Integrating Web Services and Mobile Agent Systems

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

Abstract

Middleware supports based on Mobile Agents (MAs) have recently emerged with the goal of allowing application developers to easily manage and control the properties and behaviors of mobile systems, especially in novel wireless and ubiquitous scenarios. Several MA systems have grown independently with no widely recognized standardization and still lack functions for easy integration. The primary consequence is that one MA platform tends to offer middleware facilities scarcely compatible with other MA systems. Moreover, MA tools are sometimes difficult to employ by common users who have to overcome a significant knowledge gap before properly exploiting MA-specific functions. The paper proposes a solution to open up the usage of MA systems via the bridge of the emerging Web Services (WS) standard technology to achieve interoperability. We have developed an integration infrastructure, called WSMI, with the twofold goal of exporting the MA functions with a WS interface and of simplifying the access to legacy WS components from any MA system. This promotes the uniform access to different MA system functions, independently of the specific MA implementation. We have thoroughly evaluated the WSMI prototype performance, also by comparing the WSMI integration functions with similar non-WS-based facilities: the increased dynamicity and openness of our integration infrastructure have demonstrated to largely counterbalance the overhead introduced by WS adoption.

1 Introduction

In the last few years, the Mobile Agent (MA) [1] approach has emerged as a promising way to address the new challenges in the design and implementation of distributed systems. MAs are software components capable of migrating between different hosts during execution, by carrying their code and the reached execution state. The MA paradigm can help modeling applications whose dynamicity clashes with more traditional interaction solutions: for instance, information retrieval and network management applications had largely and successfully adopted MA-based approaches. Some MA-based support infrastructures have also been proposed to face the requirements of the emerging mobile and ubiquitous scenarios, where computational resources move across different places without losing the capability to interact with other network resources [2-4].

However, state-of-the-art MA platforms still suffer a major drawback: there is no accepted standard for their integration and for providing an easy and simple interface to application developers and final users. While several MA systems have been proposed by different vendors and have become separately available, there is no significant and widely-adopted standard for their interworking, neither among different MA platforms nor with heterogeneous non-MA-based systems. For instance, OMG MASIF and FIPA offer a way mainly to standardize interactions between MA systems and, moreover, have not achieved the foreseen diffusion [5, 6]. So, if an application wants to take advantage of any feature offered by an MA system, it can only interact by following the specific rules and protocols defined by the MA system itself. In summary, MA platform functions are usually available only in a proprietary and hard-to-use way.

Cooperation, integration, and interaction are typical problems to overcome in the collaboration of heterogeneous distributed systems: *Service Oriented Architectures* (*SOAs*) [7] have been proposed as an architecture model to facilitate and promote the cooperation of two or more generic systems. SOA requires any cooperating system to expose interfaces consisting of services and described in a standard common language. The adoption of standard interfaces for cooperation and integration is not new, but in the last years Web Services (WS) have renewed the interest in such strategies by proposing widely-adopted Web

standards to realize SOA models, e.g., HTTP communications and XML-based data description.

While WS couple the SOA interaction approach with the wide diffusion of Web-based protocols, they are more than a promising way of realizing the integration of heterogeneous distributed systems. Moreover, we feel that they can be suitably adopted to address the integration needs of MA systems. The paper proposes a WS-based infrastructure for integrating and exporting MA features called **WSMI** (**W**eb **S**ervices for **M**obile agent systems **I**ntegration). WSMI permits a generic system to take advantage of the main functions of an MA platform and, conversely, MA systems to extend their functions by simplifying the exploitation of WS to access external generic systems. From another perspective, the WSMI infrastructure acts as a bridge linking two different worlds: the typically asynchronous MA scenario and more traditional synchronous SOA environments.

Section 2 describes primary guidelines and functions of WS and MA systems. Section 3 presents the requirements and goals of the WSMI design and implementation, which are respectively described in Sections 4 and 5. The experimental evaluation of WSMI performance, conclusions, and on-going research work end the paper.

2 Models and Functions of Web Services and Mobile Agent Systems

One of the most relevant differences between MA systems and WS is the nature of the interactions between their components. While WS adhere to a typical synchronous paradigm in which a client requests service execution to a server endpoint, MAs interact with clients by following a typical asynchronous behavior: after creation, MAs can migrate between network nodes and interact with their needed resources, with no necessity to maintain continuous connectivity with the associated client.

2.1 Web Services

SOA proposes an architecture for the cooperation of two or more systems via interface-exposed services described in a widely understandable language. SOA – and the deriving WS model – involves three main entities: a provider, a registry (or broker), and a requestor. The first entity hosts service implementation and acts as an access point to the service itself: based on the service description, the requestor sends a service request message to the provider which processes the request, executes service logic, and finally sends back a response message. The registry entity stores service-related naming information, such as service access points or interface definition locations. Interactions between SOA entities typically follow a traditional synchronous request-response model: a requestor sends a service request to a provider that executes the invoked operation and then returns a response to the requestor.

WS realize the SOA model using largely adopted Web protocols, like the HTTP for communication of exchanged data and XML-based grammars for exchanged data format definition (SOAP protocol [8]), service interface description (WSDL protocol [9]), and service registry standardization (UDDI protocol [10]). While all previous protocols are considered mature and largely accepted, WS still have some weak and less explored areas, thus motivating current research efforts: on the one hand, security protocols proposals aim at identifying standards for message authentication and encryption ([11, 12]) and for user identification [13]; on the other hand, WS workflow protocols propose a base for the creation of service aggregations and compositions [14].

2.2 Mobile Agents

MA systems usually consist of a middleware layer deployed over different physical nodes where MAs can migrate and interact with resources. They facilitate application development by enabling the easy movement of application logic, which could be more effective than adopting more traditional programming paradigms in several deployment scenarios. MA middlewares could also help in new wireless environments, where MA asynchronicity can become an advantage in modeling user and device mobility: MA-based solutions are often adopted to dynamically trace, manage, and coordinate mobile users and terminals, even by handling temporary disconnections [2-4]. Nevertheless, we have already stressed that MA-based platforms still exhibit portability and integration problems (see also Section 3).

Nowadays, no globally accepted MA platforms have been proposed and spread. However, the evaluation of most existing MAs, e.g., [15-17], coupled with the investigation of some reference work guidelines [18], helped us gathering the basic MA features into a simple reference model which relies upon the following entities:

- **mobile agents**. Software entities that can migrate across system nodes;
- **places**. System nodes offering computational environments where agents can migrate and interact with owned resources. Usually places map real network nodes;
- **domains**. Groups of homogeneous places, e.g.,

close the one to the other from the physical location point of view. Domains may be structured in hierarchically nested sub-domains.

We assume that places of the same domain know all local places and also all places of nested sub-domains. Coherently with most existing MA systems, our basic model provides the following tools for interaction and communication:

- **blackboards**. Places offer storing entities where senders/receivers (usually MAs and/or users) can deliver messages. Blackboards usually allow asymmetric and non-blocking forms of communication and interaction;
- **messaging tools**. MA systems provide direct communication between senders and recipients, thus enabling symmetric and typically blocking ways of interaction.

Since our goal is to obtain WSMI portability across different MA implementations, the WSMI design will be based on the basic features of the considered reference MA model.

3 Integration Requirements and Goals

The MA integration with heterogeneous systems follows two main directions. The first direction relies upon the proposal of a standard set of functions to access all MA system features, so that traditional applications can take advantage of MA benefits in a dynamic and flexible way. The second direction is to allow MA platforms to extend their functions by interfacing, in a standard uniform way, with external (possibly legacy) systems.

The WSMI solution adopts WS to address both integration directions. First of all, WSMI proposes a set of standard WS interfaces that an external system could invoke to access typical MA system functions (*WS2MA*). In addition, to easily extend MA middleware functions, WSMI defines an integration interface by which MAs can easily invoke also non-MA-based WS components, wherever and whenever available (*MA2WS*). Figure 1 depicts the two directions of integration.

The main requirement followed in the WSMI design and implementation is to achieve maximum interoperability and generality: WS2MA should work in the same way even while changing the underlying MA system and MA systems should be able to invoke external WS (via MA2WS), independently of the nature of the generic external system.

Moreover, an important design goal is to control the costs related to the adoption of WS protocols; these costs are usually non-negligible due to the nature of exchanged data (verbose XML SOAP documents) and of the communication protocol (SOAP over HTTP). Therefore, on the one hand, the WS2MA integration component should not lead to unpredictable cost growth in MA function execution and should represent a good trade-off between costs and benefits. In other words, the WS2MA design should take into special consideration service granularity, by requiring that integration functions provided to open up MA systems must be *coarse-grained*. On the other hand, since WSMI represents a centralized access point to MA system functions, it should preserve system scalability, by avoiding bottlenecks and maintaining acceptable costs, even when the number of service requests grows.

As the WS2MA integration component opens up MA systems to a larger public of potential users, we have carefully considered mechanisms for access control. Further considerations about security issues, e.g., encryption and/or signature of WS requests/responses, are not the focus of the paper; WSMI addresses them by adopting standard WS security-related protocols [11-13]. In addition, to avoid WS2MA users having to guess MA behaviors and functions from mere MA names, we have introduced agent profiling techniques to represent, describe, and dynamically exchange MA semantics.

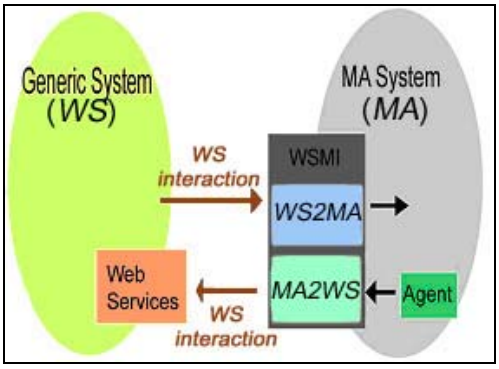

Figure 1. WS2MA and MA2WS directions of integration.

4 WSMI Design

The section discusses the design guidelines for the realization of the WSMI infrastructure that satisfies the requirements sketched in Section 3.

4.1 WSMI properties

The WSMI infrastructure should be independent from the target integration systems, both the MA and the external system counterpart. This is achieved by basing WSMI design on the reference WS and MA models that we have previously identified. The WSMI prototype has been realized for the SOMA MA framework, which is compliant with the reference MA model of Section 2.2 [15].

Service granularity largely depends on the actual service logic (internal actions performed). Therefore, a suitable choice of the WS2MA services to provide requires a trade-off between service costs and results. Service granularity can be controlled by introducing the concept of service-involved topology: users may specify places and domains interested by the execution of a service. We have distinguished a basic set of different levels of service-involved topologies:

- **global topology**. All places defined in an MA system are affected by that service;
- **domain topology**. All places within a specified domain are affected by that service;
- **place topology**. Only a specified place is affected by that service.

For instance, a service reporting a list of all active MAs could involve only one place, by reporting all agents in that specified place, more separate places or the whole MA system topology (in this case the report service provides all currently running agents).

Obviously, granularity control cannot resolve all performance problems: in some MA systems, global topology services could be too heavy for the MA platform itself. Let us stress again that the goal of WSMI is to propose a flexible and rich set of interaction tools for both MA and non-MA systems, by leaving any MA platform the final decision about which tools to use and with which granularity. For example, a particular MA system could decide to reject all global topology services, while others would not.

WS2MA services extend the number of users that can take advantage of MA systems; that introduces a cogent need for access control and user identification policies. While we have no assumptions on how MA systems could specifically achieve these security goals, we feel the need of a common mechanism to realize these policies. That has led to be coherent with the UDDI user identification model and the WS security specifications [10, 13], by proposing user identification and authentication via authentication tokens that users enclose in their service requests as an identity certification. Each user is identified by a couple of IDs (one for the user and another for a possible group she belongs to) and a string containing some authentication information. No assumptions will be made about the nature of this information, as it is important to give MA systems the flexibility to choose their authentication mechanisms/algorithms.

Agent profiles allow to classify agents based on some common semantics. On the one hand, agent profiles can easily express agent behavior that could be difficult to infer from the MA class name; on the other hand, they can also permit to realize some semantic-based correctness controls before service executions. For example, if a Stationary Agent profile is specified to describe MAs that cannot migrate and an MA system tries to expose a WSMI-based service for moving that agent, the WSMI infrastructure automatically determines the mismatch and notifies the possible integration error. We have defined two basic profiles useful for the integration scenarios of our infrastructure: bounded-visibility and synchronous.

The first profile describes agents whose existence, mobility, and activity effects are visible only to specified classes of clients. This kind of MA profile is useful where we do not want to permit unauthorized users to interfere with hidden MA activities. We provide three basic levels of visibility:

- **principal visibility**. MAs are visible only to agent creator (principal);
- **group visibility**. MAs are visible only to members of a specified group;
- **global visibility**. MAs are visible to all users.

MAs are typically asynchronous entities and their activities have not always computational results to be synthesized in a data structure. For instance, consider agents that migrate between places to maintain consistency of data copies: even having a well-defined activity, the execution result of these MAs could be hardly captured and put into a simple data structure. Agents whose activity can produce simple results can be described by synchronous MA profiles and classified accordingly. A successive classification can distinguish the way synchronous agent results are delivered to final users:

- **pull model**. Synchronous agents publish their activity results into blackboards that other users may query;
- **push model**. While there are many ways for MAs to send results to their clients, we have decided that MAs deliver results by invoking an external WS chosen by clients when creating their associated MAs.

For both profiles, and also for additional profiles that will be defined in the future, the WSMI infrastructure can consistently check the service semantics.

4.2 MA2WS

The first MA extension provided by WSMI is the WS invocation feature. The key difference that makes traditional WS invocation inapplicable is the mobility of the endpoint (typically an MA) that requests the services. In fact, an MA could move after a WS invocation and WS response should possibly arrive at a location that is different from the request origin.

Message delivery to mobile entities is a well-known issue in mobile systems and traditionally adopted solution strategies could clash with the WS

interaction model. We propose a request-response interaction that allows agent mobility by defining a proxy-based architecture. WSMI proxies play the role of static WS clients/servers:

- receive WS requests from MAs;
- deliver requests to WS providers;
- receive service responses from WS providers;
- forward responses to the current location of MA requestors, by possibly following different delivery models.

The proxy solution realizes a non-blocking interaction between MA and WS providers and achieves the redirection of service results.

Different types of request delivery channels are provided to better fit MA and/or proxy needs, as shown in Figure 2. The first uses HTTP and performs a typical synchronous blocking interaction: MAs send their requests via SOAP messages and wait until proxies return back receipt confirmations, again via SOAP messages. After confirmation, proxies start operating, by forwarding requests to actual WS providers. The second model, instead, uses the TCP communication protocol and provides a typical asynchronous non-blocking interaction: MAs send requests but proxies do not send any confirmation; MAs are not blocked and proxies can forward requests immediately after their reception.

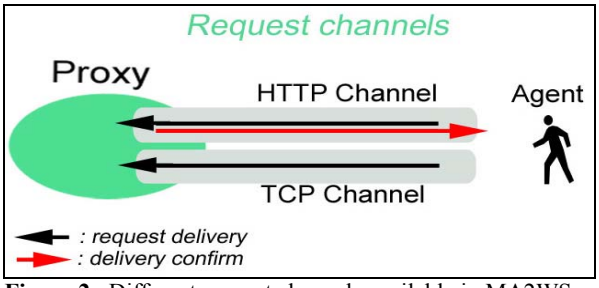

Figure 2. Different request channels available in MA2WS.

Response delivery channels rely upon the different communication tools of the MA reference model. The first way uses a blackboard-based interaction: responses are delivered to blackboards from where agents have to explicitly pull service responses out (pull model). Different flavors are possible: responses can be sent back to the blackboard owned by the place where the MA has first requested the service (standard pull model) or to a blackboard owned by a place specified by the MA itself and where it is going to wait for the response arrival (remote-pull model - MAs expect results from places different from the invocation one). The second way is based upon messaging tools: responses are eventually delivered to MAs as messages. We refer to this way as a push model because it suggests a closer interaction between the proxy and its MA (it is the proxy to push the response to the MA as a directly delivered message).

4.3 WS2MA

WS2MA exposes a set of services via a WS interface to provide external users with a standardized way to access relevant MA system functions.

We have designed two types of WS2MA functions: descriptive services describe runtime MA system properties either in a general (browse services) or in a detailed (drill-down services) way; active services permit to access and invoke MA system actions, eventually by forcing system state changes and interventions in the provisioning environment.

In addition, we distinguish different service targets: either MAs or the MA system topology. We have excluded active topology services because topology modifications to places or domains (such as creations or deletions) would have assumed a control level typical of MA platform administrators. On the contrary, our guideline is that WS2MA users should not heavily modify the MA system itself.

We have also introduced two aggregate services that join the information conveyed by more services into one aggregate: this granularity-driven choice permits to obtain several results by performing only one WSMI service invocation. We propose the following groups of services:

- **browse descriptive MA services**. They include: *listAvailableAgents* that lists all agents available for invocation in the specified topology, while *listAllAgentsOfAuthority* lists all agents visible to token-identified user and hosted by places defined in the specified service topology;
- **drill-down descriptive MA services**. They include services to monitor the current MA position (*getAgentPosition*) and the current MA status (*getAgentStatus*). For synchronous MAs, there is also the *getAgentResult* service that returns the MA activity results;
- **active MA services**. Only one service (*createAgent*) is in this category. It creates an MA instance in the specified place (by starting its execution if specified), passing a user-defined initialization string. It is also possible to specify the type of visibility for bounded visibility MAs and MA semantics for synchronous agents, e.g., the mode exploited to return activity results;
- **browse descriptive topology services**. They comprise two services: *listAllDomains* lists all domains composing the MA system topology, while *listAllPlaces* lists all places in a specified

domain. Only globally recognizable users can invoke the first service, while users invoking the second must be recognizable either globally or at least by the specified domain;

- **drill-down descriptive topology services**. They include the *getPlaceDescription* and *getDomainDescription* services that describe topology entities defined in the MA system;
- **aggregate services**. Two service aggregates are defined. The first (*getAgentDescription*) returns information about the current status, position, and activity results of a specified MA, while the second (*getTopologyDescription*) returns high-level (browse services) topology information and detailed (drill-down services) topology information. These services perform a breadth-first access to domain topology trees to obtain detailed descriptions about domains/places.

5 WSMI Implementation Guidelines

Here we point out some primary guidelines of the WSMI implementation. Additional design and implementation details, together with the full source code of the WSMI prototype, are available at `http://lia.deis.unibo.it/Research/SOMA/WSMI/`. The current implementation integrates with the SOMA platform, entirely developed in Java; consequently, we have chosen to implement all WSMI components by using the same language. However, the WSMI development has carefully considered openness and portability as primary design issues; WSMI can easily be used not only in conjunction with the SOMA platform, but also with any MA system that follows the presented MA reference model.

5.1 WSMI Properties

As most MA systems, SOMA does not provide a middleware functionality for MA profiling. WSMI extends SOMA with the MA profiling feature by exploiting Java interfaces: SOMA MAs are Java objects and MA profiles can be modeled as interfaces that MAs should implement to be compatible with those profiles. WSMI defines the following interfaces:

- **BoundedVisibilityAgent**. It is the bounded visibility MA profile with two methods to set and get agent visibility;
- **SynchronousAgent**. It is the synchronous MA profile with a method to get agent activity results as *java.io.Serializable* objects. We have implemented this interface by inheriting from *BoundedVisibilityAgent* because any MA created via the WS2MA interface should be visibility-bounded;
- **SynchronousPullAgent**. It describes the synchronous MA profile with pull-based result delivery, by extending *SynchronousAgent*;
- **SynchronousPushAgent**. It is the synchronous MA profile with push-based result delivery. It extends *SynchronousAgent* with methods to set and get the service used for result notification; notification services should be described by using the *BusinessService* UDDI data structure.

5.2 MA2WS and WS2MA Implementation

To work properly and effectively, WSMI proxies need additional MA system information: for instance, they should know the chosen response delivery model. To this purpose, MAs can generate SOAP documents to describe the characteristics of service requests by including in the document headers several operational parameters: the MA ID, the response delivery model, a key value to retrieve the generated result in a blackboard (in pull-mode interactions), the result location for generic pull models dependent on the specific MA system, and the service provider location (usually a URL).

We have realized two different proxies that implement both presented request delivery models, thus providing differentiated channels. In particular, HTTP-based proxies have been implemented as Java servlet components by extending the *javax.servlet.http.HttpServlet* class and by redefining the *doPost()* method. TCP-based proxy uses *java.net.ServerSocket* objects waiting for SOAP envelope sending.

Blackboards are data structures, local to any SOMA place, with simple storing and retrieving functions used to write/read service results. The push model uses typical MA communication tools: in SOMA, any MA has an associated mailbox for message delivery. The only action that proxies must perform to return a result according to the push model is to send a SOMA message to the target MA, which is identified by a SOMA ID independently of the current location of the target MA.

To grant our implementation the required scalability, we have implemented the WS2MA support components by using Java *Session Beans* whose methods map all previously described services. Each MA-related service first checks the compatibility of the invocation with the associated MA profile.

6 WSMI Experimental Evaluation

To evaluate the impact of WSMI adoption, we have performed extensive tests by simulating a typi-

cal case study of interaction between MAs and WS. Namely, a non-SOMA external client first invokes some broad-descriptive WS2MA service on the target system topology (either via the aggregate service *getTopologyDescription* or with *listAllDomains* and *listAllPlaces* services), to get a description of the MA system structure. Then, the client retrieves a list of MAs that can be created on such topology via a *listAvailableAgents* invocation. After the instantiation of a suitable SOMA agent (*createAgent* service), the client tracks the MA activity (position and status) by using the related WS services. SOMA agents continue to perform their activities by exploiting proprietary mechanisms for migration between SOMA places, for local/remote resource interaction, and also by invoking external WS components via the WSMI MA2WS functionality. Finally, if MAs are synchronous, clients can either invoke the *getAgentResult* service to obtain the MA computed response (according to the pull result delivery model) or wait an explicit communication from the responsible MA (according to the push result delivery model).

The respect of a primary design and implementation guideline, i.e., cost reduction and effectiveness, can be better evaluated when considering a worst-case configuration. Therefore, we have chosen a testbed consisting of resource-constrained nodes (Mobile AthlonXP 1,67GHz, 256 MB RAM, WindowsXP Home Edition; and AthlonXP 1,81GHz, 256 MB RAM, WindowsXP Professional Edition) interconnected by a 100 Mbps Ethernet LAN. Enterprise Java Beans and servlets execute in the *SUN J2EE Application Server*, while WS providers run in a single *Apache SOAP* server.

We have performed additional tests with different hardware and software configurations: these further experimental results confirm the trend of the performance indicators reported in the following. Detailed information about the additional tests accomplished are available at `http://lia.deis.unibo.it/Research/SOMA/WSMI/`.

6.1 MA2WS Performance

The MA2WS tests have involved SOMA MAs invoking simple WS components hosted at the Apache SOAP server. We have measured the average execution duration of WS components on a set of one hundred runs. Both proxies have been tested in this way. The measured difference in execution times between the two categories of proxies (about 8s) were expected: the lack of request confirmation when using TCP-based interactions produces significantly faster executions. In any case, even reliable HTTP-based interactions still have reasonable execution times, definitely compatible with the application domains typically addressed by WS service provisioning.

Average WS execution times exhibit a non-negligible threshold due to WS-based proxy-to-server invocation (approximately 3s for TCP-based proxies and up to 10s for HTTP-based proxies). Apart from that almost fixed threshold, response times have demonstrated to linearly depend only on server-side execution time.

6.2 WS2MA Performance

Table 1 reports average execution times for WS2MA services exploited in the case study scenario. We reported two different kinds of values: the time interval for SOAP-based interaction and the same time interval when directly accessing the methods of the involved Java Bean. This comparison permits to evaluate the impact of WS-related technologies: the method invocation times column reports the actual costs related to the complexity of the service logic, while SOAP-based invocation includes the overhead of the SOAP protocol and of the WS management infrastructure.

Service	SOAP-based Invocation (ms)	Socket-based Invocation (ms)
listAllPlaces	9307	4160
listAvailableAgents	8036	3458
createAgent	8265	4307
getAgentPosition	8468	3703
getAgentResult	8156	3396

Table 1. Average execution times for WSMI services in the case study scenario.

The most important experimental result is that WS adoption in WSMI does not lead to unexpected and unpredictable fluctuations of service execution times. On the contrary, the impact of SOAP-related technologies adds a highly predictable overhead, which is relatively limited and almost constant for all services. Additional tests (see also the WSMI Web site) have pointed out that the execution times of WSMI services that involve MA system topologies exhibit a well-scalable linear dependence on the number of interested places.

7 Conclusions

We have investigated the possibility of integrating MA systems and (possibly legacy) WS components. The proposed WSMI infrastructure enables application developers to easily exploit standardized synchronous access to functions that are typical of totally asynchronous systems, such as MA platforms. More-

over, MAs themselves can extend their applicability by exploiting typically synchronous modalities of interaction without losing the asynchronicity intrinsic to their programming model: MAs can invoke WS components independently of their location and of their movements between different places/domains during service execution. In novel ubiquitous mobile Internet scenarios for service provisioning, WSMI grants interoperability and heterogeneity for all MA and WS systems that adhere to the adopted reference models. We stress again that very minimal assumptions have been imposed for the sake of portability. The experimental performance evaluation has shown that the WS-MA integration can impose limited and predictable overhead, without introducing unexpected costs and bottlenecks when coupled with a flexible proxy-based support architecture.

The encouraging results obtained with the current WSMI prototype are pushing further research activities. We are currently working on extending WSMI first to address the needs of specific application scenarios, e.g., information retrieval, and to build actual industrial-case applications working on top of the WSMI middleware support. Secondly, we are extending the possibility to define flexible service aggregates, so to personalize the granularity of aggregation depending on the application-specific interaction needed between clients and their delegate MAs.

Acknowledgements

Work supported by the MIUR FIRB WEB-MINDS and by the CNR Strategic IS-MANET Projects.

References

[1] N. M. Karnik, A. R. Tripathi, “Design Issues in Mobile Agent Programming Systems”, *IEEE Concurrency*, Vol. 6, No. 3, pp. 52-61, July-Sep. 1998.

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

[3] B. Emako, R. H. Glitho, S. Pierre, “A Mobile Agent-Based Advanced Service Architecture for Wireless Internet Telephony: Design, Implementation, and Evaluation”, *IEEE Transactions On Computer*, Vol. 52, No. 6, pp. 690-705, June 2003.

[4] A. Di Stefano, C. Santoro, ”NetChaser: Agent Support for Personal Mobility”, *IEEE Internet Computing*, Vol. 4, No. 2, pp. 74-79, Mar.-Apr. 2000.

[5] Object Management Group, “Mobile Agent Facility Specification”, http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm.

[6] Foundation for Intelligent Physical Agents, “FIPA 97 Specification”, http://www.fipa.org/specs/fipa00019/.

[7] W3 Consortium, “Web Services Architecture”, W3C Working Group Note, http://www.w3.org/TR/ws-arch/.

[8] W3 Consortium, “SOAP Version 1.2”, W3C Recommendation, http://www.w3.org/TR/soap/.

[9] W3 Consortium, “Web Services Description Language (WSDL) 1.1”, W3C Note, http://www.w3.org/TR/wsdl.

[10] OASIS, “UDDI Version 2.04 API Specification”, UDDI Committee Specification, http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.

[11] W3 Consortium, “XML–Signature Syntax and Processing”, W3C Recommendation, http://www.w3.org/TR/xmldsig-core/.

[12] W3 Consortium, “XML Encryption Syntax and Processing”, W3C Recommendation, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[13] OASIS, “Web Services Security”, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.

[14] OASIS, “Business Process Execution Language for Web Services - Version 1.1”, http://www-128.ibm.com/developerworks/library/ws-bpel/.

[15] SOMA: Secure and Open Mobile Agent, http://lia.deis.unibo.it/Research/SOMA/.

[16] IKV++, “Grasshopper: an Intelligent Mobile Agent Platform", http://www.ikv.de/content/Grasshopper/Grasshopper.htm.

[17] G. Glass, “ObjectSpace Voyager Core package Technical Overview”, in *Mobility: Process, Computers and Agents*, Addison-Wesley, pp. 611-627, 1999.

[18] A. Fuggetta, G.P. Picco, G. Vigna, “Understanding Code Mobility”, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 342-361, May 1998.