CORBA Solutions for Interoperability in Mobile Agent Environments

Paolo Bellavista, Antonio Corradi
DEIS - University of Bologna
{pbellavista, acorradi}@deis.unibo.it

Cesare Stefanelli
Dip. Ingegneria - University of Ferrara
cstefanelli@ing.unife.it

Abstract

The Mobile Agent (MA) paradigm proposes several attractive solutions to deal with the problems of network-centric programming. Despite the availability of several MA platforms, there are still only a few MA-based distributed services. The paper claims that the lack of interoperability is one of the major obstacles to the large-scale diffusion of the MA paradigm, and discusses solutions to permit the interworking between heterogeneous MA frameworks and other systems, whether MA-based or not, via compliance with either accepted or emerging interoperability standards. In particular, we focus on compliance with CORBA, the accepted standard for OO components, but also with MASIF and FIPA, respectively, the OMG specification for the support of agent mobility and management, and the framework for standard languages and protocols in agent communication. The paper also reports performance results of CORBA-based interoperability in the SOMA programming framework: the presented costs, measured for a systems management application, show the feasibility of the adopted interoperability solutions.

1. Introduction

The widespread popularity of the Web and the ubiquitous availability of access points to the Internet have recently increased the interest in considering the global network as an open distributed system for the design, implementation and deployment of services. These new services are network-centric and compose a scenario where distribution, dynamic modifications, heterogeneity and openness are basic requirements.

The design of network-centric applications has stimulated the research on several new programming paradigms. Some of them (Remote Evaluation, Code On Demand and Mobile Agents [1-4]) move from the consideration that the traditional Client/Server (C/S) model is not flexible enough for the new scenario. They extend the expressive capacity of the C/S paradigm by giving the possibility of transferring code over the network at run-time. In particular, the MA paradigm permits location-aware executing entities to migrate from one network host to another one while in execution (with their code and the reached execution state) [1].

The MA technology is providing attractive solutions in many application areas, such as management of networks, systems and services, mobile computing, electronic commerce, and intelligent information retrieval and filtering [3]. The expected benefits span from realizing robust remote interworking over unreliable networks to increasing the asynchronicity in user interaction, from reducing the traffic over the network to improving performance in information retrieval by exploiting data locality, from balancing load distribution in an administered network to adding dynamically client-specified functionality to servers. Up to now, while there are many different MA systems, there are still only a few MA-based Internet services, and some are mostly limited case studies.

What is currently limiting the MA diffusion in commercial applications is the impression of immaturity around the MA technology. On the one hand, the large number of incompatible MA systems induces a sense of uncertainty. System diversity is important but it can produce incompatibility, discouraging the investments by software companies in MA applications. On the other hand, the MA paradigm certainly raises new security issues in the implementation of distributed applications. Mobile agents are untrusted pieces of code that execute on possibly untrusted hosts over a possibly untrusted network used to migrate and communicate. Many flexible solutions to MA security problems have been investigated and deployed [5]; a still unresolved issue is to find, for any specific application, the most suitable trade-off between contrasting requirements, such as desired security levels and efficiency. We claim that both the necessary degree of interoperability and the proper level of security are key challenges to widen the application opportunities of MA programming frameworks.

Interoperability is greatly simplified by a large acceptance of interoperable interfaces and standard guidelines. In the area of distributed OO systems, the Object Management Group (OMG) has proposed and spread the

Common Object Request Broker Architecture (CORBA) [6]. CORBA provides a stable model to cope with heterogeneity of distributed systems. Applications written according to the CORBA standard are independent of target implementation languages and operating systems. In addition, they rely on brokers to automate object interaction and to support flexible policies for object activation. In the paper, we argue that mobile agents and CORBA, which answer different needs, are not only compatible but also complementary each to the other. We pursue the integration of the two technologies, and, in particular, the achievement of interoperability between heterogeneous MA platforms and between mobile agents and non-MA-based service components via compliance with CORBA-based standards in the MA area.

Our ideas of interoperability and security have led to the implementation of an MA programming framework, called **SOMA**[*] (**S**ecure and **O**pen **M**obile **A**gents), implemented by using the Java 2 Platform, and designed for the development and the support of applications in open and global environments.

From the point of view of interoperability, the SOMA platform provides application designers with facilities to simplify the implementation of agents that can act as both CORBA clients and servers. SOMA is also compliant with the OMG Mobile Agent System Interoperability Facility (MASIF) [7], which standardizes the basic functions of MA frameworks for agent management and transfer to external systems, whether MA-based or not. MASIF is not the only CORBA-based standardization effort in the agent area: from a different perspective, the FIPA (Foundation for Intelligent Physical Agents) [8] proposal mainly focuses on the definition of general standard languages and protocols for communication, coordination and management of heterogeneous agents. We are working to integrate FIPA specifications in SOMA, in order to answer the interoperability issues not covered by MASIF, such as the message exchange between heterogeneous agents.

With regard to security, SOMA protects internal entities by providing an infrastructure designed on standard cryptographic tools for authentication, authorization, secrecy and integrity; in addition, SOMA interacts with external entities by taking into account the CORBA Security Services, as specified in MASIF.

2. Mobile Agents and CORBA

The term agent has many different meanings: in network computing, *mobile agents* are computing entities that can migrate during execution; in the Artificial Intelligence (AI) field, *intelligent agents* are computing elements that embody high capacities of coordination. In the following, we refer to mobile agents as programs that act on behalf of a principal (user or organization) and can autonomously migrate during the execution from one host to another one to continue their operations there. Agents can dynamically choose when and where they will move, and can return results in an asynchronous fashion, thus efficiently facing situations where network connections are unreliable, bandwidth is scarce and even temporarily unavailable. In addition, several agents can cooperate in order to provide distributed and coordinated services.

These features suggest to consider the MA paradigm in several application areas. For instance, systems management applications benefit from MA frameworks, where agents can act on behalf of remote managers to automatically perform management tasks, possibly by moving locally to administered resources. Mobile agents report significant events to the remote central authority without congesting the network with the traffic due to a continuous information exchange between the central manager and the network resources [9] [10].

When dealing with mobile computing, agents can minimize the requested bandwidth to access services on the fixed network, can become the repository of user profile information, and can dynamically distribute along the interested network paths to permit Quality of Service adaptation depending on both terminal capabilities and current availability of resources. In addition, agents can transport service results towards either requesting users or nomadic terminals once they reconnect to different points of attachment to the network [11].

The international committees of the telecommunication area have recognized the importance of structuring systems on the basis of new paradigms and standards. The example of Telecommunications Information Networking Architecture (TINA) suggests the possibility of implementing distributed processing environments (DPEs) that integrate MA-based components [12]. In addition, several projects are going in the area of Intelligent and Active Networks that find solutions based on the movement of active entities capable of dynamically injecting application- and user-specific behavior into the network when needed [13-15].

Mobile agents can offer solutions also in the wide area of electronic commerce in the Internet. The MA paradigm provides asynchronicity between clients and servers, and permits to free users from the need of controlling how their requests proceed. A mobile agent can autonomously roam the network to retrieve the most appropriate information and can simply report it back to the user after performing the needed computation.

CORBA is the key component of the OMG Object Management Architecture (OMA) [6]. Its basic idea is to

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

provide a DPE in which distributed objects can transparently interact according to the C/S model.

CORBA strongly simplifies the realization of C/S distributed applications, by hiding both the implementation and the location of server objects from requesting clients. CORBA not only permits designers of distributed services to abstract from the details related to platform heterogeneity and object distribution, but also allows to integrate already implemented, even non OO-based, software components. These components are wrapped with an Interface Definition Language interface (*legacy system integration*) that correspond to their behavior [16].

The increasing diffusion of CORBA-compliant applications, especially in the area of network and systems management, together with the continuous publication of new Object Services specifications and vendor implementations, create a large installed base of available CORBA components. This contributes to make CORBA an effective solution for fast prototyping complex distributed applications [16].

2.1. CORBA and MA Integration

CORBA and MA technologies are different from several points of view. The most notable one is that CORBA tends to assume that objects are allocated once and for all at a fixed location before their registration at the Object Request Broker (ORB), while mobile agents can dynamically and autonomously migrate during execution depending on time-dependent conditions.

Another relevant difference is MA awareness of current locations and of the locations of needed resources, basic property to permit informed dynamic decisions about migration. At the opposite, CORBA tends to hide the physical location of a server object when answering client invocations. Obviously, the ORB knows the allocation of registered objects, but this information is typically invisible to client objects and application designers. Transparency stems from the CORBA specification and design that simplify the implementation of services in distributed systems, by creating an abstraction of a local concentrated computing environment.

Another difference is in diffusion. CORBA has reached a widely accepted standardization and has a large base of compliant resources, systems and service components. On the contrary, the novelty of the MA programming paradigm has led to propose a great variety of different and non-interoperable MA platforms.

All these considerations suggest that CORBA and MA technologies are not opposite choices but can integrate and complement very well. In fact, a flexible environment for the provision of Internet services can benefit from agent mobility at run-time together with the possibility of transparent remote agent interaction, from the availability of different degrees of visibility of resources in the global system, and from the integration with legacy service components via standard interfaces.

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 MASIF and FIPA proposals adopt CORBA as the standard bridge to overcome heterogeneity.

2.2. OMG MASIF

In our opinion, interoperability among different MA systems is a key issue for widening the diffusion of MA-based commercial applications. Interoperability requires to identify the aspects of the MA technology subject to standardization. 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.

Interoperability also means opening MA systems to new security threats coming from the interaction with external components. The MASIF standard recognizes the need for security and its management: all MASIF implementations are required to introduce security mechanisms, policies and tools, built upon the CORBA Security Services in order to overcome the possible heterogeneity in the security solutions adopted by the interworking components.

2.3. FIPA

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 [8].

3. Interoperability in MA Programming Frameworks

Several MA systems have been proposed and implemented in the last few years [9] [17-21]. They differ in many aspects: the programming language adopted for agent coding (from scripting languages to OO ones), the type of agent mobility (weak/strong) and the way it is implemented (depending on whether they modify the virtual machines they execute on top of), the primitives provided for agent name resolution, communication and control, and the available security features.

Despite the approach variety in designing MA systems, some trends are largely in common. Interpreter-based programming environments represent the common basis for most MA programming frameworks. In particular, Java is frequently chosen not only for portability, fast prototyping and easy integration with the Web, but also for its security features, such as type checking and fine-grained resource access control. In addition, security has attracted deep interests: many MA platforms provide at least a subset of the security services required to implement MA applications in open untrusted environments. Interoperability, instead, has not received yet the due attention, and a definite common trend on how to achieve it and even its meaning is still unclear in the context of mobile agents.

Interoperability is sometimes confused with accessibility that is only a first basic step towards the former: users should have access to MA applications not only from the same MA platform of the application implementation, via system-specific protocols and interfaces, but also from other different systems via widely diffused interfaces such as Web browsers. We consider interoperability a much wider property: it enables full interaction between heterogeneous systems, whether MA-based or not, by permitting cooperation between all existing components in the provision of new and flexible network-centric services.

CORBA helps in answering this requirement. On the one hand, compliance with CORBA allows MA applications to be accessed by any CORBA client, independently of its implementation. On the other hand, the same MA applications can invoke other (even legacy) systems functions via CORBA interfaces. The MA programming framework itself can take advantage of the growing set of CORBA Object Services available from ORB vendors.

A final level of interoperability concerns the possibility of interoperation of heterogeneous MA platforms. At the very least, we need features for agent management and tracking, and for migrating agents among MA systems. In addition, there is the need to exchange information between heterogeneous agents. The compliance with MASIF and FIPA can provide this kind of interoperability.

A few MA frameworks have already approached the issue of interoperability with other systems and applications, whether MA-based or not. Those proposals promote CORBA as the standard integration technology. For instance, *Jumping Beans* [18] implement their agents by providing mobility to CORBA objects. Whenever a Jumping Beans agent decides to migrate, however, it is forced to move to a central server that becomes an exchange point, thus severely limiting the scalability of the system. From the accessibility point of view, *Voyager* [19] permits agents to be activated via an applet interface; it additionally allows access from and to external objects via CORBA. *Aglets* [20] provide an additional package for running an aglet context on an HTTP-browser, and have announced their intention to fully achieve MASIF compliance in the near future. Finally, *Grasshopper* [21] seems to be the commercial MA platform that most com-

pletely addresses interoperability: it permits Web connectivity through external interfaces, allows interaction with non MA-based objects via CORBA, and is currently the only MA programming framework compliant to both MASIF and FIPA.

4. The SOMA Programming Framework

SOMA offers to mobile agents a flexible execution environment built on top of a distributed infrastructure of services. These services include basic agent functions and more complex features necessary to design and develop network-centric applications. In addition, the openness of the SOMA infrastructure allows to dynamically extend the programming framework by adding new agent-based services, even built on the already provided functions.

Figure 1 depicts the SOMA infrastructure that consists of two service layers. The lower layer, which provides the basic functionality for SOMA agents, includes:

- the *identification* service that permits to associate any system entity with a globally unique identifier. SOMA entities include agents, resources, service components and principals, i.e. users/organizations responsible for agent execution;
- the *communication* service that provides tools for coordination and communication between possibly mobile entities. When hosted in the same execution locality, agents interact by means of shared objects, such as blackboards and tuple spaces for tight cooperation. Otherwise, agents can perform coordinated tasks by exchanging asynchronous messages that are delivered to agents also in case of migration;
- the *migration* service that supports the transport of one entity that requests to change its allocation. The reallocated entity should be traced also in the new location by any entity in need of its services, and, if it is active, should transparently restart its execution at the new location;

On the basis of this first level of features, an upper layer of MA-based services is provided in SOMA:

- the *naming* service that organizes globally unique identifiers in naming systems to make possible the tracing of entities even if they move. This service allows to put together a set of different naming systems (DNS-, CORBA-, and LDAP-compliant [22] [6] [23]), possibly characterized by different resolution policies, and is currently implemented by a coordinated set of dedicated agents;
- the *security* service that aims to protect both mobile agents and hosting execution localities. Authentication is based on standard certificates and on a public key infrastructure; authorization extends the Java standard mechanisms for access control; secrecy is achieved by integrating the cryptographic libraries furnished by external providers; integrity has required the development of MA-specific protocols for the protection of mobile agents from the execution environment;
- the *interoperability* service that allows SOMA agents to interwork with existing software and hardware components via compliance with CORBA IIOP specification. In addition, SOMA implements the MASIF interface, thus permitting the interaction of SOMA agents with other MASIF-compliant MA platforms. A more detailed description of the SOMA interoperability service is presented in Section 4.1 and 4.2.

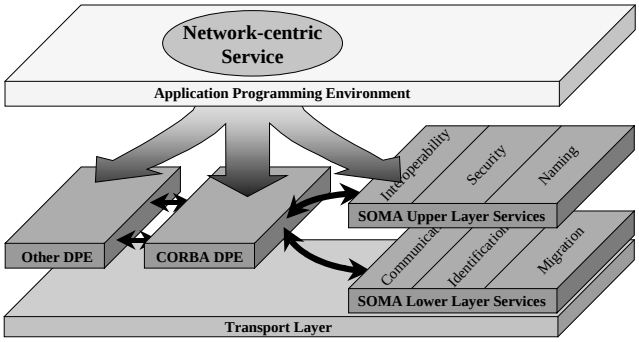

Figure 1. SOMA architecture for the design of network-centric services

In addition to providing this service infrastructure for mobile agents, SOMA offers locality abstractions to describe any kind of interconnected system, ranging from simple Intranet LANs to the Internet (see Figure 2). Any node hosts at least one *place* for agent execution; several places are grouped into *domain* abstractions that correspond to network localities. In each domain, a *default place* is in charge of inter-domain routing functionality and integration with legacy components via CORBA. The *mobile place* is suitable for supporting mobile devices: it enhances the place locality abstraction with specific functions for automatic reconfiguration when changing domain.

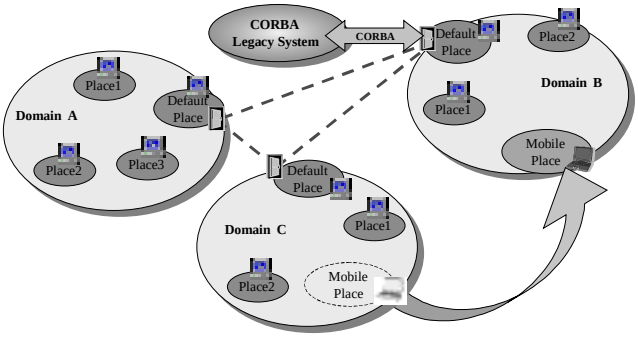

Figure 2. SOMA locality abstractions

Other details about the design and implementation of the SOMA programming framework are presented elsewhere [24] [25] and are out of the scope of this paper that concentrates on the SOMA interoperability features.

4.1. SOMA Interoperability

SOMA faces four different challenges to provide interoperability (see Figure 3):

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).

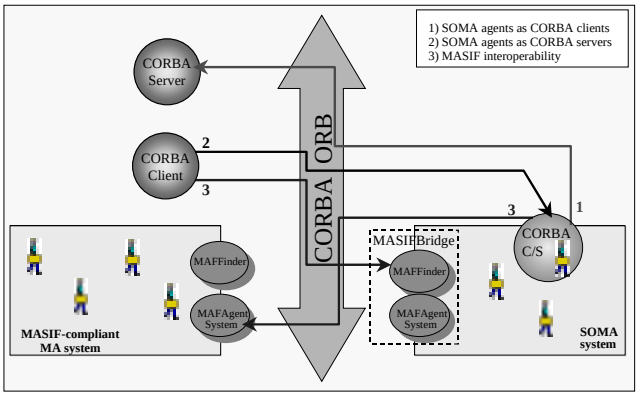

Figure 3. SOMA compliance with CORBA

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 itself 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. SOMA takes into account the security problems that stem from interacting with external components and provides solutions compliant with MASIF security features.

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 [24]. 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.

4.2. SOMA Interoperability Implementation

The SOMA programming framework achieves interoperability by extending its basic functions. In particular, places in charge of interoperating are extended with the *CORBABridge* add-on that is composed by two 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.

Since MASIF implementation increases the code dimension of SOMA places, our default configuration does not extend all places with the *MASIFBridge* module, 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 [27]. 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 [28], introduced to overcome possible incompatibility between different ORB products.

Our *MASIFBridge* module already implements the basic functionality for agent management and naming: the `create_agent()`, `fetch_class()`, `receive_agent()`, `get_MAFFinder()` methods in the `MAFAgentSystem` class, and all the methods in the `MAFFinder` class. We are currently integrating the additional features specified in the MASIF standard to achieve full conformity. We are also testing SOMA interoperability by designing applications in which our agents cooperate with mobile agents from Grasshopper.

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.

5. Designing Applications in SOMA: the *MA_Install* Example

The SOMA infrastructure simplifies the design of complex MA-based applications and their integration with already existing services. First of all, SOMA is implemented in Java and can benefit from all the advantages of object-orientation: inheritance, polymorphism and reuse that help in the rapid development of both prototypes and market-ready applications. In addition, SOMA-based applications can exploit a large set of available services, either provided by the SOMA programming framework itself or accessible via SOMA agents that act as CORBA clients. Finally, the *MASIFBridge* extension makes it possible to perform complex and coordinated tasks in cooperation with mobile agents running in other MASIF-compliant MA frameworks.

SOMA has been used for the design and the implementation of an environment for the management of networks, systems and services called MAMAS (Mobile Agents for the Management of Applications and Systems) [24]. In MAMAS, agents act on behalf of administrators and fulfil administration needs by moving and executing on different nodes. MAMAS makes possible to delegate management actions to mobile agents, simplifying the administrator duty and providing the automation of control actions. Any administrator can implement her policy by using mobile agents, and can propagate new system policies at runtime by agents, with no need to shutdown the system. The MAMAS environment already provides a set of MA-based management functionality, from monitoring the state of a distributed system to controlling and coordinating replicated resources. In addition, it is easy to tailor new agents to new specific administration needs.

As an example of a MAMAS management service, Figure 4 shows a scenario which extensively exploits the interoperability features of the SOMA *CORBA-Bridge* add-on. The *MA_Install* service installs a new program on a constrained subset of the hosts in the system. The service makes use of a number of coordinated agents that dynamically identify target hosts during the installation process on the basis of the current system state: for each administration domain, the program is installed on the hosts whose free space in the local file system satisfies a predefined constraint. In the following, we sketch the main steps of the installation process.

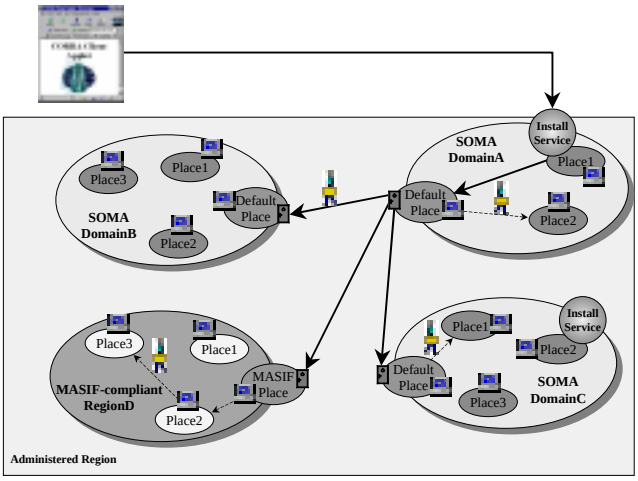

Figure 4. The *MA_install* application

1. The *MA_install* application is registered as a CORBA server object (*CORBA Naming and Trading Services*) and is available to all CORBA clients that satisfy the security requirements of the system (*CORBA Security Service*). For example, a user can employ a standard Web browser interface to invoke *MA_install* by means of a CORBA client applet. The internal implementation of *MA_install*, provided by specialized SOMA agents, is completely hidden from the client.
2. The *MA_install* service is accessible via a stationary agent (SA) that can be present on several places of the administered region (service replication). In the figure, CORBA binds the applet request to the SA instance on `<Place1, DomainA>`. The SA sends a mobile installing agent to the default place in DomainA in order to collect information about other SOMA domains and

about other MA systems that are running in the administered region and registered with the `MAFFinder`. For any SOMA domain, the installing agent duplicates itself and sends its clones there by using SOMA-specific migration mechanisms. For every other MASIF-compliant domain, it controls type compatibility with SOMA, and, in case of compatibility, it sends there its clones. This behavior is implemented with a few lines of code, as shown in Figure 5.

3. All agents autonomously perform their tasks in parallel in each domain, by migrating from one place to another one. They collect information about local disk space; on this basis, they can decide where to install the program and finally command local installation.
4. The client of *MA_install* has the possibility to stop the installation process at any time. The stationary agent can terminate all other *MA_install* agents, by using either the SOMA proprietary features for agent tracking and management or the `MAFAgentSystem.terminate_agent()`method. At the moment we are testing our FIPA ACC implementation while it is used to exchange stop messages between the different agents involved in the *MA_install* service.

We can consider our *MA_Install* application to give an idea of the cost of interoperability, and in particular of the cost for agent migration when using the MASIF interface. The experimental results have been carried out in a SOMA system composed of several domains, and we have measured the cost for migration between default places of different domains. Table 1 compares the average cost of the native migration mechanism of SOMA with the one imposed by the MASIF interface. The results have been obtained in a 10-Mbit Ethernet LAN of 300-MHz PentiumII PCs with Windows NT.

As we expected, the MASIF agent migration is more expensive than the SOMA proprietary one, as reported in the Total columns. In detail, the Total columns represent the cost for moving one agent between two default places, and contain all the overhead associated to establish a connection between previously unconnected places. The initial setup overhead is not paid when considering successive migrations of agents between the same default places, as can be seen in the Successive Migrations columns.

```
public class MA_Install extends SOMA_Agent

{ …
/*  At this point, the MA_Install mobile agent is on    */
/*        <DefaultPlace, DomainA>       */

/*  It obtains a reference to its MAFAgentSystem object */
/* through CORBA Naming Service and CORBA C/S  module
*/
my_MA_System = CORBAClent.narrow(nameService.
        resolve(MAFAgentSystem));

/*  It obtains a reference to its MAFFinder object  */
my_MAFFinder = my_MA_System.get_MAFFinder();

/*  It obtains the list of the agent systems registered with */
/*  my_MAFFinder  (locations) and converts it in places */
/*  enumeration, which uses the SOMA naming  format  */
locations = my_MAFFinder.
        lookup_agent_system(any_name, any_type);
places = CORBAClient.convert(locations);

while (places.hasMoreElements())
  { remote_MA_system=places.nextElement();
    agent_system_type =
      remote_MA_system.get_agent_system_info();

/*  It clones itself and sends its copy to the remote agent system */
/*  by means of SOMA-specific methods:  clone and go */
     if (agent_system_type.is_SOMA_type()) then
        { copy_of_me = this.clone();
          copy_of_me.go(remote_MA_system,
                method_to_start_from)}
/*  It clones itself and sends its copy to the remote agent system  */
/*  by using SOMA clone and  MAFAgentSystem.  */
/*  receive_agent methods */
     else if (agent_system_type.
        SOMA_compatible()) then
        { copy_of_me = this.clone();
                remote_MA_system.receive_
                        agent(copy_of_me,…)
         }

     else { System.out.println("MA System" +
                remote_MA_system.name() + "is not
                SOMA compatible");
            System.out.println("I cannot
                command the installation there!")
          }
    }
…}
```

Figure 5. Agent migration between SOMA and MASIF-compliant agent system

The initial overhead is differently motivated in the MASIF interface and in the proprietary migration mechanism. In the former case, there is a preliminary phase in which the first agent has to obtain the initial reference to the destination `MAFAgentSystem` (phase 1). The first phase includes the invocation of the CORBA Naming Service to solve the name of the source `MAFAgentSystem`, the request for the reference of the source `MAFFinder`, and the final request to the `MAFFinder` for the destination `MAFAgentSystem` reference. Successive migrations can avoid this overhead by maintaining a "stringified" reference to the destination `MAFAgentSystem` (only about 50 ms for name resolution). The phase 2 column of MASIF migration measures the interval between the invocation of `MAFAgentSystem.receive_agent()` and the restarting of agent execution in the destination place. In SOMA proprietary migration, the initial overhead is required in order to establish a dedicated TCP connection between the SOMA default places

of different domains; the overhead due to this connection establishment does not affect successive migrations.

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

Table 1. Time performance for agent migration through MASIF/SOMA transfer methods

Agent Size (kB)	MASIF Migration (ms)				SOMA Migration (ms)	
	Phase 1	Phase 2	**Total**	**Succ. Migr.**	**Total**	**Succ. Migr.**
5	660	633	**1293**	**687**	**699**	**454**
50	645	3215	**3860**	**3267**	**3204**	**2919**

6. Conclusions

The design of network-centric Internet services motivates the adoption of new programming paradigms. The paper integrates two solution directions, mobile agents and CORBA, that are sometimes considered diverging and in contrast.

On the one hand, global network systems force to distribute responsibilities to a plurality of components, able to operate autonomously and in need of coordinating with each other. In this context, MA technology represents a clean and uniform approach to a wide spectrum of application areas, from network and systems management to mobile computing, from distributed information retrieval to electronic commerce. On the other hand, in large corporate and inter-corporate networks, which are inevitably heterogeneous, CORBA is the most widely accepted integration technology to deal with network, platform and programming language diversity. CORBA allows designers to concentrate on high level problems of the application domain instead of worrying about the implementation details connected to distributed heterogeneous environments. In particular, in the MA field, CORBA-based standards such as MASIF and FIPA can play a central role to achieve interoperability by providing standard bridges among proprietary MA implementations.

Our work has focused on the integration of CORBA and mobility, in order to combine the benefits of both C/S and MA paradigms. The implementation of SOMA is an example of how such integration can produce a flexible and secure programming infrastructure, able to interoperate with other systems, whether MA-based or not, and in which building complex distributed applications can be easier than in traditional programming environments. The implementation of SOMA has already shown that the compliance with the CORBA-based MASIF standard can achieve reasonable efficiency. In addition to the use of SOMA agents for network and systems management, we are working on different application domains that stress the interoperability issue, such as the implementation of a SOMA-based middleware to support user, terminal and resource mobility and the development of SOMA-based information retrieval services for distributed and heterogeneous museum data.

Acknowledgements

Investigation supported by the Italian “Consiglio Nazionale delle Ricerche” in the framework of the Project “Global Applications in the Internet Area: Models and Programming Environments” and by the University of Bologna (Funds for Selected Research Topics: "An integrated Infrastructure to support Secure Services").

References

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

[2] J.W. Stamos, and D.K. Gifford, "Remote Evaluation", *ACM Transaction on Programming Languages and Systems*, Vol. 12, No. 4, Oct. 1990.

[3] D.B. Lange, and D. Milojicic (eds.), *1st Int. Symposium on Agent Systems and Applications and 3rd Int. Symposium on Mobile Agents (ASAMA’99)*, IEEE Computer Society Press, Palm Springs, CA, Nov. 1999.

[4] M. Wooldridge, and K. Decker (eds.), Special Section on Agents on the Net, *IEEE Internet Computing*, Vol. 4, No. 2, Mar. 2000.

[5] G. Vigna (ed.), *Mobile Agents and Security*, Lecture Notes in Computer Science, Vol. 1419, Springer-Verlag, 1998.

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

[7] GMD FOKUS, and IBM Corp, *Mobile Agent Facility Specification*, Joint Submission supported by Crystaliz Inc., General Magic Inc., the Open Group, OMG TC Document orbos/98-03-09, ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf, Sep. 1998.

[8] Foundation for Intelligent Physical Agents – *FIPA’99 version 0.2*, http://www.fipa.org/.

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

[10] P. Bellavista, A. Corradi and C. Stefanelli, "An Integrated Management Environment for Network Resources and Services”, *IEEE Journal on Selected Areas in Communi-*

cation, Special Issue on Recent Advances in Network Management and Operations, Vol. 18, No. 5, May 2000.

[11] 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)*, Honolulu, HI, Apr. 2000.

[12] Telecommunications Information Networking Architecture Consortium, *TINA DPE Architecture*, http://www.tinac.com/, Mar. 1998.

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

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

[15] S. Covaci (ed.), *Active Networks*, 1st Int. Working Conference (IWAN'99), Springer Verlag, Lecture Notes in Computer Science, Vol. 1653, Berlin, Germany, June 1999.

[16] K. Seetharaman (ed.), Special Section on CORBA, *ACM Communications*, Vol.41, No.10, Oct. 1998.

[17] N.M. Karnik, and A.R. Tripathi, "Design Issues in Mobile-Agent Programming Systems", *IEEE Concurrency*, Vol. 6, No. 3, July-Sep. 1998.

[18] Ad Astra Engineering Inc. – *Jumping Beans*, http://www.JumpingBeans.com/.

[19] ObjectSpace - *Voyager*, http://www.objectspace.com/.

[20] IBM Japan – *Aglets SDK1.1*, http://www.trl.ibm.co.jp/aglets/.

[21] IKV++ - *Grasshopper 2*, http://www.ikv.de/products/.

[22] P. Albitz, and C. Liu, *DNS and BIND*, 3rd Edition, O'Reilly & Associates, Sep. 1998

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

[24] P. Bellavista, A. Corradi, and C. Stefanelli, "An Open Secure Mobile Agent Framework for Systems Management", *Journal of Network and Systems Management*, Special Issue on Mobile Agent-based Network and Service Management, Vol. 7, No. 3, Sep. 1999.

[25] P. Bellavista, A. Corradi, and C. Stefanelli, "Protection and Interoperability for Mobile Agents: A Secure and Open Programming Environment", *IEICE Transactions on Communications*, Special Issue on Autonomous Decentralized Systems, Vol. E83-B, No. 5, May 2000.

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

[27] Borland-Inprise – *VisiBroker 4 for Java*, http://www.inprise.com/visibroker/.

[28] 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.