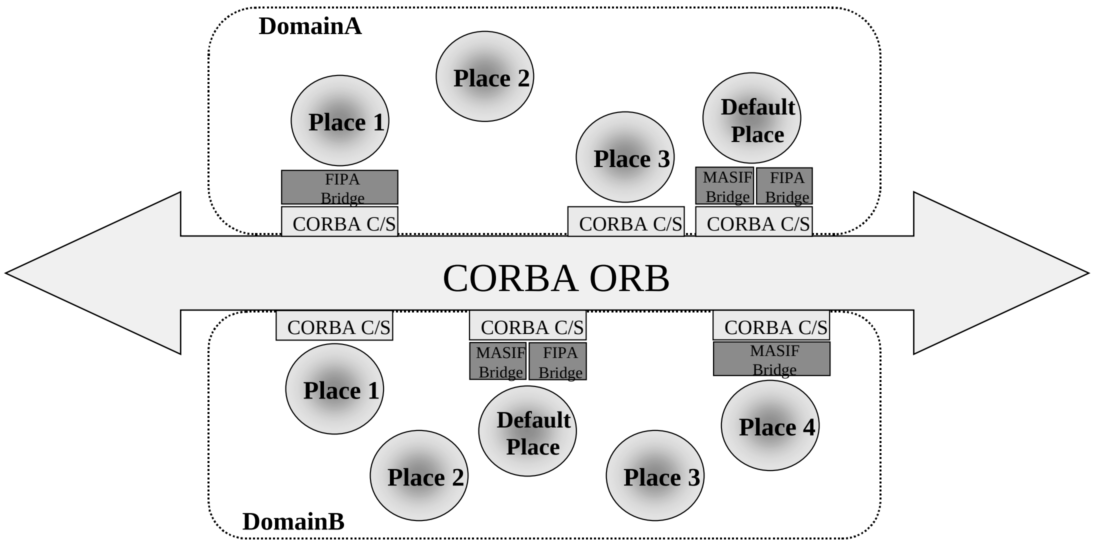


Figure 3.6: The modular implementation of the SOMA interoperability facility

To give an idea of the interoperability costs in SOMA, Figure 3.7 compares the average cost of the native migration mechanism with the one imposed by the MASIF interface. A more exhaustive discussion of the different SOMA interoperability costs (e.g., including interoperable registration and communication) can be found in [Bellavista, 99].

Here, we have carried out the experimental results in a SOMA system composed by several domains, and we have measured the cost for migration between default places of different domains. The measurements have been taken on an Ethernet LAN of 300-MHz PentiumII PCs with Windows NT. As we expected, MASIF agent migration is more expensive than SOMA proprietary one, as reported in the Total columns. The Total columns include the initial setup overhead to establish a connection between previously unconnected places. For this reason, any successive migration between the same places achieves better performance, as indicated in the table. In addition, in the case of MASIF migration, the Total column includes also the cost to obtain needed references to the `MAFAgentSystem` and `MAFFinder` from CORBA and MASIF naming services (as reported in the phase 1 column).

The results also show that MASIF performance is only about 10% worse than the SOMA one in the case of successive migrations for 50kB-sized agents. This is mainly due to the fact that both the VisiBroker ORB and the SOMA framework use the same Java de/serialization mechanisms, and the de/serialization overhead represents the most relevant factor with the increasing of agent size. The performance obtained for MASIF migration, however, is extremely acceptable and demonstrates the viability of the MASIF approach to interoperability.

Agent size (kB)	MASIF migration (ms)			SOMA proprietary migration (ms)	
	Phase 1	Total	Succ. migrations	Total	Succ. migrations
5	660	**2320**	1714	**1648**	1393
50	645	**4854**	4261	**4204**	3919

Figure 3.7: Costs of SOMA interoperable/native agent migration

3.2.3 The Agent Security Facility

The service provision in untrusted Internet environments imposes a thorough security framework, which should be also flexible enough to accommodate the range of service operators with different levels of authorized operations, from service providers to network administrators and to simple users. This motivates the SOMA model of trust, that defines who or what in the system is trusted, in what way, and to what extent [Oppliger, 98].

While the naming and interoperability facilities have been designed and implemented within my doctorate work, the security facility has been the focus of the activity of another doctorate candidate in my research group. For this reason and to permit the understanding of some parts in the following, I report here only a brief description of the security facility. A more extensive discussion about SOMA security solutions can be found in [Corradi, 99] [Montanari, 01].

SOMA has been developed for an untrusted Internet environment, where the communication network is considered insecure and any node may host the execution of possibly malicious entities. In addition, a SOMA agent is an active entity that acts on behalf of a principal, i.e. the person/organization that has launched the agent execution and that is responsible for its operations. SOMA agents are authenticated by means of standard certificates, provided and administered through the integration with a public key infrastructure [Entrust]; this integration permits agent authentication not only in the case of single-hop migration, but also when considering multiple-hop mobility. The actions that agents are authorized to perform depend on roles associated to agent principals. SOMA permits the dynamic definition and control of a range of roles, from administrators to users [Lupu, 97].

The SOMA security mechanisms support the model of trust and enforce security policies: authentication permits to identify the role associated with SOMA agents; authorization recognizes whether an operation is permitted on a resource; integrity guarantees that agents and data have not been maliciously modified during reallocation; secrecy permits to protect entities from any exposure to malicious intrusions.

In SOMA, security is provided with application-level tools, taking advantage of available standard solutions and products (e.g., the IAIK cryptographic functionality and the Entrust Public Key Infrastructure [IAIK] [Entrust] [Gong, 97]). If the debate about at which level a system has to offer security is still open, the discussion concentrates on the issues of transparency, flexibility and performance [Chen, 98] [Oppliger, 98]. Independently of the abstraction level adopted, it is important to consider security as a property to be integrated at any system layer. Only this pervasive approach followed by SOMA design can achieve the full level of security, higher than the minimal one obtained by systems that add a posteriori security strategies.

The security infrastructure for mobile agents extends the traditional sandbox solution used to protect network nodes from the execution of untrusted code, because the sandbox approach limits too much the MA expressive power [Gong, 97]. With regard to implementation, SOMA agents use X.509 certificates for authentication, which ascertain the role of the agent principal before authorizing any interaction with resources. In addition, we have integrated SOMA with a commercial Public Key Infrastructure (PKI), provided by Entrust [Entrust], to automatically distribute keys, to manage certificates and to perform all related administrative tasks. The integrity check can employ either MD5 or SHA1. Secrecy is granted when needed by encrypting/decrypting communications with DES and SSL [IAIK].

CORBA and MASIF standards recognize the security requirement by imposing tools and mechanisms to enforce security when interacting with external components. In accord with this guideline, SOMA addresses the security threats introduced by interoperating with CORBA. On the one hand, sending/receiving

CORBA requests/replies requires channel encryption to ensure privacy on exchanged messages. On the other hand, the possibility for SOMA agents to act as CORBA servers and for SOMA localities to host agents from other MA platforms calls for mechanisms for client/agent authentication, auditing and access controlling. SOMA provides security solutions compliant both with CORBA Security Services and with MASIF security specifications [OMG, 98] [Milojicic, 98]. We have worked also on providing SOMA compliance with the Secure Inter-ORB Protocol [OMG, 98] to enable secure interactions between entities resident on different ORBs, provided that they adopt the same security technology.

Finally, SOMA gives users the possibility to choose the best trade-off between security needs and required performance, according to the intended usage, as depicted in Figure 3.8. Agents in trusted environments (e.g., a private Intranet of a department) could directly access resources after the authorization check, while agents moving in untrusted environments (e.g., the Internet) generally have to pass all security steps for secrecy, integrity, authentication and authorization.

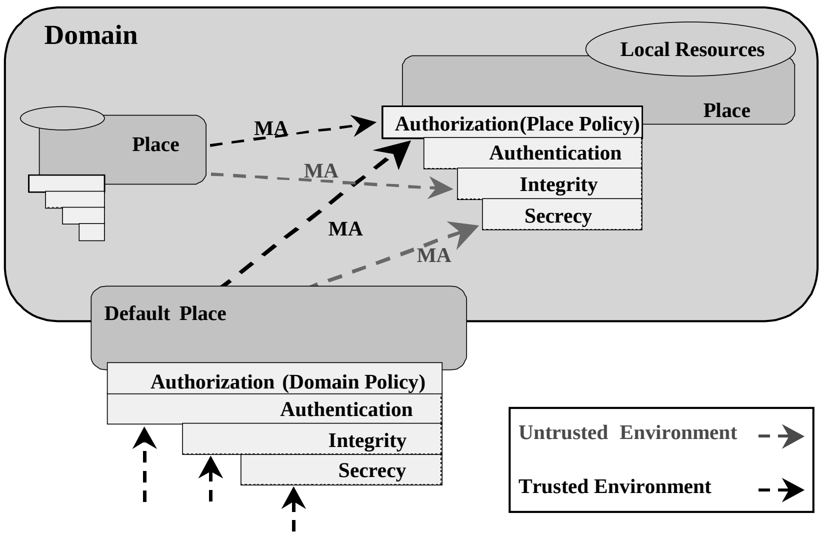


Figure 3.8: The SOMA security facility in trusted/untrusted environments

To conclude, the SOMA middleware is available in two general-purpose configurations: the basic lightweight configuration and the complete configuration, which extends the basic one with facilities for naming, interoperability and security. In addition, to support SOMA developers in the design, implementation and deployment of Internet services in application domains with specific requirements, we have already developed two different SOMA profiles that run on top of the complete configuration. The SOMA profile of middleware facilities for integrated management and the one for mobile computing are extensively presented, respectively, in the following Chapter 4 and Chapter 5.

4 The SOMA Profile for Integrated Management

Several research efforts have investigated solutions for the increasing requirements of QoS differentiation and guarantees in Web-based services, by considering different levels of abstraction [Chalmers, 99] [Hutchison, 94].

On the one hand, the definition and standardization of new protocols has been investigated to ensure the reservation of the needed amount of network resources [Zhang, 93] [Busse, 96]. However, the process of acceptance and deployment of new standards for network-layer protocols is long and difficult, mainly due to the large base of non-programmable and already installed network equipment. In this field, mobile agents have shown their suitability for implementing tunneling techniques to integrate network resources that are not compliant with the reservation standards [de Meer, 98].

On the other hand, some work has shown the opportunity of an application-layer approach to QoS, especially in the areas of mobile communications and multimedia distribution [Chalmers, 99] [Kone, 98]. Application-layer solutions propose service infrastructures that tries to respect the specified QoS requirements without any guarantee of satisfaction, but with no need to modify the underlying best-effort network layer. The idea is to monitor the available QoS and to notify service components of quality modifications in order to adapt to the network traffic.

The requirements for on-line monitoring and management of QoS are the most evident example of how the provision of services in the global Internet environment is significantly increasing the complexity of the management issue. Management not only involves the configuration and control of geographically distributed network elements, but also tends to include the administration of heterogeneous general-purpose systems (e.g., configuration, load-balancing and performance management) and the dynamic control, adaptation and tailoring of the offered services with specified properties (e.g., scalability and fault-tolerance). Other examples of emerging service requirements are the registering, billing and accounting of service users for their effective resource consumption, and the protection from the different possible forms of denial-of-service attacks, whether malicious or not.

The evolution of management requirements has suggested considering new management models to overcome the limits of traditional centralized client/server approaches. There is a growing interest in taking into account Web-based management systems [Thompson, 98] [Anerousis, 99] and in adopting integration standards such as CORBA that also permits to deal with legacy components [OMG, 98] [Haggerty, 98] [Mazumdar, 96] [CORBA/CMIP]. There is strong emphasis in the use of mobile entities to provide flexible, scalable, and effective management solutions by programming network resources dynamically [Goldszmidt, 95] [Fuggetta, 98] [Karmouch, 98] [Breugst, 98] [Bieszczad, 98] [Bellavista, 99b] [Bellavista, 00a] [Chen, 98] [Magedanz, 96]. There are also encompassing efforts in defining open architectures to integrate the management of traditional telecommunications with new distributed services [Inoue, 98] [Glitho, 95]. Recent research approaches recognize the following important issues in resource and service management for open, global, and untrusted systems:

- to facilitate delegation and automation of control actions, thus reducing network load, relieving the central manager duties, and improving scalability;
- to address the management of heterogeneous network elements by focusing on interoperability and by promoting acceptance of new standards;
- to help in the design and fast deployment of new services, improving user customization and avoiding time-consuming redesign;
- to provide secure environments on top of intrinsically untrusted networks.

New management approaches propose solutions to the above issues with different peculiarities and at different levels of abstraction. For instance, consider the case of resource allocation that can be either visible or transparent to managers. While allocation visibility permits to obtain efficient solutions, allocation transparency helps to cope with the complexity of internetworked systems. Active Networks (ANs) exemplify how allocation visibility can be used for management purposes and also for introducing new protocols without discontinuing system operations. On the contrary, CORBA-based solutions propose a higher-level approach that hides allocation to applications, simplifying the development of distributed services. We believe that a management environment should offer both allocation visibility and allocation transparency. The former is compulsory to express management policies and to obtain efficient solutions while final users prefer the latter when designing complex distributed services.

We feel that the main issue still to be faced is the definition of a comprehensive solution for the integrated management of both network resources and services, able to provide all the features required by different levels of usage and with different levels of abstraction [Glitho, 98].

To provide a framework capable of answering all the issues sketched above, we have designed and implemented a specific SOMA profile, called *Integrated Management Profile*. The profile aims at providing all needed mechanisms, tools and policies to support the management of network elements, systems and services in the global Internet environment. It runs on top of the SOMA complete configuration, and can be installed, even at runtime, on the nodes that require the integrated management functionality. The profile is composed by a module called MAPI (Monitoring Application Programming Interface), which extends the visibility of the state of heterogeneous networked resources achievable from within the JVM, and by a set of management agents that migrate to their target resources, exploit MAPI for their monitoring, and autonomously perform management operations on local resources.

This chapter gives first an overview of the recent related work about integrated management. Then, it presents the MAPI component for local on-line monitoring of heterogeneous resources, and the MAPI exploitation in MA-based tools for distributed management. The experience of applying the SOMA profile for integrated management to the domain of multimedia flow distribution with differentiated QoS ends the chapter.

4.1 Related Work

Recent researches have proposed several different approaches to overcome the limits of traditional management systems. We do not want to give a general overview of these approaches, but only try to sketch their peculiarities and to identify their main differences. In particular, we stress that proposed solutions are at different levels of abstraction and suit different specific issues in the management domain.

The main idea in ANs is to program network components, so that users can directly modify the behavior of the network itself while it continues to operate. ANs push programmability down to the network layer of the OSI protocol stack, and have already shown their capacity of achieving significant results in terms of flexibility, performance, scalability and QoS provision [Chen, 98] [Tennenhouse, 97] [Psounis, 99]. However, there are several typical management issues difficult to be solved at the network layer. For instance, there is no general agreement on the level at which security should be faced, and people discuss whether security should be considered at either the network layer or the application one [Oppliger, 98]. We believe that many security-related tasks are more easily addressed at a higher level. For instance, user authentication requires public key infrastructures usually available as application level tools, and also the association of authenticated users with recognized roles needs application level facilities for defining and managing the proper trust model [Lupu, 97].

Other solutions that make use of code mobility for network management come from the MA research activity [Karmouch, 98] [Breugst, 98] [Bieszczad, 98] [Bellavista, 99b] [Rothermel, 98] [Gavalas, 99]. Probably, the most limiting feature of MA-based management approaches seems the fact that only a few MA platforms address interoperability with existing management components, whether MA-based or traditional ones [Grasshopper] [Voyager] [SOMA]. In addition, they do not generally provide a layered architecture of common services, making difficult the development of complex management applications.

CORBA is the most widely diffused architecture to deal with distributed heterogeneous programming. However, even if CORBA has raised great interest in the management area, it currently seems more to play the role of integration technology among existing solutions (CORBA gateways toward SNMP/CMIP components [Mazumdar, 96] [CORBA/CMIP]) than to propose a framework to build new CORBA-based management applications. Some peculiarities of CORBA partially limits its use in the management area: CORBA-based applications are typically location unaware, while managing distributed resources and services usually requests visibility of topology and locality information. In addition, CORBA implementations lack abstractions for managing object groups, even if the collection abstraction is clearly necessary for the management of replicated services [Felber, 98] [OMG, 98]. Finally, the interaction of objects using diverse security technologies is complex because CORBA does not standardize the possibility to negotiate security technology [Staamann, 98].

Other proposals abstract from implementation technologies and describe solution frameworks at the architecture level. The Telecommunications Management Network (TMN) framework [Glitho, 95] goes beyond the manager/agent model of OSI systems management [ISO, 92] by introducing a distributed set of cooperating systems for monitoring and control, conceptually separated from the telecommunications network being managed [Glitho, 98] [Psounis, 99]. TMN main limitation for highly dynamic and open systems seems to be its client/server management model.

The Telecommunication Information Networking Architecture (TINA) [Inoue, 98] proposes a solution at a higher level of abstraction. TINA architecture is directed to design any kind of service, running on a global scale and on different network technologies. TINA suggests a uniform support for management where the management itself is considered a service. TINA applications, service components and network resources reside on top of a DPE that can hide the complexity of distribution and heterogeneity from service developers. Unfortunately, to present a global solution, TINA seems to push towards very complex implementation, so that some research works have addressed the issue of implementing simplified architectures of TINA to offer an earlier opportunity for cost-effective evolution of current networks [Redlich, 98].

In addition to the research activities on management architectures and organizational frameworks presented above, several researchers have recently worked on the implementation of mechanisms and tools to support both local and distributed monitoring with very different approaches [Buyya, 00] [Russ, 99] [Al-Shaer, 99] [Miller, 95] [Lange, 92] [Bakic, 00] [Schroeder, 95] [Weiming, 98] [Corradi, 97] [Liang, 99] [Stallings, 98].

Several distributed instrumentation systems have achieved interesting results, especially in terms of minimization of monitoring intrusion [Miller, 95] [Lange, 92] [Bakic, 00]. However, they require to instrument statically/dynamically monitored applications and tend to be either language- or platform-specific. This is not suitable for the on-line monitoring of Web-based services that consist of distributed components not to be suspended during execution.

Some efforts have specifically addressed on-line monitoring. They have concentrated on producing effective tools by generally exploiting ad-hoc mechanisms available only for specific operating systems [Schroeder, 95] [Weiming, 98]. These solutions are too platform-dependent to be suitable for open and intrinsically heterogeneous distributed environments such as the Internet.

In the area of network monitoring and management, many researchers have used standard protocols to interrogate the state of network equipment. The most diffused protocol is still SNMP, briefly described in Section 4.2, mainly because of its simplicity [Jiao, 00]. Other approaches start to be common: some of them provide network traffic monitoring with the granularity of a whole network segment (Remote MONitoring - RMON [Stallings, 98]); others exploit platform-dependent libraries and commands (such as the UNIX `libpcap` library) to enable network packet capture, filtering and analysis at general-purpose hosts [Kumar, 00] [Deri, 00]. The goal of these tools, however, is mainly the dynamic observation of network traffic, and not the provision of on-line monitoring of service components at the application level.

The diffusion of Java for the implementation of Web services has changed the perspective also in the monitoring area. First activities have simply addressed the enhancement of standard SNMP solutions with Web accessibility [Lee, 00]. Then, some proposals have started to exploit Java networking facilities and code mobility to provide an integrated middleware for distributed monitoring. Probably due to the novelty of the technology, there are few examples of Java monitoring tools based on the JVM Profiler Interface (JVMPI), presented in the following section. Perfanal [Meyers, 00] exploits the SUN HPROF profiler agent to perform an off-line analysis of collected monitoring data and to obtain a user-level concise view for debugging Java applications. JProf [Pennington, 00] implements its own profiler agent and profiler process, and provides a large set of functions to present the results of an off-line data analysis in user-level interoperable formats, such as tables and diagrams organized by using XML. To the best of our knowledge, there are no solutions that currently exploit Java Native Integration (JNI), briefly sketched in the following, to integrate native monitoring mechanisms in a Java-based monitoring tool.

4.2 On-line Monitoring

To support integrated management, the service infrastructure should be informed of the current usage of all heterogeneous resources at runtime. In other words, the service infrastructure should include an on-line monitoring tool able to detect the current condition of network, system and application components during service execution. This is necessary to enable dynamic service management via runtime corrective operations.

Monitoring information covers different abstraction levels, from system conditions at each node (the usage of CPU, memory, bandwidth, etc.), called kernel state in the following, to the state of application-level service components (the state of a service-specific daemon process, etc.), sometimes referred as application state [Buyya, 00]. In addition, the on-line requirement makes critical to have short response time and to reduce the overhead in the observed target, thus forcing to collect only a restricted set of kernel and application state indicators.

We have designed and implemented a Java-based Monitoring Application Programming Interface (MAPI) for the on-line monitoring of Web services. MAPI overcomes Internet platform heterogeneity and permits to observe the state of systems/applications during execution. MAPI collects monitoring data at different levels of abstraction. At the application level, it permits the dynamic interaction with the JVM to gather detailed information about the execution of Java-based services. At the kernel level, it enables the access to the needed system indicators of the monitored target (either Java-based or external to the JVM), such as CPU and memory usage of all active processes.

The extensive utilization of the Java technology as the middleware to develop Internet applications and services has motivated the choice of Java for the implementation of the MAPI interface. To overcome the transparency imposed by the JVM, MAPI exploits some recent extensions of the Java technology: the JVM Profiler Interface (JVMPI) [JVMPI] and the Java Native Interface (JNI) [Gordon, 98]. In addition, MAPI integrates with external standard monitoring entities particularly diffused in the network management domain, i.e., Simple Network Management Protocol (SNMP) agents [Stallings, 98]. JVMPI makes possible to instrument dynamically the JVM for debugging and monitoring purposes, and MAPI exploits it to collect, filter and analyze application-level events produced by Java applications, e.g., object allocation and method invocation. At the kernel level, MAPI collects system-dependent monitoring data, e.g., CPU usage and incoming network packets, by interrogating SNMP agents that export local monitoring data via their standard Management Information Base (MIB). To monitor the kernel state of hosts without any SNMP agent in execution, MAPI uses JNI to integrate with platform-dependent monitoring mechanisms, which we have currently implemented for the Windows NT, Solaris, and Linux platforms.

4.2.1 Java Technologies for Monitoring

The Java technology plays a fundamental role in the design, implementation, and deployment of Web services over the Internet infrastructure. Apart from Java portability, dynamic class loading, and easy integration with the Web, the main motivation of Java diffusion is its virtual machine that hides the local operating system and presents a uniform vision of all available computing resources and middleware facilities.

However, the monitoring perspective requires a complete and low level visibility of both JVM internals and underlying platforms. At the application level, the MAPI monitoring component exploits JVMPI to acquire visibility of the JVM internal events. At the kernel level, MAPI employs modules external to the JVM, to gather information about platform-dependent resources and non-Java application components. In MAPI, these external modules include both native monitoring mechanisms integrated via the JNI technology, and standard monitoring components, i.e., SNMP agents.

The Java Virtual Machine Profiler Interface

JVMPI is an experimental interface featured by the Java 2 platform and mainly designed to help developers in monitoring Java-based applications during debugging. JVMPI is an API available to the JVM and to a dedicated profiler agent, often implemented as a platform-dependent native library for sake of performance (see Figure 4.1). In one direction, the JVM notifies several internal events to the profiler agent; in the other direction, the profiler agent can enable/disable the notification of specific types of events and can perform some limited management actions on the JVM.

With a finer detail degree, several conditions trigger JVMPI events: Java thread state change (start, end, when blocking on a locked monitor); beginning/ending of invoked methods; class loading operations; object allocation/deallocation; beginning/ending of the JVM garbage collection. Any event notification carries full information about the entities that have generated that event. For instance, the allocation of a new object triggers the `JVMPI_EVENT_OBJECT_ALLOC` event, and the profiler agent receives the identifiers of the new object and of its class, together with the size of the allocated heap memory.

The profiler agent can also use JVMPI in the opposite direction, to modify dynamically the behavior of monitored applications. Apart from notification enabling/disabling, the agent can intervene on the JVM by invoking a very small set of JVMPI methods: management actions are limited to suspend/resume Java threads and to enable/disable/force the immediate execution of the JVM garbage collector.

Figure 4.1 shows the SUN-proposed profiling architecture. The profiler process acts as a front-end to provide application developers with a readable presentation of the monitoring data. The profiler agent collects all notifications of enabled JVM events. Both process and agent can also be external to the JVM.

The SUN distribution provides a simple implementation of the profiler agent for both Solaris and Windows NT operating systems. This agent, called HPROF [JVMPI], collects general-purpose events and allows simple static configurations. It is not designed for on-line monitoring, but works mainly as an off-line post-mortem tool for debugging and performance analysis. In fact, it tends to collect a large volume of monitoring data that requires heavy filtering and processing to obtain significant and concise service indicators. For this reason, some researchers have implemented their ad hoc profiler processes to organize HPROF data in immediately readable graphic interfaces [Pennington, 00] [Meyers, 00].

JVMPI constrains the provision of monitoring information. Developers can only specify whether the JVMPI supported events should be notified to the profiler agent, and the specification is coarse-grained, with no possibility of fine selection and dynamic refinement. For instance, a profiler agent can only choose to enable/disable all events related to all Java classes (or objects/methods/monitors), but it can neither focus on the events generated by a specific class nor define user-/application-specific events. The only way to obtain more fine-grained indicators is to implement ad hoc profiler agents, as our MAPI profiler agent presented in Section 4.2.3, capable of filtering the events of interest and suitable to compose them in higher level indicators. Moreover, JVMPI cannot give any direct information about system resources and application components outside the JVM, e.g., the number of non-Java processes and the set of files opened by a non-Java process.

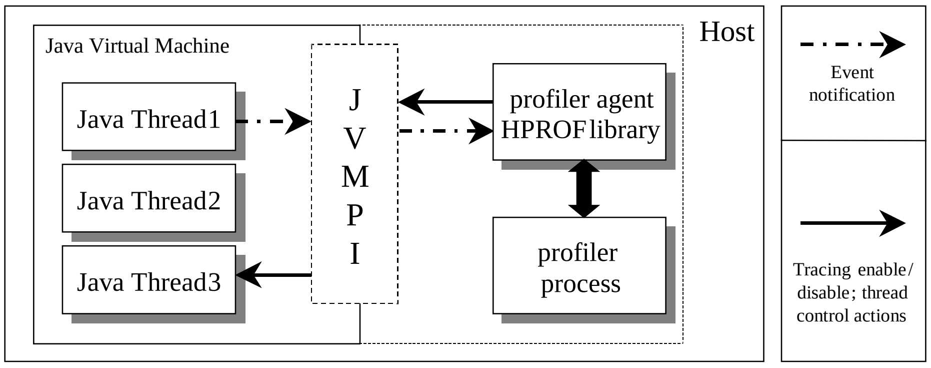


Figure 4.1: JVMPI-based architecture for JVM monitoring

<u>The Java Native Interface</u>

JNI permits Java threads to invoke *native methods*, i.e., platform-specific functions typically written in C/C++, usually available as Dynamic Link Libraries (DLL) in the Windows platform and Shared Object (SO) libraries under Solaris and Linux (see Figure 4.2). These native libraries contain platform-dependent executable code and cannot be directly ported to heterogeneous targets. SUN has introduced JNI to ensure compatibility of native code invocations in all JVM implementations.

JNI is a two-way interface. In one direction, a Java program invokes a native method, by declaring the method with the keyword `native` and with no body. After the binding obtained by calling the `System.loadLibrary()` method, the JVM uses JNI to call the requested function in the native library during execution. JNI specifies the details of method invocation: for instance, it rules the parameter marshalling/unmarshalling between the Java invoking thread and the invoked native method.

In the other direction, from the native library towards the JVM, JNI allows native methods to interact with their invoking Java framework. JNI permits native code to callback the Java environment and the invoking Java object, to access and modify object values, to call Java methods, and to raise Java exceptions.

With regards to monitoring, JNI permits to integrate the JVM with native monitoring libraries, to obtain the visibility of kernel and application indicators not accessible via JVMPI. For instance, our MAPI component can collect information about all active processes (e.g., to record process CPU usage in a specified time interval), by invoking the execution of C-based native libraries that extract that information, depending on the monitored target, from either the Windows NT registry keys or the Solaris `/proc` directory.

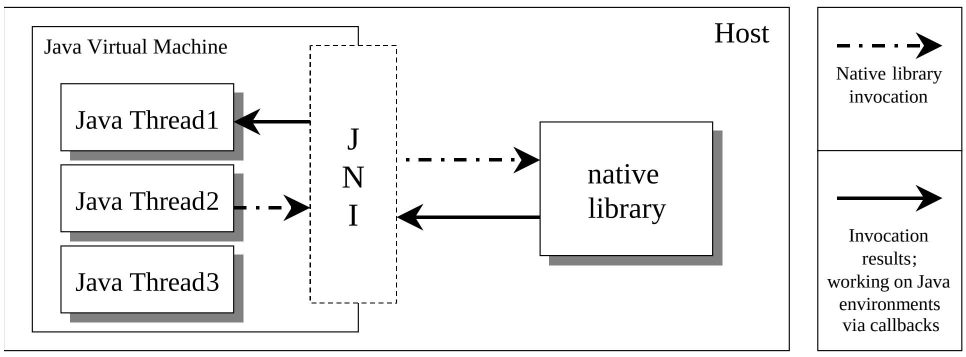

Host
Java Virtual Machine
Java Thread1
Java Thread2
Java Thread3
J N I
native library
Native library invocation
Invocation results; working on Java environments via callbacks

Figure 4.2: Two-way JNI for native library invocation

The Java Integration with SNMP Agents

Recent research work has recognized the relevance of Java as the technology to simplify the integration of new and legacy monitoring components in a multi-layered management framework, because of Java object-orientation, dynamic extensibility, and ease of programming [Lee, 00].

The adoption of the Java technology facilitates the integration of standard SNMP agents, either Java-based or not. SNMP is currently the most diffused monitoring solution in the network management domain. It is a specialized request/reply protocol between two types of possibly remote entities, SNMP managers and SNMP agents. The SNMP manager uses the protocol to request the current value of a state indicator to one of its SNMP agents. SNMP agents act as servers that reply to manager requests by extracting the indicator values from their local MIBs. The MIB specification defines the organization and formats of the maintained state indicators. SNMP has significantly evolved from its first introduction: SNMPv2 has introduced the possibility to organize SNMP managers hierarchically to increase the protocol scalability; SNMPv3 has added security specifications to support agent-manager mutual authentication. SNMP agents compliant to different protocol versions are currently integrated either directly in some operating systems or as components of commercial management frameworks [Stallings, 98].

In case of SNMP agents implemented in Java, it is possible to exploit Java-specific facilities for distributed computing and management integration, such as, respectively, the Java Remote Method Invocation and the Java Management Extensions API. In any other case, Java directly supports at the language level a wide variety of mechanisms and tools to interface with networked heterogeneous components [Lee, 00].

4.2.2 The MAPI Component

Software-based middleware approaches for QoS control and adaptation are usually based on the capacity to monitor dynamically the quality offered by distributed service components, to enable either application-transparent or application-aware management operations; this kind of solutions is demonstrating its viability [Chalmers, 99] [Bellavista, 00a]. Service components operate in a global and open distributed system that is intrinsically heterogeneous. As a consequence, there is the need for on-line monitoring tools that are capable of controlling distributed heterogeneous resources and systems, possibly without imposing any service suspension and providing developers with a uniform interface.

This section presents the architecture and the interface of the MAPI component for the local monitoring of heterogeneous kernel/application resources, while the following section gives some technical insights of its implementation.

Figure 4.3 shows the MAPI architecture. MAPI provides a uniform monitor interface independent of platform heterogeneity, and implemented by the *ResourceManager* class that integrates three different components: *MAPI Profiler Agent*, *MAPI SNMP Agent*, and *MAPI*ResManager*.

MAPI Profiler Agent is able to gather application-level information about the Java environment on the monitored target. It not only collects JVMPI events but also filters and processes them on-line, to offer concise monitoring indicators during service execution. These JVMPI-based monitoring functions are immediately portable on any host that runs the JVM version 2.

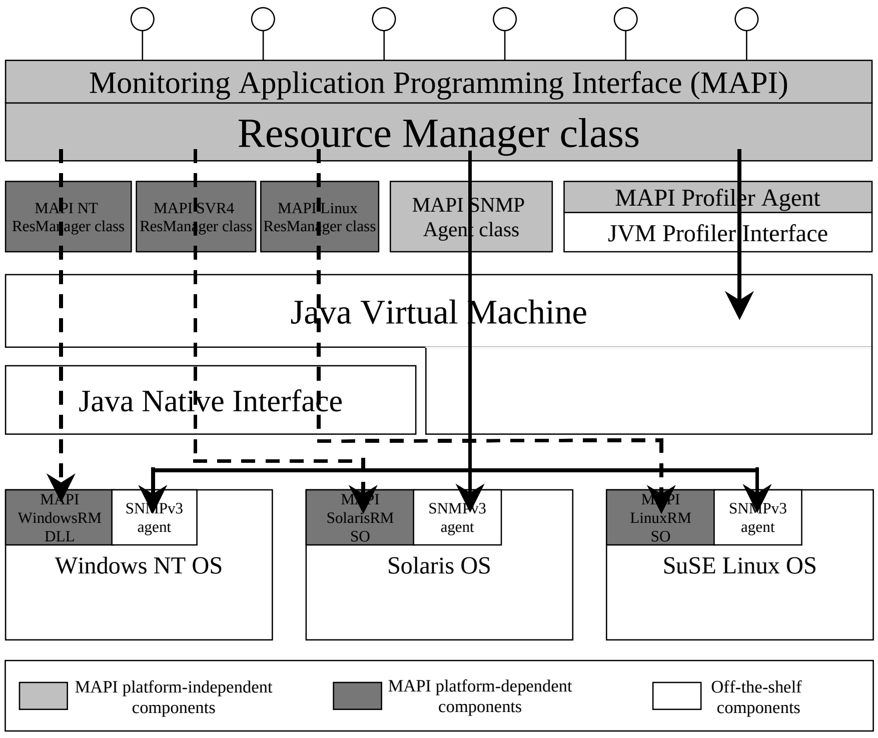


Figure 4.3: The architecture of our Java-based MAPI

According to the SNMP terminology, *MAPI SNMP Agent* acts as an SNMP manager that interrogates the standard SNMP agent available on its local target to obtain kernel-level monitoring data. *MAPI SNMP Agent* not only provides a uniform Java interface by wrapping possibly non-Java SNMP agents. It also implements several local optimizations of the SNMP protocol (mainly, bulk-transfer and filtering described in the following section). In addition, it simplifies the configuration phase of the security parameters needed in SNMPv3, by integrating with the SOMA distributed security infrastructure.

In case the monitored targets do not host the execution of suitable SNMP agents, *ResourceManager* exploits *MAPI*ResManager* classes to integrate with platform-dependent monitoring functions via JNI. These functions are implemented as native libraries with uniform interfaces for different platforms (*MAPI WindowsRM* DLL on Microsoft Windows NT 4.0, *MAPI SolarisRM* SO on SUN Solaris 2.7, and *MAPI*

LinuxRM SO on SuSE Linux 6.2). *ResourceManager* transparently loads at run-time the correct native library for the current monitored target, to provide platform independence.

MAPI offers a set of methods that provide concise monitoring parameters to summarize the current state of the monitored target. Service administrators (or even autonomous software-based service managers) can use MAPI to obtain the dynamic management information to adapt service provision. In this scenario, the overhead is critical, and monitoring results should be prompt and immediately available to managers (see Section 4.2.4). For this reason, MAPI can tune its intrusion to service-specific time constraints: all MAPI methods have a `msec` invocation parameter that indicates the time interval for the refresh of monitoring indicators. This time interval is propagated to all MAPI components to update the statistics of collected JVMPI events, to interrogate SNMP agents, and to invoke native monitoring libraries.

MAPI methods return either an object or an array of objects of the three classes `ProcessInfo`, `NetworkInfo`, and `FileSystemInfo`. The `ProcessInfo` object maintains all the data related to the current `pid` process. Monitored data include the CPU usage (percentage and total time) for any specified process, its allocated memory (physical and virtual), and miscellaneous information on its composing threads. It is also possible to obtain additional data about the threads possibly contained in a process. In addition, in the case of JVM threads, MAPI maintains the reference to the Java thread object, its lifetime, and the number of loaded classes, used monitors, allocated objects, invoked methods, network and file system operations. For non-Java threads, MAPI provides the thread identifier and the percentage/effective time of CPU usage.

The `NetworkInfo` class reports aggregated monitoring data about the usage of the communication infrastructure on the target host. Monitored data include the total number of sent/received UDP/IP packets, of TCP connections and sent/received segments, the percentage of UDP/IP packets received with errors, and the percentage of discarded UDP/IP output packets. These parameters are sufficient to give an overall evaluation of the host traffic conditions and to identify congestion situations.
Finally, the `FileSystemInfo` class maintains general information about the file system of the target (disk free space and its percentage on total size) and detailed data about currently opened files. In particular, for any active process and for any file

opened in the current session, the class returns the opening time and its opening mode (read/write/both/locked).

4.2.3 The MAPI Implementation

The MAPI tool is a Java-based component that permits to monitor both kernel and application state, without modifying the standard JVM. This conformance and the possibility to monitor service components without requiring any intervention on either their source code or their executables are fundamental for the MAPI application to the on-line distributed monitoring of the open Internet infrastructure.

MAPI has required the design and implementation of several ad hoc modules: 1) the *MAPI Profiler Agent* for dynamically configurable on-line monitoring of the JVM state; 2) the *MAPI SNMP Agent* to obtain monitoring data from SNMP agents in execution on the targets; 3) *MAPI*ResManager* and its native libraries (*MAPI Windows/Solaris/Linux RM DLL/SO*) for uniform data acquisition via heterogeneous platform-dependent monitoring mechanisms.

MAPI Profiler Agent permits to configure dynamically the JVMPI-based event notification, and to provide the *ResourceManager* class with concise monitor indicators, obtained as the results of the previous performance history. In a more detailed view, our profiler agent gives the possibility to change the set of notifiable events with no suspension of the monitoring execution, by implementing methods to enable/disable the event notification related to object allocation/deallocation, method invocation/exit, and lock/unlock of Java monitors. The profiler keeps and updates statistics of the monitored events, to provide immediately readable indicators with no need to maintain huge logs of monitoring data. For instance, the profiler traces only the size of the total memory allocated to a Java thread and does not log the full data related to the execution of any system call for memory allocation. In addition, the refresh interval of monitoring indicators can change dynamically to tune the MAPI intrusion depending on service-specific constraints and run-time conditions, as described in the following section.

Figure 4.4 sketches a piece of the *MAPI Profiler Agent* code. When a registered event occurs, the JVMPI signals an event `ev` to the profiler that performs event-specific actions. In particular, the figure shows the initializations made when the class `SocketInputStream` is loaded. After initializing the internal `socketread` variable, the profiler can trace any invocation of the method `socketRead()` by incrementing the `stat->tcp_read` counter, which maintains the account for the TCP read operations of any Java thread in a specified time interval.

These registered data represent a rough estimation of the incoming network traffic produced by Java service components. If there is the need of more precise information about the traffic due to specific Java threads, *ResourceManager* can command *MAPI Profiler Agent* to examine dynamically the invocation parameters of the `socketRead()/socketWrite()` methods. This is possible via the JVMPI-based triggering of `JVMPI_EVENT_OBJECT_DUMP` of the needed objects, at the maximum level of detail (`JVMPI_DUMP_LEVEL_2`). *MAPI Profiler Agent*, of course, behaves differently at default to avoid the excessive overhead of the dynamic generation and processing of object dumps.

```
   JVMPI_Event *ev;                          // JVMPI event reference
   jmethodID socketread = NULL;              // method reference

switch(ev->event_type)
{...
   case JVMPI_EVENT_CLASS_LOAD:
     if(strcmp(ev->u.class_load.class_name,
                "java/net/SocketInputStream")==0)
       {
       JVMPI_Method *meth;
       for(meth=ev->u.class_load.methods; ...; meth++)
         if(strcmp(meth->method_name,"socketRead")==0)
           socketread=meth->method_id;
       }
     break;
   case JVMPI_EVENT_METHOD_ENTRY2:
     stat = tab1.get(ev->env_id);
     if(ev->u.method.method_id==socketread)
    stat->tcp_read++;                         // update TCP statistics
...
```

Figure 4.4: Monitoring the invocation of the `socketRead()` method in *MAPI Profiler Agent*

MAPI SNMP Agent acts as an SNMP manager that locally interrogates its SNMP agent. *MAPI SNMP Agent* is programmed to request monitoring information maintained not only in the standard MIB (monitoring data about network elements and protocols), but also, where supported, in the MIB extensions included in the Host Resources Groups called Storage, Running Software, and Running Software Performance [Waldbusser, 00]. These groups provide the data about resource usage of processes currently in execution to obtain the MAPI `ProcessInfo` and `FileSystemInfo`, while `NetworkInfo` exploits the standard SNMP MIB.

MAPI SNMP Agent can improve the efficiency of standard client/server SNMP operations, especially when dealing with the network transfer of large chunks of monitoring data. It transmits only the changed MAPI indicators to *ResourceManager,* which maintains old values for the non-received parameters. Most important, *MAPI SNMP Agent* locally interrogates its SNMP agent and pre-processes the obtained results to offer concise indicators to possibly remote managers, thus significantly reducing the generated network traffic. In fact, a single MIB variable is usually at a lower level than the MAPI indicators, and an aggregation of multiple variables is required. These aggregations are known as health functions [Gavalas, 00]. For instance, the percentage of discarded IP output packets is obtained by combining five MIB variables:

$$\text{ipPackOutErr} = \frac{(\text{ipOutDiscards} + \text{ipOutNoRoutes} + \text{ipFragFails}) * 100}{\text{ipOutRequests} + \text{ipForwDatagrams}}$$

In addition, *MAPI SNMP Agent* can perform all the operations needed for the support of mutual authentication in case of interaction with SNMPv3 agents. It can obtain dynamically the needed security information from the public key infrastructure integrated with the SOMA programming framework. Finally, it can locally store configuration parameters specific for its SNMP agent (e.g., the supported MIBs), in order to automate the possibly complex phase of initialization at any reboot of the MAPI tool.

If either the SNMP agent or the Host Resources MIB extensions are not supported on the target host, *ResourceManager* automatically enables the gathering of monitoring data via native mechanisms. MAPI native modules extract uniform data by exploiting heterogeneous monitoring mechanisms provided by the target operating system. The *ResourceManager* class employs JNI to load the target-specific native library at runtime. We have currently implemented the native monitoring components for Windows NT (*MAPI WindowsRM* DLL), Solaris (*MAPI SolarisRM* SO) and Linux (*MAPI LinuxRM* SO). Each component integrates with Java via the system-specific classes called MAPI *NTResManager*, *SVR4ResManager* and *LinuxResManager,* as depicted in Figure 4.3.

Figure 4.5 shows a piece of the *MAPI WindowsRM* DLL that accesses kernel and application state indicators maintained in Microsoft system registry keys. In particular, the figure reports the polling of the registry to obtain updated information about the processes in execution. The system call RegQueryValueEx(HKEY_PERFORMANCE _DATA, …) permits to obtain some performance data. The reported invocation returns a perfdata reference to the native method; perfdata is used to access the whole information about a process with identifier PID.

For Solaris and Linux platforms, we have implemented native monitoring modules as dynamic SO libraries that mainly exploit the /proc feature. /proc is a virtual directory that exports kernel and application state indicators as a specific sub-tree of the file system. The *MAPI SolarisRM/LinuxRM* library polls monitoring information about currently executing processes by reading the corresponding files in the /proc directory.

For instance, the ioctl()call, with PIOCPSINFO and PIOCUSAGE parameters, permits to obtain prpsinfo and prusage information, which maintain several data about the identity of a specified process and its CPU usage, respectively. Similarly, *SolarisRM/LinuxRM* native components extract the descriptors of the open files from the /proc/PID/fd virtual directory, where PID is the identifier of the monitored process. File descriptors data are combined with information from the system file table, with an approach similar to the one followed in the implementation of the Unix fuser utility. Aggregated information about the network usage is obtained via the invocation of the standard netstat system call [Nemeth, 00].

```
   RegQueryValueEx(HKEY_PERFORMANCE_DATA,  “232”,  NULL,  NULL,  perfdata, 
&size);
                                 // "232" for process-related data
   RegCloseKey(HKEY_PERFORMANCE_DATA);
   pointer = (PBYTE)perfdata + perfdata->HeaderLength;
   obj = (PPERF_OBJECT_TYPE)pointer;
   pointer = (PBYTE)obj + obj->HeaderLength;
   cnt = (PPERF_COUNTER_DEFINITION)pointer;
   while (cnt->CounterNameTitleIndex != PID)
                   { pointer = (PBYTE)cnt + cnt->ByteLength;
                     cnt = (PPERF_COUNTER_DEFINITION)pointer;
                   }
   pointer = (PBYTE)obj + obj->DefinitionLength;
   inst = (PPERF_INSTANCE_DEFINITION)pointer;
   pointer = (PBYTE)inst + inst->ByteLength + cnt->CounterOffset;
   value = *((jlong*)pointer);
```

Figure 4.5: Monitoring process information in *MAPI WindowsRM* DLL

4.2.4 The MAPI Overhead

It is critical for on-line monitoring tools to limit the overhead on their targets because they are expected to help service management during provision. To validate the MAPI applicability, we have measured separately the costs of the MAPI monitoring modules (*MAPI*ResManager*s, *MAPI Profiler Agent*, and *MAPI SNMP Agent*). All reported costs are averaged on several hundred measurements for Intel PentiumIII 600MHz PCs with either Microsoft Windows NT4 or SuSE Linux6.2, and for SUN Ultra5 400MHz workstations with Solaris7. Hosts are interconnected via 10 Mb Ethernet Local Area Networks (LANs).

To measure the overheads of JVMPI-based monitoring solutions and of JNI-based integration of native monitoring modules, we have installed a Java benchmark application that stresses CPU and memory usage by generating a fixed number of different threads and objects. In particular, the measurements reported in the following refer to the case of 50 benchmark processes in execution, each one with an average number of 5 threads. The benchmark executes first on target hosts with neither applications nor the MAPI component. We have compared the average completion time of the benchmark, called T_{noMon}, with the average time (T_{Mon}) measured with our MAPI tool in execution. The graphs in Figure 4.6 and Figure 4.7 report the overhead percentage (*Overhead%*) introduced by the monitoring tool, defined as:

$$Overhead\% = \left(\frac{T_{mon}}{T_{noMon}} - 1 \right) * 100$$

Let us preliminarily note that the experiment with only the benchmark application in execution is the worst possible case for *Overhead%*. In fact, when the host is either average loaded or even overloaded, T_{noMon} tends to increase faster than T_{mon}, and consequently their ratio *Overhead%* tends to decrease. This has been validated by taking *Overhead%* measurements with several general-purpose benchmark tests running: the measurements have confirmed the expected behavior, with an average decrease of about 1.0-1.5% of the *Overhead%* parameter in conditions of average load.

Figure 4.6 depicts the *Overhead%* introduced by *MAPI*ResManager* to monitor all the information contained in `ProcessInfo`. The diagram of *Overhead%* is drawn function of the sample time for the three supported platforms. The refresh time represents the interval between two successive invocations of the native monitoring modules via JNI. *Overhead%* exhibits a linear dependence on the reciprocal of the refresh time. We have obtained analogous trends in intrusion measurements in case of native monitoring of `NetworkInfo` and `FileSystemInfo`.

In general, all tests show that *MAPI*ResManager* causes a limited overhead (*Overhead%* lower than 3%) when the refresh time interval is greater than 3 s. On the one hand, this overhead can be considered acceptable because the refreshing of monitor indicators with this time period is sufficient for most Web-based services. In fact, native modules continuously collect monitored events, and the refresh time is only the period to interrogate native monitoring results. On the other hand, however, the overhead imposed by *MAPI*ResManager* is sensibly higher than the one of *MAPI SNMP Agent* presented in the following. This has suggested to prefer the latter if it is available and supports the Host Resources MIB extensions.

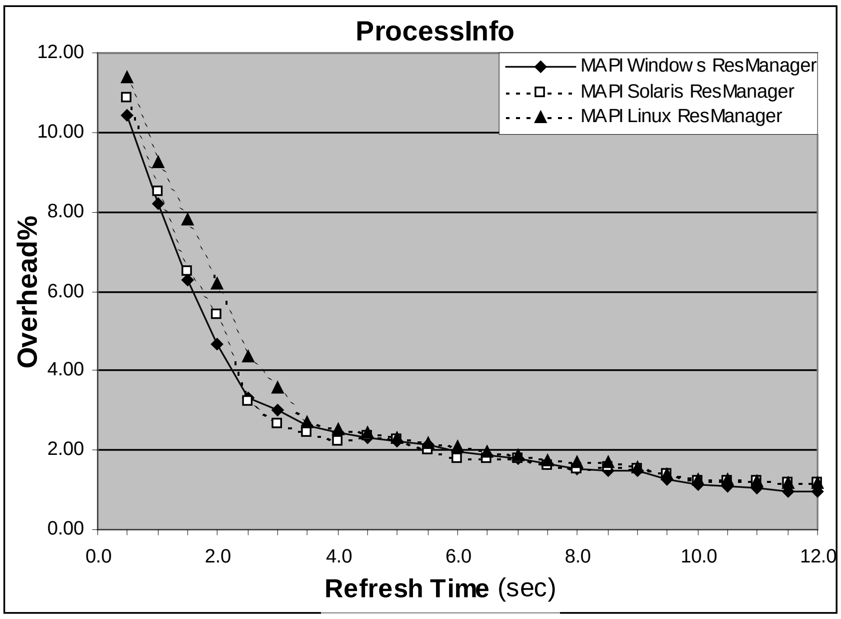


Figure 4.6: *Overhead%* of *MAPI*ResManagers* for process monitoring

We have also measured the overhead introduced by *MAPI Profiler Agent*. It requires the JVM to notify continuously certain kinds of events (not only at polling times), and the refresh time represents the interval to process the collection of observed events in order to obtain the desired concise monitoring indicators. Figure 4.7 depicts the different contributions of the *Overhead%* related to data access and to different JVMPI kinds of events, i.e., monitor, method and object tracing. The figure reports only the results obtained for the Solaris platform, because they are very similar to the ones for Windows NT and Linux, with differences lower than 3% of the maximum *Overhead%*.

The JVMPI notification mechanisms are scarcely intrusive under different load conditions and independent of refresh times. Object tracing has shown to be the most relevant factor in the *MAPI Profiler Agent* intrusion because of the large amount of data it requires to receive and collect in the profiler. Apart from object tracing, JVMPI notification introduces a very low overhead (lower than 0.6%). The refresh time affects only the overhead due to the processing of collected events and to the reading of MAPI indicators. In any case, the total overhead can be kept under 2.0% when the refresh interval is greater than 2 s and object tracing is disabled.

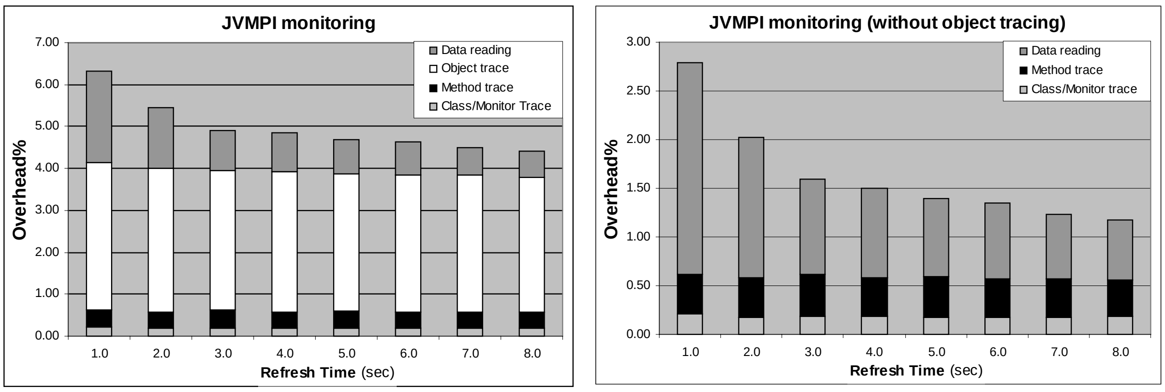


Figure 4.7: *Overhead%* of the *MAPI Profiler Agent* for Java thread monitoring

4.3 On-line Distributed Monitoring and Management

We have integrated the MAPI component in the SOMA profile for integrated management both to exploit the MA technology for distributed monitoring and to build an on-line monitoring tool for SOMA-based service components. We have also implemented a MAPI-based distributed monitoring tool in terms of mobile agents that cooperate to enforce distributed management policies. Monitoring agents can migrate close to the networked resources to observe and manage autonomously on behalf of system administrators. We exploit the MAPI distributed tool to monitor and manage the QoS provided by SOMA-based distributed applications. In particular, Section 4.3.1 presents the employment of the distributed tool for the dynamic control of resource usage by SOMA agents, for sake of accounting and security.

We have designed and implemented a distributed tool in terms of SOMA agents that use MAPI to monitor any single target. The MA technology is adopted because it can improve the performance in collecting/processing distributed monitoring information and facilitates the coordination of groups of autonomous monitoring agents, as detailed in the following.

The SOMA profile for integrated management provides developers with an API to invoke a set of common system management operations that are implemented in terms of the SOMA monitoring and management agents described in the following. In addition, it is easy to tailor new agents to new specific administration needs. The following list gives a few examples of functionality already available in the SOMA profile:

- monitoring the state of the distributed system;
- controlling and coordinating replicated resources;
- dynamically installing new services;
- helping in the configuration of any new or reinserted node;
- shutting down the whole system by ensuring a minimal survival service level.

The MAPI-based distributed tool is the result of the interworking of two types of agents: monitoring *Managers* and monitoring *Explorers*. Each *Explorer* agent is in charge of collecting monitoring data from one set of targets (i.e., *target domain*), usually belonging to the same network locality. The *Manager* agent commands the *Explorers*, combines their monitoring results and presents a global view of monitored domains to system administrators.

There are several possible organizations, with different hierarchical levels and numbers of *Managers/Explorers* per target domain. In our organization, each system administrator can delegate several management operations to one or more *Managers* that operate autonomously. Each *Manager* coordinates its set of *Explorers*, one per each target domain.

For instance, a *Manager* agent can be in charge of controlling the storage devices of its administered domains. If the *Manager* ascertains that the available space is lower than a specified threshold, it can alert the system administrator and even command simple actions to remove useless files (e.g., files in the `/tmp` directory) on overloaded target hosts/domains. To collect information about the overall state of its administered domains, the *Manager* coordinates the operations of its *Explorer* agents. In particular, it can ask them to gather specific monitoring data, with specific alert thresholds and refresh time intervals, and can also command management operations on controlled *Explorers*. In addition, a *Manager* can create new monitoring *Explorers* at run-time to go and control existing/new target domains.

Explorer agents periodically migrate to their target domain hosts to invoke there the MAPI functions, in order to observe locally both kernel and application state indicators. *Explorers* can be launched with alert thresholds that the responsible *Manager* is authorized to modify dynamically: when thresholds are exceeded, the *Explorer* can either notify its *Manager* or autonomously take corrective operations on service components. These solutions allows *Explorers* to reduce the network traffic due to distributed monitoring and to perform autonomously local management operations without requiring the intervention of either *Managers* or system administrators.

Explorers can also invoke *MAPI Profiler Agent* functions to control and manage local Java threads. In particular, they can modify the priority of running threads and can force thread suspension/termination. These functions help in controlling the execution of Java applications, and make possible to limit their resource consumption at run-time, as described in the example of the following section.

Managers can tune dynamically the overhead introduced by *Explorer*-based distributed monitoring by modifying at run-time the time period with which *Explorers* visit their target domain hosts. In addition, *Managers* can command *Explorers* to invoke the *MAPI Profiler Agent* methods to enable/disable the notification of specific kinds of events, thus adapting dynamically the collection of monitoring data to the currently enforced management policy.

The SOMA profile for integrated management can represent the basic middleware to enable the on-line QoS management and accounting of resource consumption in most classes of Web services. To show the applicability and the usage of that profile, in the following section we present the example of the control and limitation of SOMA agent operations and, consequently, of the resource consumption of SOMA-based Web services. The ultimate goal is to provide service administrators with an API to account agent operations on their responsible principals and to protect target hosts from possible denial-of-service attacks due to either malicious or badly programmed agents.

4.3.1 The On-line Distributed Control of MA Resource Consumption

SOMA administrators can specify management policies to automate the distributed control of agents. Policies can be enforced over either a single target host or a whole target domain, and include conditions and management actions. Conditions are expressed as even complex functions of the MAPI monitoring indicators: the trespass of the specified conditions triggers management actions to correct agent behavior. Admissible conditions require to monitor not only SOMA agents and their Java threads, but also system/service components external to the JVM.

An example of local condition specifiable in SOMA is “on target host A, if the CPU usage percentage of the local Web server (`httpd` process) is higher than `t%`, the total CPU usage percentage of all the agents of principal Paolo cannot be higher than `10*(1-t%)`”. If the scheduler at host A assigns `httpd` threads a higher priority than the one of SOMA agents, this policy attempts to preserve the throughput of the local Web service when highly loaded, by monitoring and limiting the number of CPU cycles consumed by Paolo’s agents. There is no limit on agent execution, instead, when the Web server is less loaded.

As an example of distributed condition, SOMA administrators can request to enforce “the total number of socket read/write operations performed by Paolo’s agents cannot overcome `sock#` per day in the target domain B”. The aim of this policy is to limit the agent consumption of network resources during a time window, by assuming in first approximation that the number of socket operations is proportional to the sent/received network traffic. This condition should be enforced over the whole set of targets.

Apart from monitored conditions, any policy specifies a time interval to control condition respect and corrective actions to execute in case of trespass. At the moment, the integrated management profile API permits three different corrective actions: 1) to decrease the priority of the Java threads of breaker agents; 2) to suspend breaker agents for a specified interval time, by exploiting the SOMA persistency mechanisms [Bellavista, 00b]; 3) to force the termination of breaker agents.

4.3.2 The Performance of Distributed Monitoring

Distributed monitoring is based on the interworking of mobile *Explorer* agents with the local MAPI modules. The size of mobile agents vary from about 8kB (at first migration, without carrying any monitoring indicator) to 15kB (at the end of exploration, including all the monitoring state of the target domain). The average time needed for migrating *Explorers* from one target host to another one within the same LAN is 113ms (307ms with enabled controls on the integrity of agent code and state) in the case of 8kB, and is 189ms (477ms with integrity enabled) in the case of 15kB. [Bellavista, 00c] fully describes the costs of agent migration, when enabling the different security levels provided in SOMA.

Figure 4.8 reports the time for an *Explorer* to collect monitoring indicators using *MAPI SNMP Agents*, in function of the number n of hosts in the target domain. *MAPI SNMP Agent* filters and pre-processes the monitoring data that *Explorer* has to collect. On the basis of a wide set of resource consumption policies that we have enforced for SOMA agents, we have measured an average 1/3 reduction factor for the size of MAPI indicators collected by the *Explorer* with respect to the size of corresponding raw MIB values. This kind of optimization is not possible when adopting a traditional client/server approach where a fixed and centralized manager remotely interrogates the different SNMP agents involved (SNMP Client/Server graph in the figure). In addition, the MA technology is particularly suitable when the target domain includes different LANs, interconnected with low bandwidth links (in our measurements, two Ethernet LANs with n/2 hosts and connected via a 56kb modem link). In this case, *Explorer* uses the slow link only once to migrate from one LAN to the other, while the Client/Server solution exploits the link at least for n/2 SNMP requests and n/2 SNMP replies, wherever the centralized manager is located.

All results show that, in case of either significant data filtering or heterogeneous network connectivity, the adoption of mobile agents can reduce both time performance and generated traffic. Our results confirm model estimations and first experimental measurements of recent research work on MA-based SNMP distributed monitoring [Gavalas, 00] [Baldi, 98].

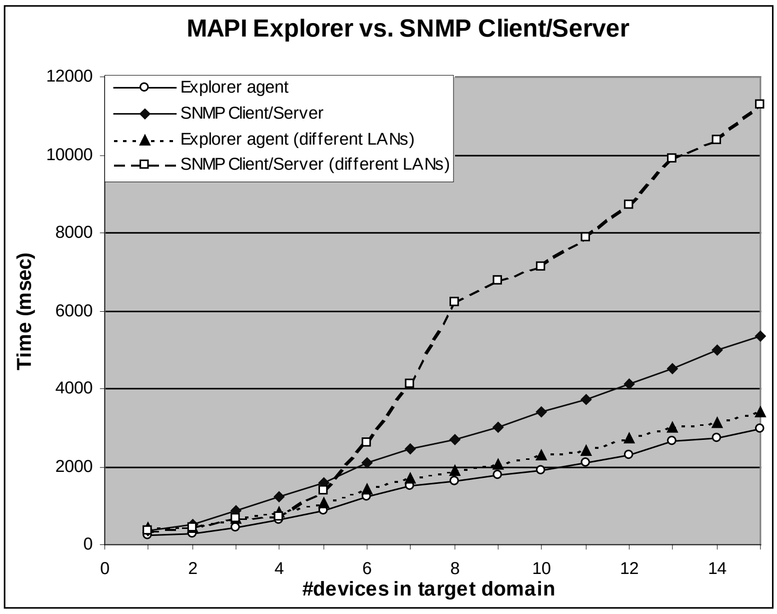


Figure 4.8: MAPI *Explorer* vs. SNMP Client/Server to monitor one target domain

4.4 Video on Demand Service Management

The main goal of the SOMA profile for integrated management is to provide application developers with a complete and flexible middleware support for the management of complex network services, even obtained by tailoring and composing existing ones, and to dynamically introduce new services in the existing infrastructure without suspending operations.

Apart from the already mentioned tools for the distributed monitoring of heterogeneous services and for the control of resource consumption of SOMA agents, we have exploited the SOMA profile in the Video on Demand (VoD) application area. We have designed and implemented a VoD service, called Mobile Agent-based Distributed Architecture for Multimedia Applications (MADAMA) [Bellavista, 01a], in collaboration with the University of Naples, in order to unify their experience in VoD systems [DIVA] with our competence in MA-based middleware facilities.

MADAMA is based on the set of profile facilities previously described, and implemented in terms of SOMA agents that are distributed over the paths between the source and the targets of the video stream. MADAMA permits users to require a QoS level for any multimedia stream, and allows managing and adjusting the requested quality during service provision, to respond to dynamic modifications of network resource availability. The monitoring, control and management of the offered QoS level are currently made at the application-layer; we are extending the MADAMA implementation to integrate network-layer technologies, such as ATM, that provide direct control of QoS parameters, with a solution that is similar to [Kassler, 99].

The VoD service is realized by coordinating two different types of SOMA agents built on top of the integrated management profile: the QoS Negotiators (QoSNs) that define and grant a specific level of quality for the service, and the Admission Controllers (ACs) that manage the resources to be engaged by local intermediate nodes (see Figure 4.9).

ACs are present on every node of the network; this assumption is not severe because they are implemented by mobile agents that can move and be installed whenever they are needed. Each AC monitors and manages local resources (it is a specialization of the SOMA Explorer agent presented in Section 4.3). In addition, it keeps track of the current commitment of local resources to already accepted multimedia streams. The flow specifications of streams are recorded in a local table of `<receiving-host, bandwidth, delay, loss>` tuples [Gibbs, 94]. Any tuple represents the statistics of VoD traffic between the local and the receiving host: the first time, it contains values calculated upon a short sample of communication; then, it is updated by monitoring real traffic of current VoD sessions. ACs are in charge of answering to reservation requests from QoSNs.

MADAMA requires the coordination of a set of QoSN agents located at the source, at the target and at some intermediate nodes. QoSNs maintain session state: they record user preferences and flow specifications for a video stream. QoSNs evaluate the feasibility of meeting these requirements against the local AC database of monitoring indicators and exploit the communication facility of the SOMA basic configuration to perform the negotiation phase for the definition of the achievable QoS. After the negotiation phase, during multimedia streaming, any QoSN is in charge of receiving packets from the previous QoSN and of forwarding them to the next QoSN. When multiple video streams interest the same network node, one QoSN can handle all of them.

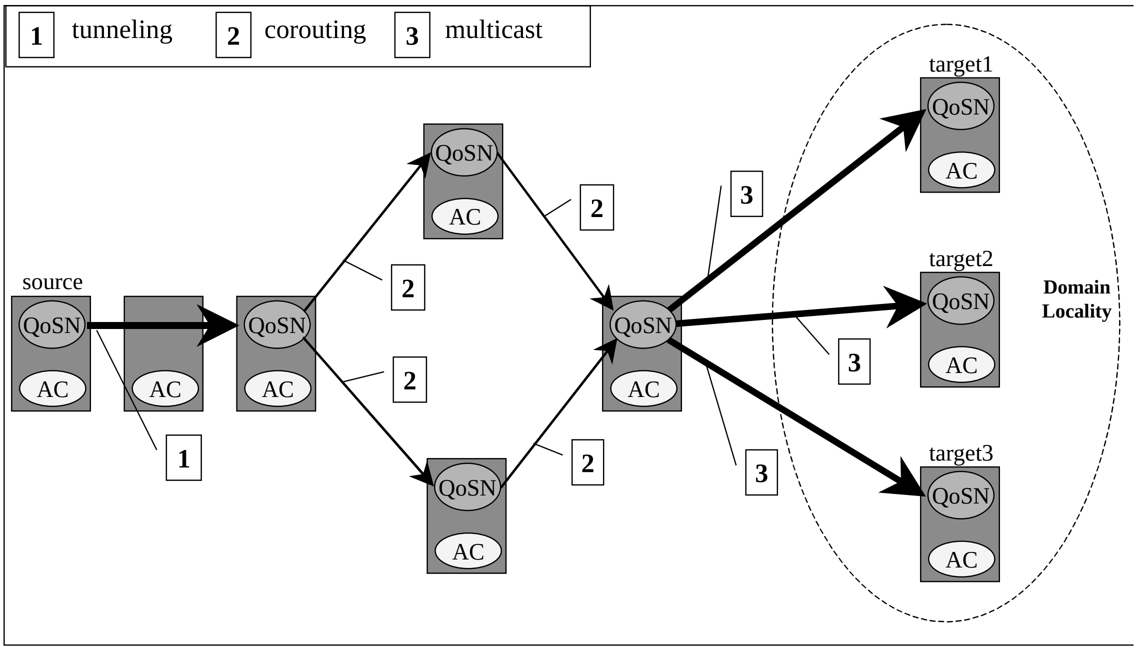


Figure 4.9: Tunneling, corouting and multicast in MADAMA

Let us first consider the case of a video stream addressed to one target only. The path between the source and the target is automatically determined at run-time, by tracing the route via one dummy packet sent from source to target (it can be also predetermined by the MADAMA source according to some previously collected routing information). QoSNs move to the chosen hosts on the path and interrogate the AC database: if available resources are not enough for the desired QoS, QoSNs can coordinate and reduce their requests by scaling the stream (at the moment, by dropping frames in Motion JPEG streams or by reducing resolution in MPEG-2 ones [Gibbs, 94]). Only if these diminished reservation requests cannot be satisfied, the VoD service is denied.

After a successful negotiation phase, the (possibly scaled) multimedia stream starts to flow. During the video distribution, a link can fail or its quality can deteriorate, thus making impossible to a particular QoSN to maintain the negotiated quality. In that case, the interested QoSN can enhance the throughput of its link via stream striping on non-corouted paths [Traw, 95]. In this case, it sends back a message to temporarily stop the stream, and sends forward a message to suspend updates in AC tables on the path. Then, it sends its clones to handle new non-corouted paths and starts the negotiation phase with the clones. When negotiation completes, the QoSN sends back a message that restarts the stream: apart from a delay in receiving the stream, the VoD target goes on transparently.

In the case of multicast distribution of the same video stream (for N targets), the generated network traffic can be limited by exploiting location awareness of agents. Mobile agents give an active role to intermediate nodes that can perform acknowledgement aggregation and buffering to reduce implosion problems, and can realize soft state caching of multicast data to permit local recovery with no need to request all retransmissions to the multimedia service provider [Calderon, 98].

In our opinion, however, the most significant contribution of the MA technology to VoD multicast is in the mobile agents capacity of dynamically adapting multimedia traffic to receiver requirements and network conditions. In fact, the common usage scenario is the multimedia distribution over the Internet to a group of heterogeneous receivers with different processing capabilities and different (possibly time-changing) bandwidths available in the paths, with the goal of optimizing network traffic according to the QoS requirements of the receivers. Various approaches are possible [Li, 99] (see Figure 4.10). The simplest one is usually called the *single-stream multicast*: the sender simply adjust the multimedia QoS to the worst receiver [Bolot, 94]. More sophisticated and fair is the *replicated-stream multicast* approach: the sender distributes a small number of multimedia flows with different QoS to receivers with different capabilities [Li, 96]. Another case, indicated as *layered multicast*, exploits the possibility for many compression techniques [Tudor, 95] [McCanne, 96] to separate the multimedia flow in different layers. Each multimedia client receives a subset of the layers suited to its QoS requirements.

The SOMA integrated management profile is a suitable technology not only for the deployment but also for the enhancement of these multicast architectures. Agents are able to locally operate on the exchanged data to perform suitable transformations on the multimedia flow. In case of replicated-streams, they can reduce network traffic by avoiding the sender to transmit different flows at different QoS levels: agents themselves are able to reduce the quality of their incoming multimedia flow before distributing it to the next node (see Figure 4.11). The time needed for agents to perform the transformations on the flow does not affect the quality of the service once the client has started to receive the multimedia flow. If adequately supported by buffering policies at the intermediate nodes, agent operations on data only increment the service startup delay.

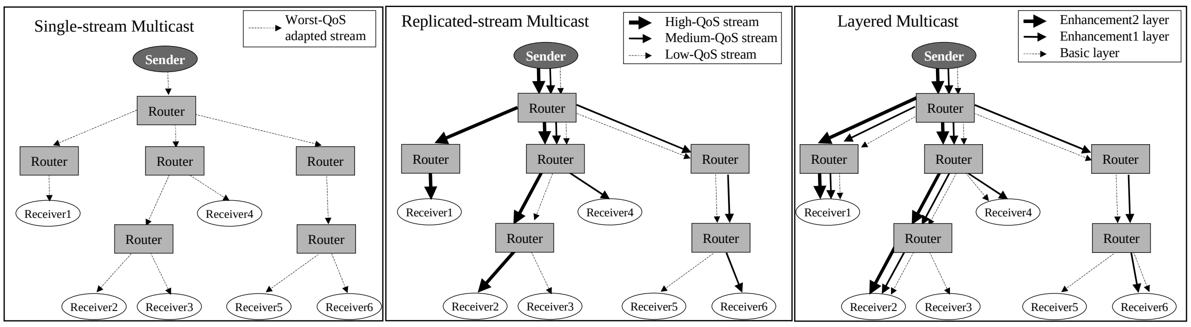


Figure 4.10: Single-stream, replicated-stream and layered multicast

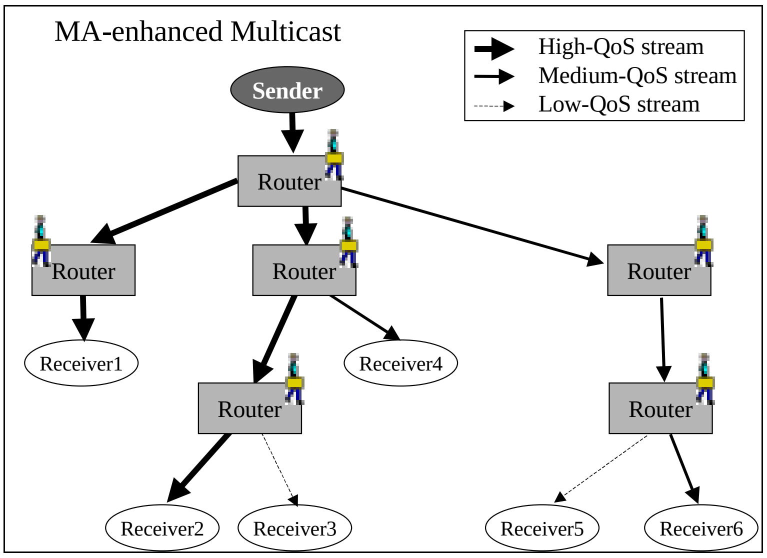


Figure 4.11: Multicast and differentiated QoS in MADAMA

4.4.1 The MADAMA Performance

The MADAMA service requires the presence of the above described infrastructure composed by distributed and coordinated agents. Therefore, before the multimedia stream can start to flow, users have to wait for the completion of a setup phase in which the service provider determines the path between the source and the target, and distributes all ACs and QoSNs needed on participating nodes. After the setup phase has negotiated the service levels, the stream can flow from source to target. During service provision, the flow can also be dynamically adapted, to adjust the required QoS level at runtime with a best-effort approach, or to dynamically organize corouted paths, with a further distribution of agents on new nodes. Dynamic adaptations introduce overhead that is only a percentage of the one required by the setup phase.

For that reason, we report about the setup costs in a normal usage scenario. In this phase, one lightweight agent (about 1KB-sized) is sent from the source to the target to identify the path for the multimedia stream. This agent reports back to the source the information about how many ACs (about 6KB-sized) and QoSNs (about 4KB-sized) have to be instantiated and to be sent in parallel to interested nodes. We have considered the worst case when none of the intermediate nodes has neither the AC nor the QoSN agent. In more realistic scenarios, hosts may have already the AC agent running for purposes of remote monitoring and diagnosis.

In addition, the VoD service should be typically carried out in untrusted environments, where cooperating agents have to pass integrity and authentication checks before being allowed to operate to local resources. The SOMA complete configuration permits users to choose which subset of functionality are used by specific services. In that way, services can obtain the most suitable trade-off between performance and security, depending on the level of trust of the environment and on the criticality of the application domain. In particular, the VoD service provided in a trusted environment may omit authentication and integrity checks, with considerable time saving on the setup overhead.

We report the setup time for the MADAMA VoD service in a non-dedicated network consisting of the interconnection of several LANs. In particular, the results apply to a case where the video source is 8 hop far from the target, i.e. the multimedia flow has to pass through 7 intermediate non-tunneled nodes to reach its target. Any intermediate node hosts the default place of the domain it belongs to, and any domain abstraction models a distinct real LAN in our university organization. The LANs are composed by heterogeneous hosts (PentiumII PCs with Windows NT 4.0 and Sun SPARCstation with Solaris 2.5) and are based on different communication technologies, mainly Ethernet and Fast-Ethernet.

In the case of untrusted environments, the multimedia source has to calculate a 1024-bytes blocksize MD5 hash of all sent agents and to sign them with its 1024-bits RSA private key. Any intermediate node must perform the security checks to verify the signature and the hash. The average setup time measured in this scenario is 10907 ms and shows the feasibility of the approach in the case of complex interconnection of non-dedicated local networks.

In the case of trusted environments, MADAMA gives the possibility of providing the VoD service with no security checks, with a considerably reduced average setup time (7634 ms that is 30% less of the first case). We are experimenting other significant time reductions via the utilization of the HotJava just-in-time compilation techniques and via the recent introduction in our organization of fast communication technologies based on FDDI and ATM.

5 The SOMA Profile for Mobile Computing

The diffusion of the Internet permits an almost ubiquitous availability of attachment points, and users expect to access Internet services independently of their physical location, e.g., at their workplace, at home, at a public telephone-Internet box. In addition, advances in cellular telecommunication and device miniaturization are forcing to open Internet services to the increasing number of portable network devices, e.g., laptops, Wireless Application Protocol (WAP) phones and Personal Digital Assistants (PDAs) [Lewis, 98].

This trend suggests considering an advanced infrastructure to support different forms of mobility. The support for *user mobility* should provide users with a uniform view of their preferred working environments, i.e. user preferences and subscribed services, independently of their current positions in the network [Kumar, 96]. The support for *terminal mobility* should allow devices to move and connect to different points of attachment in the global network. An emerging issue is the dynamic adaptation of mobile-aware resources and services, to be automatically retrieved by mobile users/terminals independently of their current location [Jing, 99]. In the following, we call this issue *mobile access to resources*.

The ultimate goal is to permit the movement and execution of a whole computing environment in heterogeneous networks, while maintaining both the Internet service provision and the access to distributed resources even in case of network disconnection. Mobile computing issues span from the network layer to the application one, and require the design and implementation of a *mobility middleware* that integrates suitable support protocols, mechanisms and tools. The idea of a global infrastructure has already been accepted in the communication area, where the Telecommunication Information Networking Architecture standardizes a layered architecture to integrate service components, network operators and management functionality.

The mobility middleware should be capable of dynamically reallocating and tracing mobile users and terminals, and of permitting communication and coordination of mobile entities. In addition, open and untrusted environments impose to deal with different devices and systems and to grant the needed security level.

Solutions to these issues require compliance with standards to face heterogeneity and to interoperate with legacy components, and the presence of a thorough security infrastructure based on standard cryptographic mechanisms and tools.

The realization of a mobility middleware can significantly take advantage of the adoption of the Mobile Agent (MA) technology because many MA requirements coincide with mobility ones [Kovacs, 98] [Kotz, 97] [Lipperts, 99].

Mobility requires *dynamicity,* intended as the possibility of modifying and extending the middleware with new components and protocols to adapt to evolving service/user requirements at run-time. Dynamic distribution/modification of code and dynamic resource binding are very similar in case of both mobile agents and mobile users/terminals. Mobile agents benefit from the additional flexibility of moving code together with the state produced by performed computation.

Mobility stresses the *security* issue, to authenticate mobile users/terminals, to authorize the access to system resources and to grant secrecy and integrity in communications. After the pioneering results of IBM Aglets [Lange, 98], recent MA research activities have identified solutions for security problems: several MA platforms provide flexible mechanisms and policies to grant the most suitable security level [Tripathi, 00] [Bellavista, 00c]. For instance, many MA systems integrate with Public Key Infrastructures to simplify authentication of mobile users/terminals.

Mobile users/terminals need *interoperability* when moving to unknown hosting environments to interact with available resources and services. To face similar problems, the MA research has promoted interoperable and standard interfaces. For instance, some MA platforms already provide compliance with CORBA and related standards, such as the MASIF and the FIPA specifications [Milojicic, 98] [FIPA], as described in Section 3.2.2.

Mobile computing can greatly benefit from the possibility of *asynchronicity* between requests of user/terminal operations and their execution. For instance, wireless connections impose strict constraints on available bandwidth and on communication reliability, and force to minimize the connection time for the wireless device support. The MA paradigm does not need continuous network connectivity because connections can last only the time needed to inject agents from mobile terminals to the fixed network. Agents are autonomous and permit to carry on services even when launching users/terminals are disconnected and to give results back at their reconnection [Kovacs, 98] [Kotz, 97] [Lipperts, 99].

The mobility middleware should give application designers location awareness to perform service-specific optimization and to adapt to local resources. A mobility-enabled application should face the possibility of mobile users to change location and should dynamically tailor to the properties of the current network connections and to the characteristics of hardware devices. *Location awareness* of the MA paradigm can propagate allocation visibility up to the application level, thus simplifying the support of dynamic QoS adaptation to the local situation [Kotz, 97] [Lipperts, 99]. In addition, the MA autonomy from users simplifies dynamic *personalization*: mobile agents can follow user movements and tailor service results depending on personal preferences.

Because of these considerations, we have decided to design and implement a SOMA-based specific profile for the support of mobile computing. The mobile computing profile runs on top of the SOMA complete configuration, and offers middleware facilities for the deployment of Internet services in contexts of user/terminal mobility and of mobile access to resources. These middleware facilities are organized in three modules: the User Virtual Environment (UVE), the Mobile Virtual Terminal (MVT), and the Virtual Resource Management (VRM). The UVE facility is in charge of providing users with a uniform view of their working environments independently of current locations and currently used terminals. The MVT facility extends traditional terminal mobility by preserving the whole terminal execution state to be restored at new locations (including active processes and subscribed services). The VRM facility permits mobile users/terminals to maintain the access to resources/services by automatically re-qualifying the bindings, and also to move specific resources/services to permit load balancing and replication.

In the following, after presenting an overview of the recent related work on user mobility, terminal mobility and mobile access to resources, the chapter will describe the design, implementation and functionality of the three modules of the SOMA profile for mobile computing. Then, Section 5.5 will report and discuss the costs of the middleware facilities, while Section 5.6 will end the chapter by presenting a real usage scenario where the mobile computing facilities provide the middleware for the deployment of mobility-enabled Internet services.

5.1 Related Work

Several researches deal with mobility, with different approaches and goals. This section presents some common approaches to user and terminal mobility, and also describes the first directions of solution in the area of mobile access to resources. Figure 5.1 sketches an example with a mobile user and a mobile terminal, which change their location but maintain access to a printer service. In this example, the mobile terminal chooses to maintain its previous binding to the currently remote printer (Figure 5.1 – case a), while the mobile user re-qualifies the binding to an equivalent printer in the new hosting locality (Figure 5.1 – case b).

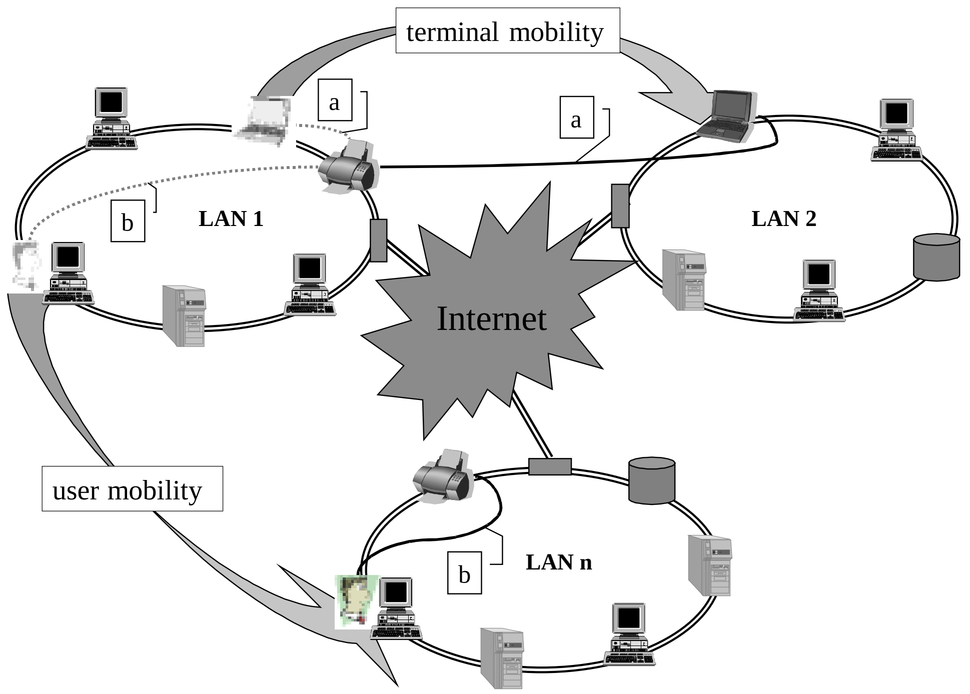


Figure 5.1: An example of user mobility, terminal mobility and mobile access to resources

5.1.1 User Mobility

User mobility provides users with a uniform vision of their preferred working environments independently of both terminal properties and current physical location. The support for user mobility has to authenticate user access and to organize her working environment according to the information contained in her user profile. The profile describes the graphical interface information and all user preferences, e.g., the default language, the required security level, and the subscribed services. Profiles may also include user-specified information to adapt the working environment to the hardware characteristics of the current terminal. For instance, if the user is connected via a PDA with limited bandwidth and limited graphic resolution, it is worthwhile to discard large size images.

Several research activities on user mobility focus on the definition and management of suitable user profiles. The UMTS proposes a service infrastructure based on the concept of Virtual Home Environment (VHE) [UMTS] that presents users with the same personalized features, user interfaces and services independently of the current hosting network. The provided user working environment should depend on profile preferences, terminal equipment and current network conditions.

The World Wide Web Consortium promotes the Composite Capability/Preference Profiles (CC/PP), a standard proposal for both the representation of profile information and the exchange protocol, based on the Resource Description Format encoded in XML [W3C CC/PP]. WAP mobile phones are going to adopt CC/PP to tailor the provision of Internet services to their specific characteristics [WAP]. In addition, the Foundation for Intelligent and Physical Agents (FIPA) is working to define an agent interoperability framework for nomadic support that deals with the information for user profile management and mobile device characteristics [FIPA].

Let us note that several proposals tend to integrate in the user profile both user preferences and information about current terminal characteristics, e.g., hardware/software platform and screen resolution [WAP]. Though personalization and adaptation of services need both user- and terminal-dependent information, we claim that the two dimensions should be cleanly de-coupled. User profiles should contain only user-related information, and devices should inform the support infrastructure of their characteristics in an independent way. This achieves maximum flexibility and reusability in all real scenarios where, for instance, the same user can exploit a set of different terminals with different characteristics.

5.1.2 Terminal Mobility

Terminal mobility refers to the possibility of moving portable devices while maintaining current working sessions with different degrees of transparency for running applications. In more detail, roaming computing focuses on communication-layer mechanisms to preserve communication channels transparently to applications, while nomadic computing mainly aims at reactivating network connectivity after migrations.

Many state-of-the-art proposals face roaming computing at the lower layers of the OSI protocol stack. Network-layer protocols, such as *Mobile IP* [Bhagwat, 96], associate a mobile host with two IP addresses. The first one represents the current point of attachment to the network; the second one reflects the mobile host home address, i.e. the address of a fixed care-of entity that has the duty of tracing the current position of the mobile host. Mobile IP is backward compatible with IP but cannot achieve optimal routing because it always requires packets to pass through the care-of (triangle routing problem). Another solution is *IPv6* [Bhagwat, 96] that adopts an approach similar to Mobile IP but also provides acceptable performance and excellent scalability by permitting senders to cache information about current location of their mobile destinations. The process of acceptance and adoption of IPv6, however, is likely to last long, as for all new protocol proposals.

The issue of wireless communication in local area networks is addressed by the IEEE 802.11 standard protocol. It spans from the physical-media layer that defines frequencies and their usage, to the media access layer that defines basic packet framing and headers. Unfortunately, not all the wireless device producers have accepted the 802.11 and full multi-vendor interoperability is only a long-term hope [IEEE P802.11].

The long acceptance process of new protocols has recently motivated and suggested the proposal of programmable network architectures to simplify protocol prototyping and deployment [Psounis, 99]. While programmable networks are still an open research issue, there are other TCP/IP-based solutions for specific aspects of the mobility support. For instance, the Dynamic Host Configuration Protocol (DHCP) can automate the configuration of nomadic hosts via dynamic assignment of temporary IP addresses [Perkins, 99].

As a final consideration, it is still unclear whether network-layer approaches can provide a flexible solution to fundamental mobility issues, such as security and interoperability, that are faced at a higher level of abstraction and can benefit from standard tools at the application level [Bolliger, 98].

5.1.3 Mobile Access to Resources

Research activities about mobility have not fully addressed the possibility of maintaining access to available resources and services while moving. This goal requires mobility-enabled naming solutions to keep the information about availability and allocation of resources and services. Naming solutions suitable to face mobility can be classified in two main categories, *discovery* and *directory* services.

Discovery services usually employ simple protocols to obtain information about entity location (address and simple configuration data) with a minimal knowledge of hosting environments. Recent researches have produced widely accepted distributed protocols to maintain information about current resource availability in local networks and to answer simple queries. Different implementations reflect the different types of resources they classify, from simple embedded devices, as in the Simple Service Discovery Protocol of Microsoft proprietary Universal Plug and Play [UPnP], to more complex service components, as in Jini and in the Service Location Protocol (SLP) [Perkins, 99].

Directory services usually organize names and properties for registered entities in a very flexible way and provide operations to browse all registered information with complex search patterns. Apart from the traditional work on directory standards, such as the X.500 Directory Access Protocol (DAP), the Internet community has defined lightweight protocols with a simplified interface, such as the Lightweight DAP (LDAP), that run directly on top of TCP/IP [Howes, 97].

However, the automatic maintenance and updating of bindings to resources/services in mobility scenarios via discovery and directory services is still in its infancy. The first proposals agree on the necessity of middleware functions to provide, for instance, the transparent reestablishment of TCP/IP connections in nomadic computing [Hansen, 98], and the automatic adaptation of service flows to the type of resources that are currently available to PDAs [Fox, 98].

5.2 User Virtual Environment

The UVE facility is the component of the SOMA profile for mobile computing that supports the concept of VHE. It permits users to connect at different locations, possibly via heterogeneous terminals, while maintaining the personal configurations indicated in user profiles. Users can specify profile information at the first registration, and modify it at any time. The profile information contains not only usual attributes, such as the preferred icon arrangement on the display. It can also include more complex data, such as personal X.509 certificates, the resources requested to the hosting environment for ordinary tasks (e.g., a printer of a specified quality), and user constraints to direct QoS adaptation to the current terminal type (e.g., in a connection via a mobile phone, the user may request the mobility middleware to discard large size attachments in incoming mail).

The UVE facility stores user profiles at the *user home*, i.e., a fixed host where the user has first registered. For increased reliability and efficiency, UVE can replicate and cache user profile copies at several other locations. UVE can make user profiles globally available, by exploiting the directory service integrated in the SOMA complete configuration. The directory service can transparently map requests to the most convenient UVE server available, according to any user preferred metric (network distance, response time, load balancing, etc.).

The UVE facility greatly benefits from its MA-based implementation because SOMA agents simplify the dynamic distribution of UVE information and the realization of fault-tolerant solutions via agent replication. In addition, the persistency facility in the SOMA basic configuration provides automatic/manual mechanisms to save the state of the user session, and to move and restore it to the new location, where the user can find previously configured services, possibly adapted and scaled. UVE can also permit to yield execution results to users independently of their current location. When users are disconnected, UVE commands the SOMA-based middleware to temporarily freeze the agents with the results. These agents are restarted only at user reconnection.

UVE must ensure secrecy of the information in user profiles: the security facility of the SOMA complete configuration enforces user authentication and the corresponding security policy. Only after authentication, the hosting environment can disclose the associated user profile.

Many other add-on modules can enhance the UVE facility. In the case of a regularly mobile user, the SOMA-based middleware could arrange result delivery in advance. For instance, if the user is interested in the results of the same query, e.g., selected stock updates, the result can be sent where the user works from Mondays to

Wednesdays and to a different location the rest of the week. We are working on several of these components that can enhance the SOMA middleware in other specific application domains, possibly combined with the mobile computing support, such as the autonomous distributed retrieval of heterogeneous museum information [Bellavista, 00d] briefly sketched as current work in Chapter 6.

5.3 Mobile Virtual Terminal

The MVT facility is the component of the SOMA mobile computing profile that supports the migration of mobile devices between different locations, by permitting the mobile terminal to continue execution while preserving the state of the interactions with network resources and services. The protocols cited in Section 5.1.2 are the first steps to support MVT, but they address only network connectivity. The SOMA-based middleware, instead, simplifies the tracing of mobile terminals, the dynamic rebinding of resources/services, the support of out-of-band computations, and the persistency of interaction state.

SOMA identification and naming facilities support the tracing of mobile terminals. As it happens with mobile agents, traceability after migration is obtained via care-of solutions, discovery and directory services. A SOMA agent at a fixed location can act as the care-of entity, working as a forwarder for its mobile device. The SOMA discovery-based naming system is preferred to keep track of a mobile terminal within a network locality. The directory-based naming service makes mobile devices visible to all authorized entities in the global system, but imposes a larger run-time overhead. The SOMA integrated middleware provides service developers with the support for any of these solutions.

In addition, any mobile terminal should continue to access the needed network resources/services independently of its location. The MVT facility can interact with VRM (see Section 5.4) to provide several solutions. MVT can re-qualify terminal references by binding to equivalent resources/services in the new locality; it can maintain references to currently remote resources if re-qualification is either impossible or undesired; it can support the creation of new bindings to previously unknown resources/services.

The MVT implementation in terms of mobile agents is suitable to support out-of-band computations, i.e., non-interactive operations performed while the terminal is disconnected. Before terminal disconnection, MVT can command SOMA agents running on the mobile device to migrate to hosts in the fixed network. Agents can operate asynchronously with the disconnected terminal. When a mobile terminal reconnects at another point of attachment, MVT delivers waiting agents and messages to the mobile device.

In addition, the SOMA migration facility supports the serialization of both agent code and execution state into streams suitable for network transfer and storage persistency. MVT exploits serialization to marshal/unmarshal the terminal session state on stable storage media and to continue the execution from recovered information. This persistency checkpoint can help in critical situations, e.g., a connectivity loss or a power shortage.

MVT also faces the security and interoperability issues to protect resources from the unauthorized usage of malicious mobile agents and to face heterogeneity. In particular, MVT exploits the SOMA security facility (authentication via X.509 certificates, flexible authorization policies, secrecy and integrity via standard cryptographic libraries) to grant security to mobile terminals, and the SOMA interoperability module to integrate mobile terminals with heterogeneous resources/services.

5.4 Virtual Resource Management

The VRM facility of the SOMA profile for mobile computing is in charge of maintaining information about the properties and current location of available resources/services. For instance, in case of terminal mobility, VRM implements the server-side functions to establish dynamic connections between mobile terminals and needed resources, while MVT provides the client-side ones.

The SOMA-based implementation of VRM facilitates the modification and migration of system resources/services at run-time, thus enabling even complex management operations. For instance, an administrator can exploit VRM to favor locality in resource access and to balance the system load by means of dynamic redistribution of components. Moreover, the SOMA mobility middleware can maintain pending bindings to agent-wrapped migrated resources by exploiting the same mechanisms for agent tracing.

In addition, the mobility of resources/services significantly benefits from the flexible naming facility available in the SOMA complete configuration. VRM can employ discovery and directory solutions depending on resource/service requirements. For instance, VRM exploits the discovery service to retrieve a folder for project files of one developer team working locally in a LAN. If the project requires the collaboration of other departments, a system administrator can register the resource also at the directory service, for a wider accessibility by all authorized developers.

Global scenarios force VRM to face resource/service heterogeneity. MA systems generally employ the Java object technology to wrap resources into agents that standardize interfaces and control access; the SOMA framework also simplifies the integration with legacy systems via CORBA, as described in Section 3.2.2. From a different point of view, agent wrapping permits to exploit the same security mechanisms and policies already available for SOMA agents to control, to monitor and to log accesses to resources. The possibility to migrate resources and service components introduces additional security issues: similar problems have already produced solutions in the MA area to ensure integrity and secrecy of mobile agents during both network transmission and hosted execution [Tripathi, 00] [Bellavista, 00c].

5.5 The Performance of the Mobile Computing Profile

We report here the costs of the basic mechanisms of the SOMA mobile computing profile, extensively used to support terminal/user mobility and mobile access to resources. The costs are measured for two platforms, one cluster of 300-MHz PentiumII PCs with Windows NT, and another cluster of 120-MHz Sun SPARCstation5 with Solaris 2.6. Both clusters exploit 10Mbit Ethernet connections. The mobile place has been hosted by a 233-MHz AMD K6 laptop with Windows 95.

Figure 5.2 reports the performance of the mobile computing profile in the two platforms. The terminal mobility cost measures the time for the mobile place to connect to the new hosting domain and to rebind its resources to the local equivalent ones via the SOMA discovery-based naming service. The connection time has a linear dependence from the number of resources to rebind. The user mobility cost consists of a constant threshold due to the time needed by the support to detect user reconnection and to wake up the corresponding profile agent, and of a variable part necessary to migrate the profile agent from the *user home* to the current point of attachment. This last action permits the UVE to configure the local terminal according to the user profile. The variable part linearly depends on the profile agent size. The cost of mobile access to resources measures the migration of database resources of different size, and includes the time for de/registration at the discovery-based naming service at the leaving/entering domain. Apart from the fixed cost paid to update the SOMA naming facility, the results scale linearly with the size of migrating resources.

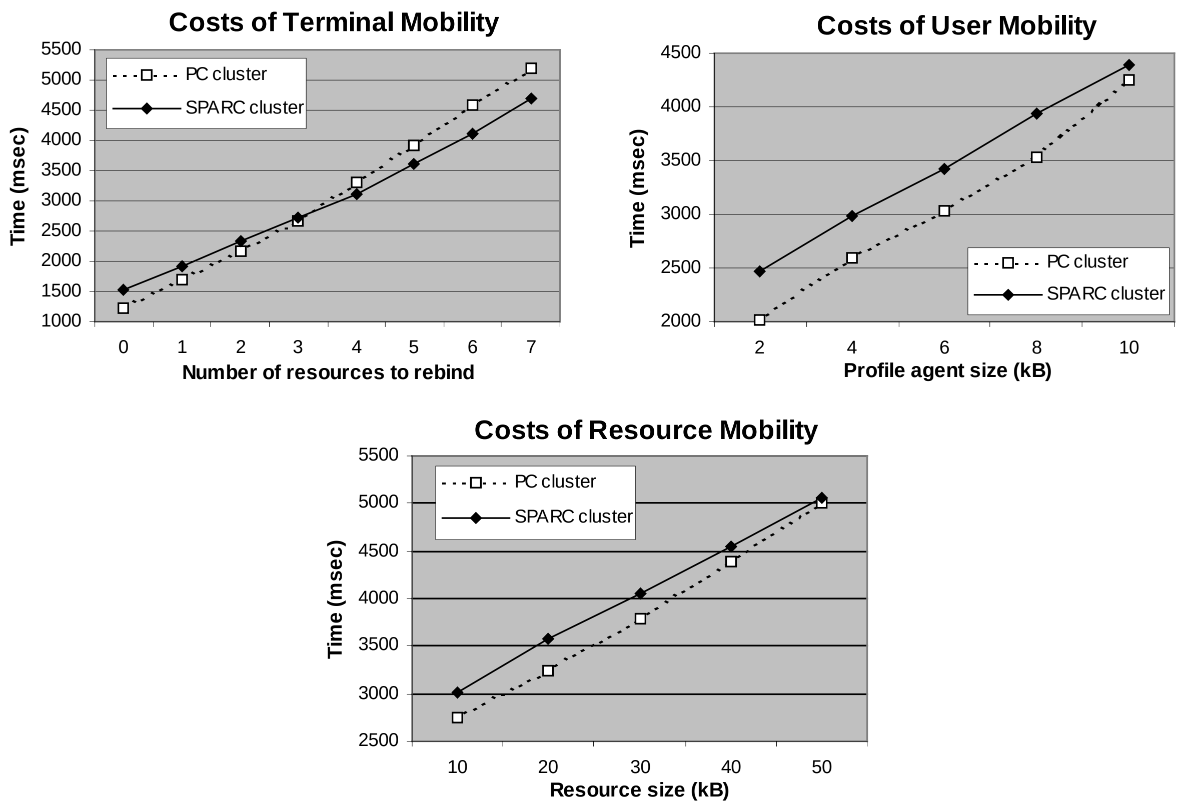


Figure 5.2: Costs of terminal/user mobility and mobile access to resources

All results demonstrate not only the viability of the SOMA approach to mobile computing but also the good scalability in case of entities limited in number and size, as it is typical in ordinary mobility-enabled Internet services. This is mainly due to scalability of the SOMA naming facility for mobile entities that delegates and distributes registration/deregistration duties only in the involved SOMA domains. As a final consideration, we assert that the performance of the different platforms is comparable: the workstations have exhibited larger starting threshold costs, mainly due to their minor processing power, but their performance enhances as soon as the number and size of mobile entities enlarge. This result stems mainly from the optimized mapping of thread and socket mechanisms of the Java Virtual Machine on Solaris platforms.

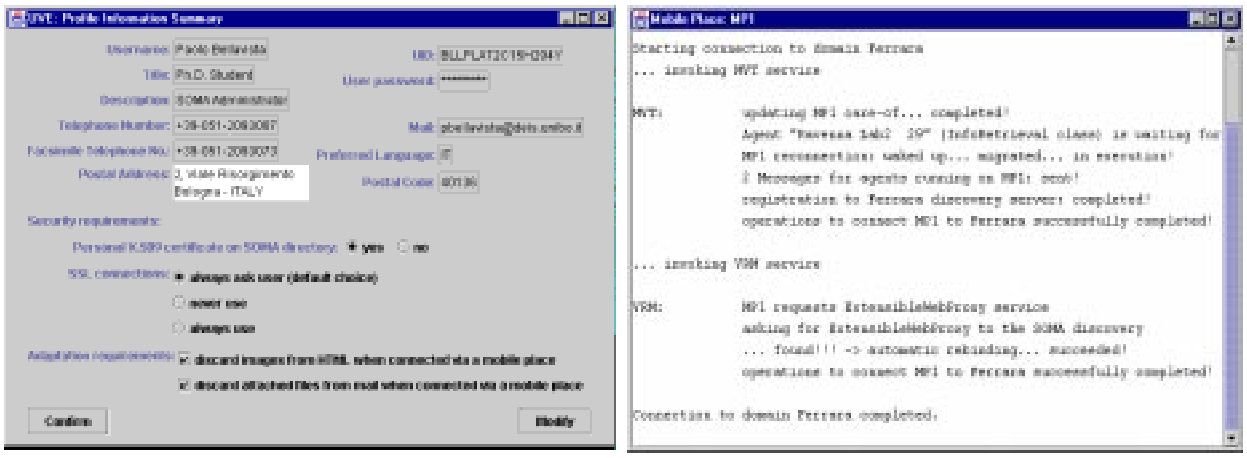
5.6 An Example of Application Scenario

To clarify the functionality available via the SOMA profile for mobile computing and the modalities of interaction of the UVE, MVT and VRM facilities, let us end this chapter with the presentation of a practical usage scenario of the SOMA-based mobility middleware.

Figure 5.3 shows the UVE interface for profile modification. The profile reports personal data, user security requirements, and user suggestions for possible service adaptation via the specialized agents for QoS control available in the SOMA profile for integrated management. For instance, if the user indicates that her X.509 certificate is registered in the directory-based naming facility, the mobility middleware exploits that certificate to perform public-key cryptographic operations; otherwise, the SOMA security facility interrogates the responsible certification authority at run-time.

In addition, the figure presents the console of a mobile place MP1, coming from the *Bologna* domain and requesting to enter the new hosting *Ferrara* domain. MVT updates the care-of at the MP1 home with the information of *Ferrara* current location. In Figure 5.3, one mobile agent is frozen at the MP1 home during MP1 disconnection (and, similarly, two messages sent to agents in execution on MP1). MVT forwards all suspended entities to MP1 in *Ferrara*. The figure also shows the case of a user, working in MP1, with the profile that states her Web access via a specific Web proxy. Before leaving *Bologna*, she owned a binding to the corresponding Web proxy server in the *Bologna* domain. At the reconnection in *Ferrara*, MVT asks VRM to know about the availability of one equivalent Web proxy. The VRM first interrogates the discovery-based naming facility: if a local proxy is available, VRM rebinds MP1 to it, as shown in Figure 5.3. Otherwise, VRM asks the SOMA directory-based naming system: if there were many functionally equivalent Web proxies, MVT would have prompted the user for a choice. Finally, MVT updates the SOMA naming facility local to the *Ferrara* domain to include MP1, which can work as a SOMA fixed place.

The UVE, MVT and VRM modules exploit the available persistency facility of the SOMA basic configuration to freeze/wake-up SOMA agents and messages in case of disconnection/reconnection of users, terminals and networked service components. Persistency minimizes the consumption of system resources while agents wait for a currently disconnected user/terminal/component. In addition, the persistency facility can provide fault-tolerance by duplicating and storing agent copies before beginning critical operations.



UVE : Profile Information Summary

Username:	Paolo Bellavista	UID:	BLLPLA72C15H294Y
Title:	Ph.D. Student	User password:	*********
Description:	SOMA Administrator		
Telephone Number:	+39-051-2093087	Mail:	pbellavista@deis.unibo.it
Facsimile Telephone No.:	+39-051-2093073	Preferred Language:	IT
Postal Address:	2, Viale Risorgimento Bologna - ITALY	Postal Code:	40136

Security requirements:

Personal X.509 certificate on SOMA directory: [✓ yes] no

SSL connections: [✓ always ask user (default choice)] never use always use

Adaptation requirements: [✓ discard images from HTML when connected via a mobile place] [✓ discard attached files from mail when connected via a mobile place]

Mobile Place: MP1

```
Starting connection to domain Ferrara
... invoking MVT service

MVT:        updating MP1 care-of... completed!
            Agent "Vanessa Lab2 29" (InfoRetrieval class) is waiting for
            MP1 reconnection: waked up... migrated... in execution!
            1 Messages for agents running on MP1: sent!
            registration to Ferrara discovery server: completed!
            operations to connect MP1 to Ferrara successfully completed!

... invoking VRM service

VRM:        MP1 requests ExtensibleWebProxy service
            asking for ExtensibleWebProxy to the SOMA discovery
            ... found!!! -> automatic rebinding... succeeded!
            operations to connect MP1 to Ferrara successfully completed!

Connection to domain Ferrara completed.
```

(a) (b)

Figure 5.3: UVE user profile information summary (a); mobile place interactions with the MVT and VRM services at reconnection in the *Ferrara* domain (b)

The SOMA profile for mobile computing is showing its effectiveness in supporting mobility-enabled Internet services [Bellavista, 00b] [Bellavista, 00e]. We are completing the implementation of several service prototypes based on the profile. In the area of network, systems and service management, the Mobile Manager service will permit to system administrators to transparently attach their mobile terminals to locally inspect different parts of the controlled systems.

In the domain of distributed retrieval of heterogeneous information, the Virtual Museum service will take advantage of the possibility of moving database resources to improve service performance and to balance the system load. In addition, the updated results of queries that are regularly of interest of some registered users will be automatically cached within the network localities where those possibly mobile users presumably will try to access them.

6 Concluding Remarks

6.1 *Lessons Learned*

The Internet global communication infrastructure is becoming the privileged medium to provide services to a fast growing market of users, interconnected by possibly mobile heterogeneous devices. The widespread diffusion of Internet services and the consequent increasing commercial competition are pushing and accelerating the demand of users, service providers and network operators for new and complex service properties, from differentiated QoS levels to ubiquitous accessibility, from dynamic personalization of service functionality to user accounting for effective resource consumption.

This scenario requires providing solutions to several technological challenges at different levels of abstraction, and induces a considerable growing of the complexity of the Internet communication infrastructure. The complexity can be faced up only with the provision of rich and flexible general-purpose middleware facilities that give service developers the possibility to use enhanced APIs to support the design, development and deployment of Internet services with the properties mentioned above. In this way, service developers can abstract from the duties of horizontal infrastructure solutions, and can focus their attention and competence only on domain-specific issues. The key relevance of providing middleware facilities for Internet services is widely recognized in the distributed systems community [Sventek, 00] [Schmidt, 00] [Bellavista, 01b].

The doctorate has investigated the design of a middleware for Internet services based on mobile agents. We have adopted the MA technology mainly because of the agent capacity of exploiting locality in the access to distributed resources and of performing distributed operations in an autonomous and asynchronous way with respect to both commanding users and originating hosts. In particular, we have designed and implemented an integrated middleware, called SOMA, organized in different layers of configurations (general-purpose horizontal facilities) and profiles (domain-specific vertical facilities), in order to support flexibly the different and sometimes interrelated requirements of Internet services in different application domains. On the basis of the SOMA middleware, we have implemented several service prototypes, especially in the areas of network and systems management, of multimedia distribution with differentiated QoS levels, and of user/terminal/resource mobility. The prototypes have contributed to the state-of-the-art of the research in the field [Bellavista, 99b] [Bellavista, 00a] [Bellavista, 01c] and have permitted to evaluate the effectiveness of the solutions adopted in the SOMA middleware, thus producing significant feedback for its refinement.

An important lesson learned is that it is feasible to provide modular, flexible and dynamically configurable middleware solutions with acceptable results in terms of both performance and support complexity. To achieve this result, we have shown that it is crucial to adopt suitable methodologies, programming paradigms and technologies, in order to support dynamic extensibility, modularity, reusability and locality in the access to distributed resources. The facilities available on the network nodes involved in the provision of a specific service should be tailored, even at runtime, to user-specific and domain-specific requirements. The middleware cannot be complete and lightweight at the same time. For this reason, we have organized the SOMA middleware in configurations and profiles, in order to give service developers the possibility to choose dynamically the most suitable balance between support completeness and achievable performance.

In addition, in our experience it is fundamental to provide service developers with different abstraction levels depending on the specific application domain addressed because the same level of transparency/awareness is not suitable for all service domains. A flexible middleware should be capable of taking transparent decisions about entity location and QoS management, but should also propagate the visibility of location and QoS properties to the application level when application-specific operations are required, e.g., multimedia flow scaling when the frame transfer rate is minor than a specified threshold.

6.2 Future Work

Given the encouraging results obtained by the implementation of both the SOMA middleware and the Internet services built on top of it, we plan to continue this research activity within the framework of two national projects (the MURST Project of National Relevance “QoS Infrastructures for Web-based Multimedia Services with Heterogeneous Ubiquitous Accessibility” and the MURST Parnaso “Ecumene: Technologies and Tools for Virtual Museums”) that have been just funded.

From the point of view of the SOMA profile for integrated management, we are working on the extension of profile facilities for QoS management in order to grant and guarantee specified levels of quality. We are evaluating and testing solutions that adopt network-layer reservation protocols, such as RSVP, and network-layer technologies providing direct control of QoS parameters, such as ATM. In addition, we are developing and deploying a SOMA-based service prototype for the adaptive scaling of multimedia flows to heterogeneous mobile devices. The service will exploit not only the middleware APIs of the integrated management profile, but also the mobile computing profile facility to manage terminal mobility and device profiling.

We are working also on the design and implementation of a new SOMA profile for the autonomous retrieval of distributed and heterogeneous museum information. The goal is to provide middleware facilities for a virtual museum service that permits to access and manage the data (e.g., pictures, figures, maps, texts, audio, and animated images) about artistic and historical objects maintained over a large number of geographically distributed servers of public and private organizations in Italy. The complexity of the scenario mainly stems from the high heterogeneity of data servers, formats and contents. This is due to the lack of common standards for neither storing the inventory data and the digital representations of the cultural patrimony, nor providing catalogues and indexing structures at the local sites or at the global level. This application domain will permit also to experiment distributed caching solutions based on the SOMA technology. Caching agents will maintain and automatically update the results of query of usual interest directly in the network locality of interested users. In addition, caching agents can migrate to follow mobile users who show recurring and predictable mobility patterns, as mentioned in Section 5.6, thus integrating with the SOMA profile for mobile computing.

We will continue to take into consideration state-of-the-art standard proposals, addressed both to configuration-layer interoperability and to domain-specific profile-layer efforts of standardization. In particular, we are interested in making SOMA fully compliant with the recent FIPA’00 specification [FIPA], and we are investigating the design guidelines emerging from the current OMG work to specify a CORBA service for object migration [OMG, 99b]. About domain-specific interoperability, we are working to provide the MADAMA video-on-demand service with a CORBA interface compliant to OMG multimedia streaming specifications [OMG, 99c]. In addition, we are going to adopt an emerging interoperable representation format for museum information to encapsulate legacy data resources in order to widen their accessibility and enable their integration. This format specifies an XML-based Document Type Definition and is the result of the standardization efforts of the Consortium for the Computer Interchange of Museum Information (CIMI) [CIMI].

Finally, from the point of view of the extension of the SOMA complete configuration, we are working on the integration with policy specification languages to express agent authorizations and duties. This will permit to SOMA system administrators to adopt a suitable high-level approach to the specification of security constraints on resource utilization and of management operations to perform automatically in response to runtime conditions. To this purpose, we are integrating SOMA with a state-of-the-art policy specification language called Ponder [Corradi, 01]. Ponder policies permit to specify both the conditions to control and the actions to perform triggered by system conditions. Local/distributed conditions can be time-independent (e.g., the agent place of origin and its principal) and time-dependent (e.g., the current state of memory usage in a whole SOMA domain), and will be controlled via MAPI-based solutions.

Acknowledgements

The research activities described in the doctorate thesis have been supported by several national research projects:

- MURST Research Project of National Relevance (ex-40%) “MOSAICO: Design Methodologies and Tools of High Performance Systems for Distributed Applications”, 1997-1999;
- MURST Research Project of National Relevance (ex-40%) “QoS Infrastructures for Web-based Multimedia Services with Heterogeneous Ubiquitous Accessibility”, 2000-2002;
- Coordinated CNR Project “Models and Environments for Distributed, Open, Parallel and Mobile Systems”, sub-project “Global Applications in the Internet Environment”, 1997-1999;
- Special Funding of the University of Bologna “Integrated Environments for the Development of Secure Services”, 1999-2001;
- Young Researcher Project of the University of Bologna “A Mobile Agent Middleware for the Support of User, Terminal and Resource Mobility in Global Systems”, 2000.

Many thanks to my supervisor, Maurelio Boari, and to Antonio Corradi and Cesare Stefanelli, for their invaluable advice, direction, encouragement and comments on this dissertation.

A further thanks to all people from academic institutions and industrial research laboratories who have contributed to the design and implementation of the SOMA middleware, and in particular to Thomas Magedanz from IKV++, Stefano Russo from the University of Naples, and Andrea Tomasi from the University of Pisa.

A final thanks to all in the Laboratorio di Informatica Avanzata (LIA) for making this an enjoyable and stimulating period of study.

References

[Aiken, 00]	B. Aiken et al., “A Report of a Workshop on Middleware”, http://www.ietf.org/rfc/rfc2768.txt, Feb. 2000.
[Ajanta]	University of Minnesota, Ajanta Mobile Agents Research Project, http://www.cs.umn.edu/Ajanta/.
[Albitz, 98]	P. Albitz and C. Liu, *DNS and BIND*, 3rd Edition, O'Reilly & Associates, Sep. 1998.
[Al-Shaer, 99]	E. Al-Shaer, H. Abdel-Wahab and K. Maly, “HiFi: a New Monitoring Architecture for Distributed Systems Management”, *19th Int. Conf. on Distributed Computing Systems (ICDCS'99)*, IEEE Computer Society, 1999.
[Anerousis, 99]	N. Anerousis, “An Architecture for Building Scalable, Web-based Management Services”, *Journal of Networks and Systems Management*, Special Issue on Enterprise Management, Vol. 7, No. 1, March 1999.
[Assis-Silva, 98]	F. M. Assis-Silva and R. Popescu-Zeletin, “An Approach for Providing Mobile Agent Fault Tolerance”, in [Rothermel, 98].
[Bakic, 00]	A. Bakic, M. W. Mutka and D. T. Rover, “BRISK: a Portable and Flexible Distributed Instrumentation System”, *Software - Practice and Experience*, Vol. 30, No. 12, 2000.
[Baldi, 98]	M. Baldi and G. P. Picco, “Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications”, *20th Int. Conf. on Software Engineering*, IEEE Computer Society Press, Apr. 1998.
[Bellavista, 00a]	P. Bellavista, A. Corradi and C. Stefanelli, “An Integrated Management Environment for Network Resources and Services”, *IEEE Journal on Selected Areas in Communication*, Special Issue on Recent Advances in Network Management and Operations, Vol. 8, No. 5, May 2000.
[Bellavista, 00b]	P. Bellavista, A. Corradi and C. Stefanelli, “A Mobile Agent Infrastructure for Terminal, User and Resource Mobility”, *IEEE/IFIP Network Operations and Management Symposium (NOMS 2000)*, Hawaii, April 2000.
[Bellavista, 00c]	P. Bellavista, A. Corradi and C. Stefanelli, “Protection and Interoperability for Mobile Agents: A Secure and Open Programming Environment,” *IEICE Transactions on Communications*, IEICE/IEEE

Special Issue on Autonomous Decentralized Systems, Vol. E83-B, No. 5, May 2000.

[Bellavista, 00d] P. Bellavista, A. Corradi and A. Tomasi, "The Mobile Agent Technology to Support and to Access to Museum Information", *2000 ACM Symposium on Applied Computing (SAC 2000)*, ACM Press, Italy, Mar. 2000.

[Bellavista, 00e] P. Bellavista, A. Corradi and C. Stefanelli, "A Mobile Agent Infrastructure for the Mobility Support", *2000 ACM Symposium on Applied Computing (SAC 2000)*, ACM Press, Italy, Mar. 2000.

[Bellavista, 00f] P. Bellavista, A. Corradi and C. Stefanelli, "CORBA Solutions for Interoperability in Mobile Agent Environments", *2nd Int. Symp. on Distributed Objects & Applications (DOA'00*), IEEE Computer Society Press, Belgium, Sep. 2000.

[Bellavista, 00g] P. Bellavista, A. Corradi, R. Montanari and C. Stefanelli, "Security in Programmable Network Infrastructures: the Integration of Network and Application Solutions", *2nd Int. Working Conf. on Active Networks (IWAN 2000)*, Japan, Oct. 2000.

[Bellavista, 00h] P. Bellavista, A. Corradi, R. Montanari and C. Stefanelli, "How a Secure and Open Mobile Agent Framework Suits Electronic Commerce Applications", *Workshop of Italian Ass. Artificial Intelligence (AI*IA) and the Italian Ass. Advanced Technologies based on Object-Oriented Concepts (TABOO)*, Italy, 2000.

[Bellavista, 01a] P. Bellavista, A. Corradi, D. Cotroneo and S. Russo, "Integrating Mobile Agent Infrastructures with CORBA-based Distributed Multimedia Applications", accepted at *9th Euromicro Workshop on Parallel and Distributed Processing (Euro-PDP'01)*, IEEE Computer Society Press, Italy, Feb. 2001.

[Bellavista, 01b] P. Bellavista and T. Magedanz, "Middleware Technologies: CORBA and Mobile Agents", accepted for publication in the book *Coordination for Internet Agents*, Springer-Verlag, Mar. 2001.

[Bellavista, 01c] P. Bellavista, A. Corradi and C. Stefanelli, "Mobile Agent Middleware to Support Mobile Computing", accepted for publication in *IEEE Computer Magazine***,** IEEE Computer Society Press, Mar. 2001.

[Bellavista, 99a] P. Bellavista, C. Cavallari, A. Corradi and C. Stefanelli, "Agenti Mobili per Servizi in Internet: Direzioni di Standardizzazione e loro Implementazione in SOMA", *37th Conf. of the Italian Association for Computer Science and Automatic Computation (AICA'99)*, Italy, Sep. 1999.

[Bellavista, 99b] P. Bellavista, A. Corradi and C. Stefanelli, “An Open Secure Mobile Agent Framework for Systems Management”, *Journal of Network and Systems Management*, Vol. 7, No. 3, Sep. 1999.

[Bellavista, 99c] P. Bellavista, A. Corradi and C. Stefanelli, “A Secure and Open Mobile Agent Programming Environment”, *Int. Symp. on Autonomous Decentralized Systems*, Tokyo, Japan, Mar. 1999.

[Bellavista, 99d] P. Bellavista, A. Corradi, C. Stefanelli and F. Tarantino, “Mobile Agents for Web-based Systems Management”, *Internet Research*, MCB University Press, Vol. 9, No. 5, Nov. 1999.

[Bhagwat, 96] P. Bhagwat, C. Perkins and S. Tripathi, “Network Layer Mobility: an Architecture and Survey,” *IEEE Personal Communications*, Vol. 3, No. 3, June 1996.

[Bieszczad, 98] A. Bieszczad, B. Pagurek and T. White, “Mobile Agents for Network Management”, *IEEE Communications Surveys*, Vol. 1, No. 4, Dec. 1998.

[Bolliger, 98] J. Bolliger and T. Gross, “A Framework-based Approach to the Development of Network-aware Applications,” *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998.

[Bolot, 94] J.-C. Bolot, T. Turtelli and I. Wakeman, “Scalable Feedback Control for Multicast Video Distribution in the Internet”, *Proc. ACM SIGCOMM’94*, Sep. 1994.

[Borenstein, 94] N. S. Borenstein, “E-Mail with a Mind of its Own: the Safe-Tcl Language for Enabled Mail”, *IFIP Transactions C (Communication Systems)*, Vol. C-25, 1994.

[Breugst, 98] M. Breugst, L. Hagen and T. Magedanz, “Impacts of Mobile Agent Technology on Mobile Communications System Evolution”, *IEEE Personal Communications*, Vol. 5, No. 4, Aug. 1998.

[Busse, 96] I. Busse, B. Deffner and H. Schulzrinne, “Dynamic QoS Control of Multimedia Applications Based on RTP”, *Computer Communications*, Vol. 19, No. 1, Jan. 1996.

[Buyya, 00] R. Buyya, “PARMON: a Portable and Scalable Monitoring System for Clusters”, *Software – Practice and Experience*, Vol. 30, No. 7, 2000.

[Cabri, 00] G. Cabri, L. Leonardi and F. Zambonelli, **“**Mobile-agent coordination models for Internet applications**”**, *IEEE Computer*, Vol. 33, No. 2, Feb. 2000.

[Calderon, 98] M. Calderon, M. Sedano, A. Azcorra and C. Alonso, “Active Network Support for Multicast Applications”, *IEEE Network*, Vol. 12, No. 3, May 1998.

[Chalmers, 99] D. Chalmers and M. Sloman, "A Survey of Quality of Service in Mobile Computing Environments", *IEEE Communications Surveys*, Vol. 2, No. 2, 1999.

[Chen, 98] T. M. Chen and A.W. Jackson (eds.), Special Issue on Active and Programmable Networks, *IEEE Network Magazine*, Vol. 12, No. 3, May 1998.

[Chess, 95] D. Chess et al., "Itinerant Agents for Mobile Computing", *IEEE Personal Communications Magazine*, Vol.2, No. 5, May 1995.

[Chess, 98] D. Chess, C. G. Harrison and A. Kershenbaum, "Mobile Agents: Are They a Good Idea?", in G. Vigna (ed.), *Mobile Agents and Security*, LNCS 1419, Springer Verlag, 1998.

[Chung, 98] P. E. Chung et al., "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer", *C++-Report*, Vol. 10, No. 1, Jan. 1998.

[CIMI] Consortium for the Computer Interchange of Museum Information (CIMI), *Dublin Core Metadata*, http://www.cimi.org/.

[CLDC] Sun Microsystems, Inc. - CLDC and the K Virtual Machine (KVM), http://java.sun.com/products/cldc/.

[Concordia] Mitsubishi – Concordia, http://www.meitca.com/HSL/Projects/ Concordia/.

[CORBA/CMIP] UH Communications ApS - The UHC CORBA/CMIP Gateway product, http://www.uhc.dk/.

[Corradi, 01] A. Corradi, N. Dulay, R. Montanari and C. Stefanelli, "Policy-Driven Management of Agent Systems", accepted for publication in *Policy Workshop 2001*, Great Britain, 2001.

[Corradi, 97] A. Corradi and C. Stefanelli, "HOLMES: a Tool for Monitoring Heterogeneous Architectures", *4th Int. Conf. on High Performance Computing*, IEEE Computer Society, Los Alamitos, 1997.

[Corradi, 99] A. Corradi, M. Cremonini, R. Montanari and C. Stefanelli, "Mobile Agents Integrity for Electronic Commerce Applications", Special Issue on Information Systems Support for Electronic Commerce, *Information Systems*, Elsevier, Vol. IS24, No. 6, Nov. 1999.

[DCOM] Microsoft – DCOM, http://www.microsoft.com/com/ tech/DCOM.asp.

[de Meer, 98] H. de Meer, et al., "Tunnel Agents for Enhanced Internet QoS", *IEEE Concurrency Magazine*, Vol. 6, No. 2, Apr. 1998.

[Deri, 00]	L. Deri and S. Suin, “Effective Traffic Measurement Using Ntop”, *IEEE Communications Magazine*, Vol. 38, No. 5, May 2000.
[DIVA]	University of Naples – Distributed Video Architecture (DiVA), http://grid.grid.unina.it/projects/diva/.
[Entrust]	Entrust Technologies - Entrust, http://www.entrust.com/.
[Felber, 98]	P. Felber, R. Guerraoui and A. Schiper, “The Implementation of a CORBA Group Communication Service”, *Theory and Practice of Object Systems*, Vol. 4, No. 2, 1998.
[FIPA]	Foundation for Intelligent Physical Agents – *FIPA’00*, http://www.fipa.org/.
[Foster, 96]	I. Foster, J. Geisler, B. Nickless, W. Smith and S. Tuecke, “Software infrastructure for the I-WAY high-performance distributed computing experiment”, *5th IEEE Int. Symp. on High Performance Distributed Computing*, IEEE Computer Society Press, 1996.
[Fox, 98]	A. Fox, S. D. Gribble, Y. Chawathe and E. A. Brewer, “Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives,” *IEEE Personal Communications*, Vol. 5, No. 5, Oct. 1998.
[Fuggetta, 98]	A. Fuggetta, G. P. Picco and G. Vigna, “Understanding Code Mobility”, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998.
[Gavalas, 00]	D. Gavalas, M. Ghanbari, M. O'Mahony and D. Greenwood, “Enabling Mobile Agent Technology for Intelligent Bulk Management Data Filtering”, *IEEE/IFIP Network Operations and Management Symposium (NOMS 2000),* USA, Apr. 2000.
[Gavalas, 99]	D. Gavalas, et al., “An Infrastructure for Distributed and Dynamic Network Management Based on Mobile Agent Technology”, *IEEE Int. Conf. On Communications*, Vancouver, June 1999.
[Gibbs, 94]	S. J. Gibbs and D. C. Tsichritzis, *Multimedia Programming,* Addison-Wesley, 1994.
[Glitho, 95]	R. H. Glitho and S. Hayes (eds.), Special Issue on Telecommunications Management Network, *IEEE Communications Magazine*, Vol. 33, No. 3, Mar. 1995.
[Glitho, 98]	R. H. Glitho, “Contrasting OSI Systems Management to SNMP and TMN”, *Journal of Network and Systems Management*, Vol. 6, No. 2, June 1998.

[Goldszmidt, 95] G. Goldszmidt and Y. Yemini, "Distributed Management by Delegation", *15th Int. Conf. on Distributed Computing Systems*, Italy, 1995.

[Gong, 97] L. Gong, et al., "Going Beyond the Sandbox", *USENIX Symp. on Internet Technologies and Systems*, USA, Dec. 1997.

[Gordon, 98] R. Gordon, *Essential Java Native Interface,* Prentice Hall, 1998.

[Gosling, 97] J. Gosling and K. Arnold, *The Java Programming Language*, Second Edition, Addison Wesley, Dec. 1997.

[Grasshopper] IKV++ GmbH - Grasshopper, http://www.ikv.de/products/grasshopper/.

[Haggerty, 98] P. Haggerty and K. Seetharaman, "The Benefits of CORBA-Based Network Management", *Communications of the ACM,* Vol. 41, No. 10, Oct. 1998.

[Hansen, 98] J. S. Hansen, T. Reich, B. Andersen and E. Jul, "Dynamic Adaptation of Network Connections in Mobile Environments," *IEEE Internet Computing*, Vol. 2, No. 1, Jan. 1998.

[Howes, 97] T. Howes and M. Smith, *LDAP: Programming Directory - Enabled Applications with Lightweight Directory Access Protocol*, Macmillan Technical Publishing, Jan. 1997.

[Hutchison, 94] D. Hutchison, et al., "QoS Management in Distributed Systems", in *Network and Distributed Systems Management*, M. Sloman (ed.), Addison-Wesley, 1994.

[IAIK] Institute for Applied Information Processing and Communications - IAIK JCE, http://jcewww.iaik.at/jce/jce.htm.

[IEEE P802.11] IEEE P802.11 Wireless Local Area Networks Committee - http://grouper.ieee.org/groups/802/ 11/index.html.

[Inoue, 98] Y. Inoue, D. Guha and H. Berndt, "The TINA Consortium", *IEEE Communications Magazine*, Vol. 36, No. 10, Sep. 1998.

[ISO, 92] ISO/IEC 10165-1, *Information Technology - Open System Interconnection - Structure of Management Information: Management Information Model*, CCITT Recommendation X.720, 1992.

[Jade] CSELT - *Jade*, http://sharon.cselt.it/projects/jade/.

[Jiao, 00] J. Jiao, S. Naqvi, D. Raz and B. Sugla, "Toward Efficient Monitoring", *IEEE Journal on Selected Areas in Communications,* Vol. 18, No. 5, May 2000.

[Jing, 99] J. Jing, A. S. Helal and A. Elmagarmid, “Client-Server Computing in Mobile Environments,” *ACM Computing Surveys*, Vol. 31, No. 2, June 1999.

[JVMPI] Sun Microsystems - Java Virtual Machine Profiler Interface (JVMPI), http://java.sun.com/ products/jdk/1.3/docs/guide/jvmpi/jvmpi.html.

[Karmouch, 98] A. Karmouch (ed.), Special Section on Mobile Agents, *IEEE Communications Magazine*, Vol. 36, No. 7, July 1998.

[Kassler, 99] A. Kassler, H. Christein and P. Schulthess, “A Generic API for Quality of Service Networking Based on Java”, *IEEE Int. Conf. On Communications*, Canada, June 1999.

[Kisielius, 97] V. Kisielius, “Applying Intelligence Makes E-commerce Pay Off”, *Electronic Commerce World*, Vol. 7, No. 12, Dec. 1997.

[Kone, 98] M. T. Kone and T. Nakajima, “An Architecture for a QoS-based Mobile Agent System”, *5th IEEE Int. Conf. on Real-Time Computing Systems and Applications*, 1998.

[Kotz, 00] D. Kotz, and F. Mattern (eds.), *Agent Systems, Mobile Agents, and Applications*, 2nd Int. Symp. Agent Systems and Applications and 4th Int. Symp. Mobile Agents (ASA/MA 2000), Switzerland, 2000.

[Kotz, 97] D. Kotz et al., “Agent TCL: Targeting the Needs of Mobile Computers,” *IEEE Internet Computing*, Vol. 1, No. 4, July 1997.

[Kovacs, 98] E. Kovacs, K. Rohrle and M. Reich, “Integrating Mobile Agents into the Mobile Middleware”, in [Rothermel, 98].

[Kumar, 00] A. Kumar, et al., “Nonintrusive TCP Connection Admission Control for Bandwidth Management of an Internet Access Link”, *IEEE Communications Magazine*, Vol. 38, No. 5, May 2000.

[Kumar, 96] A. Kumar, “Third Generation Personal Communication Systems”, *IEEE Int. Conf. Personal Wireless Communications*, USA, 1996.

[Lange, 92] F. Lange, R. Kroeger and M. Gergeleit, “JEWEL: Design and Implementation of a Distributed Measurement System”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 6, 1992.

[Lange, 98] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, A-W Professional, Aug. 1998.

[Lazar, 98] S. Lazar, I. Weerakoon and D. Sidhu, “A Scalable Location Tracking and Message Delivery Scheme for Mobile Agents”, *IEEE Int. Workshop Enabling Technologies*, USA, 1998.

[Lee, 00] J. Lee, "Enabling Network Management Using Java Technologies", *IEEE Communications*, Vol. 38, No. 1, Jan. 2000.

[Lewis, 98] T. Lewis, "Information Appliances: Gadget Netopia", *IEEE Computer*, Vol. 31, No. 1, Jan. 1998.

[Li, 96] X. Li and M. H. Ammar, "Bandwidth Control for Replicated-Stream Multicast Video Distribution", *IEEE Int. Conf. High Performance Distributed Computing (HPDC)*, Aug. 1996.

[Li, 99] X. Li, M. H. Ammar and S. Paul, "Video Multicast over the Internet", *IEEE Network Magazine*, Vol. 13, No. 2, Mar. 1999.

[Liang, 99] Z. Liang, Y. Sun and C. Wang, "ClusterProbe: An Open, Flexible and Scalable Cluster Monitoring Tool", *IEEE Int. Workshop on Cluster Computing*, 1999.

[Lipperts, 99] S. Lipperts and A. Park, "An Agent-based Middleware: a Solution for Terminal and User Mobility," *Computer Networks*, Vol. 31, Sep. 1999.

[Lupu, 97] E. C. Lupu and M. Sloman, "Towards A Role-based Framework for Distributed Systems Management", *Journal of Network and Systems Management*, Vol. 5, No. 1, Mar. 1997.

[Magedanz, 96] T. Magedanz and R. Popescu-Zeletin (eds.), *Intelligent Networks – Basic Technology, Standards and Evolution*, Int. Thomson Computer Press, London, June 1996.

[Magedanz, 99] T. Magedanz (ed.), Special Issue on Agent Technologies within Intelligent Networks and Mobile Communication Systems, *Computer Networks*, Vol. 31, No. 19, 1999.

[Mazumdar, 96] S. Mazumdar and K. Swanson, "Web Based Management - CORBA/SNMP Gateway Approach", *7th IFIP/IEEE Int. Workshop on Distributed Systems, Operations and Management (DSOM)*, Italy, Oct. 1996.

[McCanne, 96] S. McCanne, "Scalable Compression and Transmission of Internet Multicast Video", Ph.D. thesis, UC Berkeley, 1996.

[Meyers, 00] N. Meyers, "PerfAnal: A Performance Analysis Tool", http://developer. java.sun.com/developer/technicalArticles/GUI/perfanal/, 2000.

[MIDP] Sun Microsystems, Inc. - Mobile Information Device Profile (MIDP), http://java.sun.com/products/midp/.

[Miller, 95] B. P. Miller, et al., "The Paradyn Parallel Performance Measurement Tools", *IEEE Computer*, Vol. 28, No. 11, Nov. 1995.

[Milojicic, 98] D. Milojicic, et al., “MASIF: the OMG Mobile Agent System Interoperability Facility”, in [Rothermel, 98].

[Montanari, 01] R. Montanari, “Security Models for Mobile Agent Systems”, Ph.D. Thesis in Computer Science Engineering, University of Bologna, Feb. 2001.

[Nemeth, 00] E. Nemeth, G. Snyder, T. R. Hein and S. Seebass, *UNIX System Administration Handbook*, Third Edition, Prentice Hall, Sep. 2000.

[NGI] Center for Next Generation Internet (NGI), http://www.ngi.org/.

[Odyssey] General Magic – Odyssey, http://www.genmagic.com/.

[OMG, 98] Object Management Group, *CORBA/IIOP Rev 2.2*, OMG Document formal/98-07-01, http://www.omg.org/library/c2indx.html, Feb. 1998.

[OMG, 99a] Object Management Group, *The Portable Object Adapter*, OMG Document formal 99-07-15, http://www.omg.org/cgi-bin/doc?formal/ 99-07-15, July 1999.

[OMG, 99b] Object Management Group, *Proposal for a Migration Service*, http://www.omg.org/docs/ec/99-01-07, 1999.

[OMG, 99c] Object Management Group, *Control and Management of Audio/Video Streams*, CORBAtelecoms (formal/99-07-12), http://www.omg.org/cgi-bin/doc?formal/99-07-12, 1999.

[Oppliger, 98] R. Oppliger, “Security at the Internet Layer”, *IEEE Computer Magazine*, Vol. 31, No. 9, Sep. 1998.

[Pennington, 00] G. Pennington and R. Watson, JProf - a JVMPI based profiler, http://starship.python.net/crew/garyp/jProf.html, 2000.

[Perdikeas, 99] M. K. Perdikeas, F. G. Chatzipapadopoulos, I. S. Venieris and G. Marino, “Mobile Agent Standards and Available Platforms”, *Computer Networks Journal*, Special Issue on Mobile Agents in Intelligent Networks and Mobile Communication Systems, Vol. 31, No. 10, 1999.

[Perkins, 99] C. Perkins (ed.), Special Section on Autoconfiguration, *IEEE Internet Computing*, Vol. 3, No. 4, July 1999.

[Psounis, 99] K. Psounis, “Active Networks: Applications, Security, Safety, and Architectures,” *IEEE Communications Surveys*, Vol. 1, No. 1, 1999.

[Redlich, 98] J. P. Redlich, M. Suzuki and S. Weinstein, “Distributed Object Technology for Networking”, *IEEE Communications Magazine*, Vol. 36, No. 10, Oct. 1998.

[Rothermel, 98] K. Rothermel and F. Hohl (eds.), *2nd Int. Workshop on Mobile Agents (MA)*, Springer-Verlag, Lecture Notes in Computer Science, Vol. 1477, Sep. 1998.

[Russ, 99] S.H. Russ, et al., “Hector: an Agent-based Architecture for Dynamic Resource Management”, *IEEE Concurrency*, Vol. 7, No. 2, Mar. 1999.

[Schmidt, 00] D. C. Schmidt, V. Kachroo, Y. Krisnamurthy and F. Kuhns, “Developing Next-generation Distributed Applications with QoS-enabled DPE Middleware”, *IEEE Communications*, Vol. 17, No. 10, Oct. 2000.

[Schroeder, 95] B. A. Schroeder, “On-Line Monitoring: a Tutorial”, *IEEE Computer*, Vol. 28, No. 6, June 1995.

[Sessions, 97] R. Sessions, *COM and DCOM: Microsoft’s Vision for Distributed Objects*, John Wiley & Sons, Dec. 1997.

[SOMA] DEIS-LIA - Secure and Open Mobile Agents (SOMA), http://lia.deis.unibo.it/Research/SOMA/.

[Staamann, 98] S. Staamann, et al., “Security in the Telecommunications Information Networking Architecture –the CrySTINA Approach”, *TINA ’97 - Global Convergence of Telecommunications and Distributed Object Computing*, IEEE Computer Society Press, 1998.

[Stallings, 98] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2,* Third Edition, Addison Wesley, 1998.

[Stamos, 90] J. W. Stamos and D. K. Grifford, “Implementing Remote Evaluation”, *IEEE Transactions on Software Engineering*, Vol.16, No.7, July 1990.

[SUN] Sun Microsystems, Inc. - Java 2 Platform, Micro Edition (J2ME), http://java.sun.com/j2me/.

[Sventek, 00] J. Sventek and G. Coulson (eds.), *IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware 2000)*, Lecture Notes in Computer Science, Vol. 1795, Springer, USA, Apr. 2000.

[TACOMA] Tromso and Cornell Moving Agents (TACOMA), http://www.tacoma.cs.uit.no/.

[Tennenhouse, 97] D. L. Tennenhouse, et al., “A Survey of Active Network Research”, *IEEE Communications Magazine*, Vol. 35, No. 1, Jan. 1997.

[Thompson, 98] J. P. Thompson, “Web-Based Enterprise Management Architecture”, *IEEE Communications Magazine*, Vol. 36, No. 3, Mar. 1998.

[Traw, 95] C. B. S. Traw and J. M. Smith, “Striping within the Network Subsystem”, *IEEE Network Magazine*, Vol. 9, No. 4, July 1995.

[Tripathi, 00]	A. Tripathi, T. Ahmed, V. Kakani and S. Jaman, “Distributed Collaborations using Network Mobile Agents”, in [Kotz, 00].
[Tudor, 95]	P. Tudor, “MPEG-2 Video Compression”, *Electronics and Communications Journal*, http://www.bbc.co.uk/rd/pubs/papers/ paper_14/, Dec. 1995.
[UMTS]	European Telecommunications Standards Institute (ETSI), Universal Mobile Telecommunications Systems (UMTS) - http://www.etsi.org/umts/.
[UPnP]	Microsoft Corp., Universal Plug and Play Forum Resources, http://www.upnp.org/re-sources.htm.
[Visibroker]	Borland-Inprise – *VisiBroker 4 for Java*, http://www.inprise.com/ visibroker/.
[Voyager]	ObjectSpace – Voyager, http://www.objectspace.com/.
[W3C CC/PP]	W3 Consortium, Composite Capability/Preference Profiles (CC/PP) – http://www.w3.org/TR/ NOTE-CCPP/.
[Waldbusser, 00]	S. Waldbusser and P. Grillo, “Host Resources MIB”, *RFC 2790*, http://www.ietf.org/rfc/, Mar. 2000.
[WAP]	Wireless Application Protocol (WAP) – http://www.wapforum.org/.
[Weiming, 98]	G. Weiming, G. Eisenhauer, K. Schwan and J. Vetter, “Falcon: On-line Monitoring for Steering Parallel Programs”, *Concurrency - Practice and Experience*, Vol. 10, No. 9, 1998.
[White, 94]	J. E. White, “Telescript Technology: The Foundation for the Electronic Marketplace”, General Magic White Paper, http://www.genmagic.com/, 1994.
[Zhang, 93]	L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, “RSVP: a new resource ReSerVation Protocol”, *IEEE Network*, Vol. 7, No. 5, Sep. 1993.
[Zhang, 98]	T. Zhang, T. Magedanz and S. Covaci, “Mobile Agents vs. Intelligent Agents - Interoperability and Integration Issues”, *4th Int. Symposium on Interworking*, Canada, 1998.