Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 313–324. http://www.scpe.org
ISSN 1895-1767

A MOBILE AGENTS BASED INFRASTRUCTURE TO DELIVER VALUE ADDED SERVICES IN SOAS

ROCCO AVERSA*, BENIAMINO DI MARTINO* , SALVATORE VENTICINQUE* , AND NICOLA MAZZOCCA†

Abstract. The integration of Web Services and Software Agents brings new opportunities to deliver value added services in SOAs. This paper describes the functional model, the architecture design and the prototypal implementation of a platform for services adaptation and delivery. Agents technology is exploited to reconfigure the server platform to adapt the service provision to heterogeneous and handled devices. Virtual services are built on the platform for each client profile by conditioning the SOAP requests coming from WS requestors and adapting the returned responses. A general straightforward approach to extend client application in order to exploit platform facilities is described. Mobile agents are able to access and compose services accessing them by standard interfaces. User authentication, service discovery and publication, personalization are other provided facilities. Technological interoperability is provided by Web Services technology.

1. Introduction. The increase in size and performance of computer networks is both cause and effect of their own ubiquity and pervasiveness. This means that the user is able to exploit network connectivity without expensive add-on and wherever he is. Another relevant phenomenon is the increasing availability of handled mobile devices and their increasing capability through which the users are able to access Internet. In such a context a client could be able to discover countless resources without to be able to exploit most of them. In fact in order to grant the service usability it needs to care about functionalities, technological characteristics and limitations of client devices (mobile phone, palm, notebook). Further we have to consider the possibility that an user accesses the system at different times with various devices and he requires the service by a first connection mode and later collects the results by another one. A very interesting approach to design and develop a platform that provides access to distributed services by heterogeneous handled devices come from the adoption of "Mobile Agents". "Mobile Agents" have to be intended as both a technology and a programming paradigm. A mobile agent is a program able to migrate across the network together with its own code and execution state. We have experience in development of mobile agent platforms for Service Oriented Applications (SOA) [1, 2, 3, 4]. We describe here the design and the implementation of a reconfigurable Web Service architecture (WS for short) based on mobile agents technology. It has been developed upon the MAgDA toolset [5], a Mobile Agents based middleware for distributed and Grid programming. The Mobile Agent paradigm has been exploited both for services management and for service provision. In the first case agents implement the system infrastructure by providing efficient service discovery and composition, user authentication and authorization, session management and other middleware facilities. In the second case agents implements a service themselves which are able to be invoked by a Web Services Interface, further they can migrate form a host to another to optimize the system performance and utilization at server side, or to reconfigure the user device at client side. We present here the functional model and the component based architecture of the platform. The prototypal implementation of the platform was developed as a proof of concept of research activities concerning the CRDC project[1]. It was successively extended in collaboration with the Schumblerger-Sema as part of the activities of the Serviceware European project to investigate the design and development issues of a platform for providing Value Added Services to heterogeneous mobile devices. Hence this contribution has both a methodological and an experimental approach, aiming at the definition of new techniques and the application of new technological solutions to realize reconfigurable WS platforms based on the exploitation of mobile agents. Themes of interest we deal with are: mobile agent based programming paradigms for service provision and composition; service discovery and personalization; session management and system reconfiguration; design of open and interoperable systems; mobile computing and heterogeneous handled devices. In the second section we introduce the state of art of techniques and technologies involved in our activities. In the third and in the fourth sections we present the functional model and the architecture of the developed platform. In the fifth section the interaction protocol between client and middleware is described. Finally an overview of the functionalities provided by the available prototype is shown.

*Dipartimento di Ingegneria dell' Informazione, Second University of Naples, via Roma 29, Aversa, Italy {rocco.aversa, beniamino.dimartino}@unina2.it, salvatore.venticinque@unina2.it
†Dipartimento di Informatica e Sistemisitca, University Federico II of Naples, via Claudio 21, Napoli, Italy, n.mazzocca@unina.it
[1]Regional Competence Center on Information and Communication Technologies funded by Campania Reg. government

2. State of Art. QoS management is critical for distributed SOAs because of clients with different QoS requirements and also service providers with different resources, workloads and ever-changing business rules [6]. Concepts and guidelines from ISO/IEC QoS Framework [7] form the basis for the design and implementation of QoS management in SOAs. Recently, new ideas have been proposed to extend the functionality of UDDI registry by introducing additional features such as UX [8] and UDDIe [9]. Here, a QoS model (with QoS certifiers used to verify the claims of web service's QoS) is presented. [10] provides a generalized approach for building a QoS-based registry for SOA. This framework enhances the current UDDI standard to improve the efficiency and the quality of the discovery process, by taking in account user specified QoS parameters. Others propose a new infrastructure for QoS management such as WSLA [11] or SLAng [12] where the authors address the dynamic selection of services via an agent framework coupled with a QoS ontology. [13] proposes to exploit correlations between QoS properties of each Web service to provide good candidates of Web services for the matchmaking process when service brokering is performed. [14] proposes a quality of service assessment model that is able to capture the reason behind the ratings. This allows us to perform quality of service assessment in context, by using only the ratings that have similar expectations. [15] presents an integrated QoS management architecture and solution for the publish/subscribe style of enterprise SOA.

Our solution implements the QoS management architecture at the middleware layer as a service for Information Brokers. Major contributions include a general integrated QoS management architecture, a flexible and extensible XML-based QoS language, innovative resource models and exploitation of mobile agents technology.

A mobile agent is a program able to migrate across the network together with its own code and its execution state. This paradigm allows both a `pull` and a `push` execution model [16], in fact the user can choose to download an agent or to move it to another host. The agents are able to perform the computation where resources have been allocated in order to optimize the execution time by reducing the network traffic and the interactions with remote systems. Two forms of mobility characterize the migration of agents:

- `Strong mobility` is the ability of an Mobile Code System (called strong MCS) to allow migration of both the code and the full execution state of an agent (very close to process migration).
- `Weak mobility` is the ability of a MCS (called weak MCS) to allow code transfer across different Computing Environments (CEs) together with the intermediate values of those data which are relevant to resume the status of the application: the activation stack and complete image of the process could not be migrated, the application is resumed, not the process.

We will intend the second one thorough the paper. Web services [17] are an emerging distributed computing paradigm which focus on Internet-based standards technology such as XML and HTTP. SOAP, WSDL and Ws-Inspection. They have been designed to support interoperability, i. e. independence of the transport protocols, programming language, programming model and system software. For our purpose the web services support the dynamic registering and discovery of services in heterogeneous environments. WS-Inspection [18] defines a simple XML language and related conventions for locating service descriptions published by a server provider. The service is usually a URL to a WSDL document. WSDL [19] is an XML document for describing Web services. It can be referenced in a Universal Description, Discovery and Integration (UDDI) register [20]. Integrating Web services and software agents brings new opportunities and helps in defining new services. Once this integration is realized, software agent concepts and technologies will help to enable new and advanced operational and usage modalities of Web services. Several publications have been published to support this thesis, including [21, 22]. The Web Services Architecture Specification of the W3C [23] foresees itself the integration of agent technology in Service Oriented Architectures, "... software agents are the running programs that drive Web services - both to implement them and to access them as computational resources that act on behalf of a person or organization". Some interesting results can be found in Lyell et al. [22], which discuss the concept of a hybrid J2EE/FIPA-compliant agent system, in [24, 25], where the authors have proposed adoption of agents for service integration, or in [26, 27] which uses mobile agents and web services for management of virtual home environments. Especially agent migration techniques, allow to dynamically find the "best" node where to execute the service at server side [25, 28, 29]; to upload/download the embedded client application to reconfigure the user device [30].

3. The functional model. The model defined in the CRDC project is shown in figure 3.1. It is composed of the three blocks listed here:

1. A client device is characterized by its technology, the amount of memory, the computing power, the supported communication protocols, the connection bandwidth;

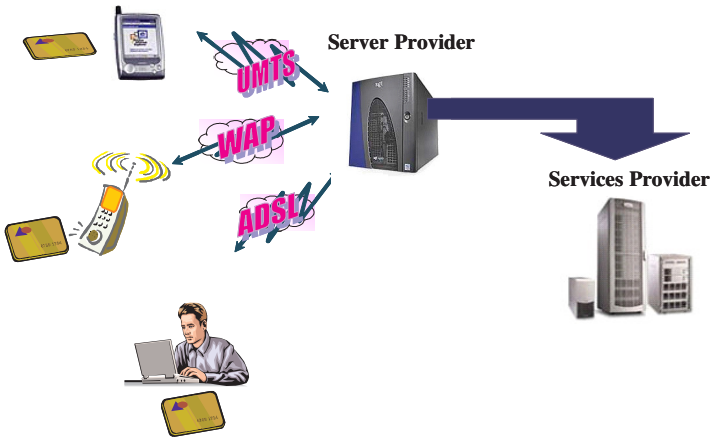

FIG. 3.1. *The CRDC model*

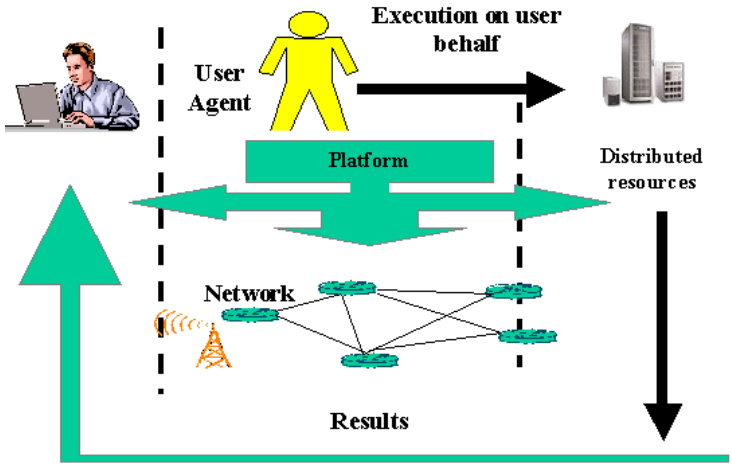

FIG. 3.2. *The Agent based approach*

2. A Services Provider offers the end-user services providing different implementations for different kind of client profiles (user, device, session profile);
3. A Server Provider represents the platform that supports the interaction between the client and the services provider, overcoming the challenges which could arise in service exploitation.

The main challenge in the exploitation of web services through mobile devices is the overcoming of their hardware and software limitations. Dynamic discovery, late binding and services composition cannot be supported by a wide class of smart devices. It motivate developers to experiment new methodology of exploiting remote resources to extend the capability of the client devices. Here we use the Mobile Code technology for local device reconfiguration and for exploitation of the back-end nodes in order to overcome the client limitations. When the execution of the agent is supported by the client device agent migration is exploited to reconfigure the terminal and to allow the user to play the service. When the device does not support the full execution of the client application some activities must be performed by remote resources. The agent executes on the server and provides the access to the service by simpler interfaces. As it is shown in Figure 3.2 the user interacts with a personal agent that is able to migrate on the server and to perform the required actions on behalf of its owner. The proactive characterization of the agent allows the autonomous completion of a task according to a delegation based model that provides the agent with the authorization profile of this owner. In fact in order to represent a human user, the delegate must be able to characterize the context of the interaction. The user context can be composed of actual temporal and spatial location, terminal device profile (configuration and

technology), user's preferences, nearby users, nearby resources, noise level, network connectivity, communication cost, communication bandwidth, and social situation. Some of this contextual information can be sensed through sensors like GPS, and network bandwidth detection, while others can be inferred from knowledge bases. Each of these contextual parameters is fairly complex. For example, terminal device characteristics could involve audio capability, video capability, image decoder, text capability, local storage capacity, screen size, network capability and content mark-up language capability. In a feasible example of use of the system:

1. The customer accesses the platform by its device, hence the system identifies the user and characterizes the access mode.
2. A session profile is built according to the device features and the connection parameters.
3. A personal agent is built and is initialized with both the user profile and the session profile by which the security level, the best appreciable QoS are defined.
4. The agent shows to the user the set of functionalities he is allowed to.
5. The user is able to ask for a service. An available version is discovered and brokered according to the to the service requirements, the session profile and the user preferences.
6. The client side of the application is provided by the service vendor; it is retrieved by the agent and forwarded to the user as a web page, a mobile agent, an applet or an embedded application that needs to be downloaded and installed.

According to a first coarse grain classification [31], we suppose here to have four classes of client devices. The considered devices types are different one from another because of their memory capacity, their computational resources, their display characteristics, their allowed connection mode and so on. Depending on the above features the platform and the client should be able to broker the best set of parameters which allow the session to be opened. Some of these parameters are the communication protocol, the Human Computer Interface (HCI), what applications have to be executed locally or remotely, the level of trustful, etc

1. A first class of devices includes all the laptops, pc and more powerful machines, which have no limited resources in our application fields.
2. The second one is composed of those devices, like palms, which have its own embedded Operative System and are able to download and install some software in order to perform any kind of application, but have limited resources. Here the user could choose to reserve the minimum amount of resources but waiving a good level of interactivity.
3. JAVA enabled smart phones belong to this class. They are able to access the system by downloading an APPLET that implements the application client.
4. The last class is composed of that handheld devices which do not support reconfiguration. They allow just the browsing of a web pages, implemented in the HTML, WML or similar languages.

Many solutions can be conceived by different vendors for developing client applications targeted to different classes of devices.

4. System architecture. The system architecture has been designed aiming at two main targets. The first one is the interoperability, in order to support the integration of new third party services. The second one is the automatic management and composition of services by software agents able to access well known standard interfaces. In Figure 4.1 is shown a component-based model of the designed architecture. Some components provide basic facilities and are embedded in the platform, some other functionalities can be integrated as external services accessible by a Web Services interface. In the first prototypal version the Mobile Agent engine is implemented by the ASDK (Aglet Software Development Kit), but we have already developed a porting to JADE, a better supported and FIPA standard agent platform. A mobile agents platform supports the mobile agent development and execution providing an Agent server and the basic mechanisms of the programming paradigm such as agent creation, cloning, migration and communication. The original platform was extended in order to integrate new features at different levels. Some facilities have been integrated inside the agent server, the other are external components or they have been developed as application agents. The platform behaves both as a services provider and as a services requestor. In the first case Web Service are exploited to provide vendors with the possibility:

- to deploy their services;
- to deliver them with added values by services composition;
- to exploit the Virtual Home Environment of the platform customers (authorization, personalization, ...).

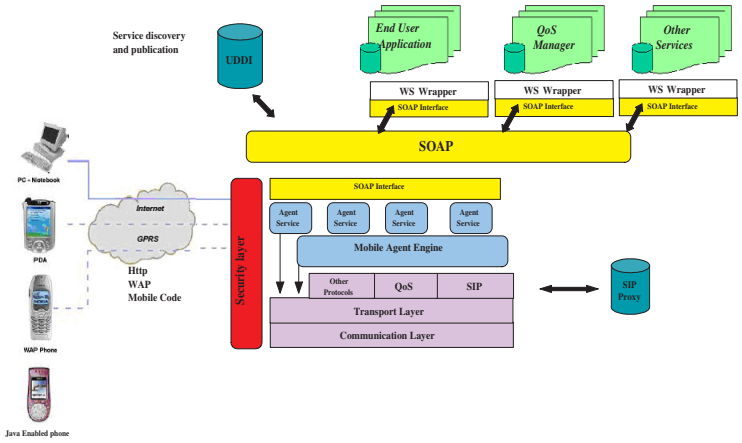

FIG. 4.1. *The system architecture*

Furthermore it allows service requestors:

1. to broker the best implementation according their client profile;
2. to exploit transparently through the same interface different service which would not be usable without the support of the platform.

In fact the platform is able to exploit the implementations deployed by trusted providers and composes them to build new value added services or to increase the availability of the existing ones building multiple alternatives. It supports the user in their exploitation and personalizes the access in order to optimize the customer satisfaction. The UDDI register is employed to support the publication of new services by the vendors and to discover the available implementations. Searching for a service implementation the client gets the WSDL description that provides methods and parameters of the implemented interface. Web services supports a kind of "technological" interoperability. The WSDL interface and SOAP protocol allows the service invocation by any kind of application, rather than by a human user. Nevertheless another issue to be addressed is related to the automatic exploitation of the services. The question is how the requestor should invoke the methods of the provided interface to exploit the service ? What is the behavior of the selected implementation ? What are the input/output function implemented by each method ? In order to overcome these problems many solutions have been proposed in related works, but we are still dealing with an open issue. Some of them deal with the automatic exploitation of the discovered services basing on the definition of standard taxonomies. Some others investigate how to understand the way of using an application exploiting a semantic analysis of a formal or semi-formal description of the service functionalities. To support the services integration and their exploitation we conceived the following model that allows the plug-in of new components both to expand the platform functionalities and to deliver new kind of services to the customers. The plug-in model is shown in Figure 4.2. We can see an agent as a new functionality provided by the platform (we can also have more agents collaborating to provide just a single functionality). Every agent is able to drive a class of services by a standard interface that needs to be implemented by the providers in order to deliver their service through the platform. In our model a service is characterized by a 3-tuple of elements:

1. a class,
2. a wsdl interface,
3. an agent driver.

The class is identified by a label and represents all the services which can be exploited by the same interface. In order to deploy a service within a chosen class the provider needs to implement the defined interface. Like the Operative System is able to drive a peripheral device through a program driver, in the same way the platform exploits the generic implementation of the discovered service within a class by a software agent. In order to handle a new kind of service the platform administrator needs to define a class (label, interface) and to implement the agent driver for that service. When the provider is publishing a new service he needs to select the class and to provide its WSDL together with some other information. A software authority performs a check to verify the compliance between the interface described in the WSDL and the one required in the selected class.

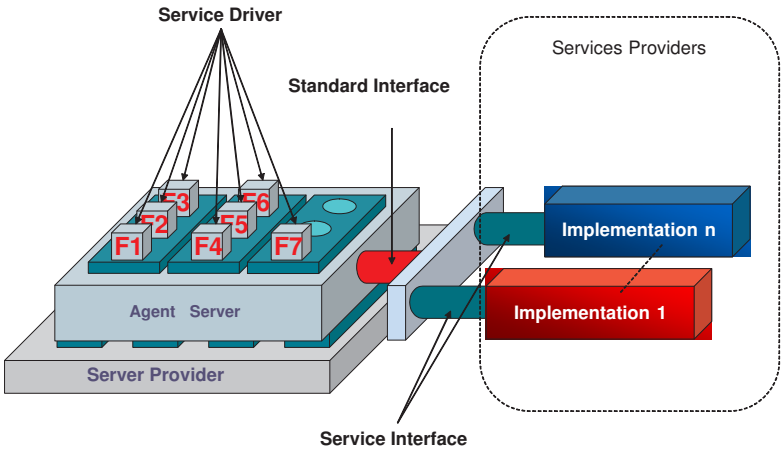

FIG. 4.2. *The service integration*

If the check is successful the new service implementation is published in the UDDI register. Note that this deal with a syntactic compliance but we do not grant a semantic correctness yet. Another kind of agreement could be checked by defining a set of test to be performed before to make available the service (we could check the output when a set of defined data is provided in input). Some other functionalities as the access manager and the agent localization facility are embedded in the platform; we mean they have been developed by extending the agent platform or providing new APIs or applications. We will deal with them in the following making clear the semantic of some components depicted in Figure 4.1 as the SIP register and the security layer. Some examples are:

- device and server localization;
- remote client management (upgrade, healing, backup, reconfiguration, ...);
- personalization (profiles management , context awareness, HCI , content management);
- services management (service integration, deployment and delivery, service reconfiguration);
- billing;
- user information (address book, remainders, e-mail)

Actually a lot of prototypal components and protocols have been developed. Some stand-alone applications are available to show how the new functionalities work, further a prototypal framework that integrates some of them is provided. The prototypal implementation is distributed on three workstations with the following roles: a server provider, a register, a service provider. The server provider supports the authentication of the user by different mechanisms, service publication, removal and discovery. Further it supports the hosting of the user agents (when the the device is not suited to execute them) and grants the persistence and the resumption of the active services. All the interaction with the register, such as services discovery and publication, are performed at destination exploiting agent migration. An agent is dispatched where the register resides in order to perform the search locally and to return just the results (we minimize communication and we support proactive connectionless discovery). In our prototypal implementation the services are implemented by mobile agents themselves and they are accessible by a Web Services interface. The agent server has been extended in order to be executed as a web application. Its Web Services interface exports those methods which allow to interact with agent based application and to exploit basic agent mechanism.

5. Interaction model. A relevant issue is how the client application can exploit the platform facilities and the services delivered. Here we propose a common approach that can be applied in general and does not depend by a specific technology. The interaction model among client, middleware and service provider allows to design and develop each software module independently. The middleware developer looks at services and clients as black box entities. Transparent access to the services must be provided. When they exists, client

```
public interface MagdaWS extends Remote {
public String connect(String login, String pwd, String profile);
public void logout(String token);
public String[] listServiceType(String token);
public String find(String serviceType, String token);
public String getResponse(String requestID);
public String[] getTokenList(String token);
public void closeSession(String token);
}
```

FIG. 5.1. *MAgDA Web Service interface*

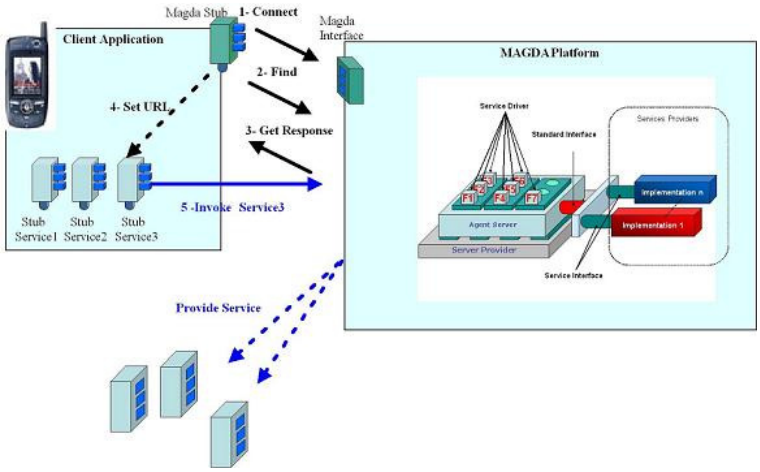

FIG. 5.2. *The middleware interaction protocol*

self-reconfiguration capabilities are unknown by the middleware. As any other requestor the client application will be conceived to work with a well known kind of services whose wsdl has been defined and has been made publicly available by the provider. The application will invoke remote services by a set of stubs (S1, ..., Sn), which have been built by their wsdl. The extension of the client application in order to exploit the middleware facilities is straightforward. The interactions among requestors and the platform take place just before to invoke a service and they initialize the session before the application is starting. At client side, during the configuration phase, the information necessary to bind the stubs to the right providers is collected. Meanwhile, at middleware side, the required services are brokered and the platform is reconfigured to support the service exploitation by the new client profile. Platform reconfiguration is exploited to adapt the service exploitation but also to allocate the necessary resources for an effective dispatching of the incoming workload.

As it is shown in Figure 5.2 the new client will be composed of the original application and of an access manager (in Figure MS—Magda Stub) that is used to exploit the facilities provided at middleware-side.

The MAgDA WS interface is shown in Figure 5.1.

In Figure 5.2 the interaction protocol between the client and the middleware is described.

The *connect* method allows the client to access. It needs to communicate the user credentials and its profile for that session. For each successful authentication a personal agent is created in any container of the platform which can be distributed geographically. The token returned identifies the personal agent for that session and must be used for all the following requests. Let us consider that Web Service technology allow to deploy stateless services. Here personal agents represent active sessions for logged users. The *logout* method closes the session without disposing the personal agent. The *closeSession* method disposes the personal agent. In order to know those services which can be exploited by the platform the *listServiceType* method can be invoked. The *listServiceType* method returns the list of service categories which are available in the platform and the wsdl address for each of them. If the exploitation of the desired service is supported by the platform the client can ask for the binding information needful to initialize its stubs. The wsdl is used to build the service interface while the binding must be retrieved whenever a new session is opened because it depends by the client

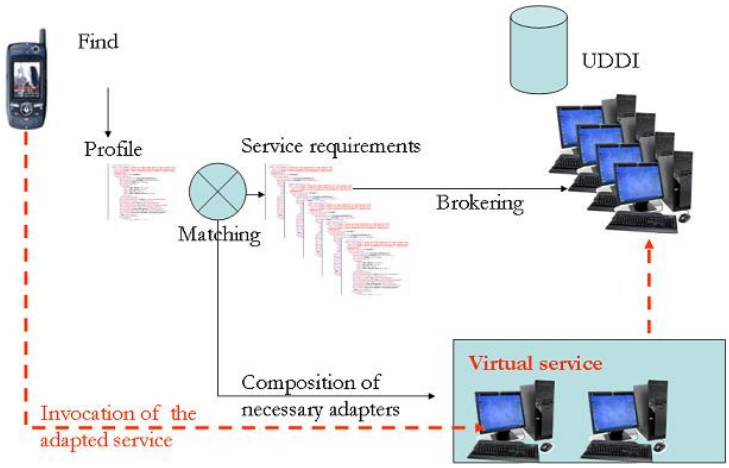

FIG. 5.3. *The middleware interaction protocol*

profile and by the dynamic condition of the system. The *find* method is used to broker the service and returns a *requestID* that identifies the request. Its invocation is not blocking. The received token will be used to get the response, when it will be available, by the *getResponse* method. The obtained link will refer to the brokered service. The result will be used to initialize the client stub before to start the application. The *getTokenList* method retrieves the list of requestIDs of all pending requests belonging to the session identified by the same token.

The invocation of *find* method of MAgDA interface starts the procedure shown in Figure 5.3. For each available service able to provide the desired functionality a matching between the service requirements and the client profile is performed. If a compliant service is found, its address is returned to the client in order to bind the stub to the right provider. When the compliance cannot be granted by the service the platform selects the service implementation that fits as well as possible the client capabilities. Then it can provide the necessary adaptation exploiting internal components or composing external services. In this case the binding information returned to the client will refer to a virtual service that is executing on the platform. Here a set of agents, acting as a bridge, processes and throws client request from an agent to the other until the message reaches the brokered provider. In the same way the response flows back to the client across a set of adapters. A tunneling of the SOAP messages allow us to distribute the workload and to adapt the contents transparently. Message elaboration along the outward path is exploited to distribute the workload among different providers or to adapt the input parameters of the brokered services. Many instance of the same bridge can be distributed on different nodes in order to offer alternative paths, which can be exploited for different profiles, or which can be replicated to distribute the workload. The backward path is exploited to adapt the contents returned by the provider.

Other approaches have been conceived for different kind of devices to exploit at the best particular functionalities. Agents could be pushed on the client to perform a reconfiguration or a software update when their execution is supported locally. Of course this model needs a larger amount of computational resources and memory. Furthermore application developers must deal with mobile agents technology. When the user owns a smart-phone or a too simple handheld device, a personal agent that supports the interaction between the client and the services could be created on the remote machine. The agent accepts the user's requests and invokes on the server the desired services. The user interface and the replies are forwarded to the client by the agent. Also a specific implementation of the Human Computer Interface needs to be provided for the target client device. This different approaches are shown in Figure 5.4. Hybrid solutions can be conceived splitting the application in a client agent and a back-end agent when the limited resources cannot carry out the execution. Hence the platform provides to the user services and the resources to support their execution.

6. A prototypal implementation. The prototypal implementation of the described platform represents for us a test-bed for the integration of the experimented technologies and the test of the defined applications and protocols. As it is shown in Figure 6.1 the platform implementation is built upon a mobile agent server. Mobile agents implement the platform facilities; multiple instances can serve many clients at the same time. In order to support interoperability with different technologies, which are not agents based, we deployed the

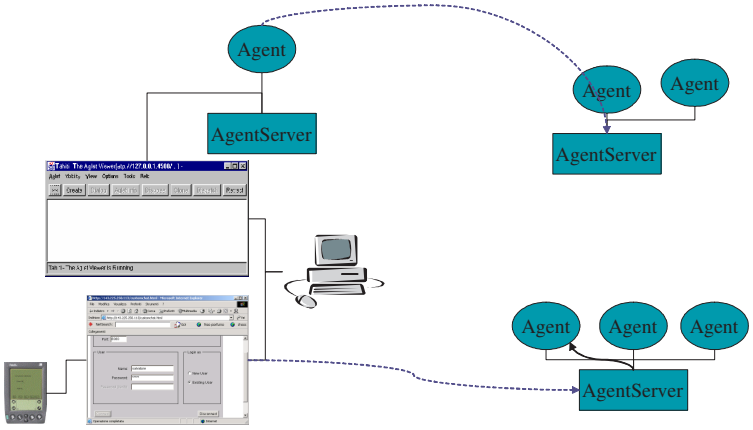

FIG. 5.4. *Interaction models*

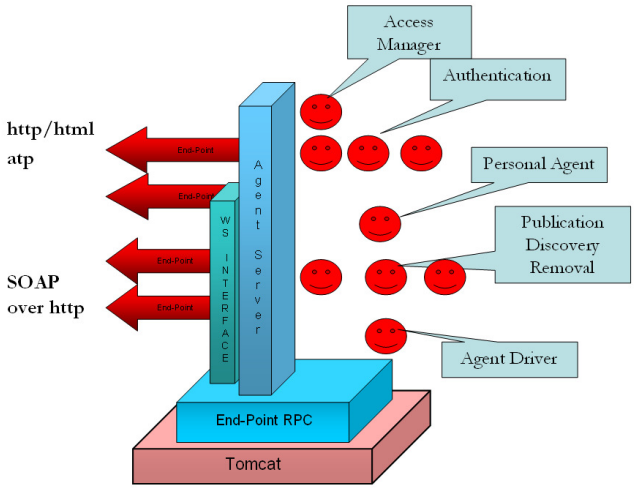

FIG. 6.1. *The Server Provider*

agent server as a web application inside the Tomcat Application server. In this way any requestor is able to invoke the platform facilities also by a web service interface. The application server needed to be patched in order to handle also the ATP protocol for agent communication and migration if the Aglet version is used, on the other hand the Jade version exploit the already supported http protocol. We provided a web interface for end-user services and an agent client for agent-enabled devices. When it needs the user is able to communicate, by a web browser, with the agents which are executing on the remote server. An agent can be identified by an URL and can handle a number of URL aliases. The http request is parsed and translated by the agent server using an agent communication language messages and it is forwarded to the agent in charge to handle it. The server provider is equipped with an agent server that hosts four resident running agents: an Access Manager and three authentication agents (each of them implements a different authentication mechanism). We provided weak (based on login and password) and strong authentication mechanisms (based on cipher algorithms). The simplest devices will support just the weakest ones and will allow the user to exploit the less security critical services. The authentication options offered by the Access Manager, at the state of art, are: 1) login and password, 2) challenge response based on the exploitation of a hash function, 3) challenge response with digital signature using a smart-card or 4) not. If the authentication is successful the user is able to be connected with his personal agents that could be already running or could need to be created. Each user is driven by his personal agent to exploit the platform facilities. In Figure 6.2 the administration panel is shown. A GUI

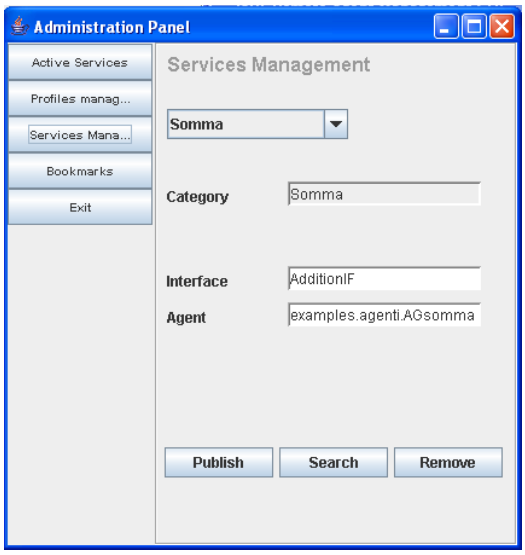

FIG. 6.2. *The administration panel*

panel implemented by a java agent can migrate on the user device and allows its owner to support asynchronous interactions and to reduce the on-line connection time. Therefore a web interface allows to exploit the services by a wide class of devices. The menu offers some already available facilities: active sessions, profiles management, services management and bookmarks. The first item allows the user to list his active requests (open sessions) and to resume or dispose the agents in charge of delivering the services. It needs when a user chooses to close a session and to resume it accessing the system by another device or from a different location. The second item allows the user to modify and store different profiles and to select the one he is exploiting in that moment. A profile is composed of personal information, device description (we are developing a facility to characterize automatically the user device), session properties, security level. A selected profile is exploited by the platform to personalize the discovery or to adapt the service. When it is supported by the devices an agent migrates to the client in order to collect automatically the profile information. Service management is described in detail in the following. It supports service publication, removal and discovery. The last item allows the user to list the saved bookmarks. The administration panel support the publication and the removal of new services, which are implemented by three mobile agents. The publication of new services can be exploited by a form that must filled with the WSDL address of the service and the class which identify that kind of application. An authority will automatically check the compliance of the services with the ones belonging to the same class by matching the WSDL with a pre-defined interface and will allow the publication. The discovery of a service is performed by a specialized agent who is able to migrate on the UDDI server and to perform the research close the data according to specialized algorithms. This approach optimizes the communications and the performance of the system. The results of discovery are filtered by a matching algorithm that compare the client profile with the service description. To select the suited implementations of a services among the published ones we perform a matching between two XML representations of user and service profile. We need that the service description and client profile should have the same XML schema and for each attribute should specify the minimum value required for the service exploitation. The matching algorithm is able to compute a distance between the two representation and to warn about the differences if it is required. The matching is performed by four steps:

1. Parsing e Hashing : each profile is represented as a DOM (Document Object Model Specification) tree (T1 and T2) and for each node an Hash value is computed.
2. Checking e Filtering : T1 and T2 are equivalent if the Hash value of the root nodes are equals. If T1 and T2 are different the algorithm looks for the differences comparing the sub-tree and branching the ones which result equals.
3. Matching: the algorithm computes the minimum-cost matching between T1 and T2 Mmin(T1,T2). Mmin(T1,T2) is a script that allows to transform T1 to T2 by a sequence made of the minimum number of simple edit operations (delete, insert,update) applied to the nodes.

4. Minimum-cost edit script generation: this provides the differences between the two XML documents.

Once the service has been chosen the next step is its invocation. We chose as case study a service for instant messaging. We defined the class "InstantMessaging" that is associated with a IMAgent and a IMInterface. An implementation of the services can be published into the UDDI register of our platform if its WSDL is compliant with the IMInterface. The definition of IMInterface methods and parameters is stored in an internal database that will be queried to check the compliance of a new implementation. To set up the demo we need to deploy the service and publish it by the platform administration panel. Any user now is able to discover the service and to enjoy it. When an implementation of a service belonging to the InstantMessaging class is invoked the platform creates an instance of the IMAgent and passes to it the URL of the service provider. The IMAgent books the service using the URL and receives some parameters for exploiting the service facilities (protocol, server address, session key,...). The IMAgent stores the session data, creates and sends an instance of the client application to the user device. If the user device is not able to host and execute the agent client a proxy agent is created. It provides a web interface and supports from remote the exploitation of the service. For example the proxy agent is able to handle the incoming call, , to store the list od contacts, the save the discussion, to set the status of the client (busy, not visible, on-line, ...), ... Of course the user can choose to disconnect his device and to reconnect by another one or from another location. In this case it will need just to authenticate himself again, to check the active services and to ask the right agent for a new player that will allow him to exploit the service resuming the state of the application. Finally we need to observe that these kind of services support the client and agent mobility. Both the devices and the agents can migrate across the network. The platform support the localization of its agents by a SIP (Session Initiation Protocol) that is used to resume the connection when the communication with agents fails. A common namespace managed by a SIP register allows to localize the agent and the hosting agent server.

7. Conclusions. In this paper we aimed to provide an overview of MAgDA, a software platform that provides to the users value added services by the composition and delivery of Web Services. Web Service Technology provides interoperability and the Mobile Agent programming paradigm is exploited to build the platform infrastructure. We described the functionalities of our system and its architectural model. We provided a WS interface to allow any WS requestor to exploit in a straightforward way advanced services provided by the platform. Finally we presented the state of art of a prototypal implementation and some mechanisms which have been integrated in the platform. Future works will deal with the integration of the all available components and the development of new functionalities. We designed and are already developing some services for automatic composition of web services and its choreographic execution. Furthermore we are going to terminate a porting to the Jade agent platform because better supported and compliant to the actual standards. Tests and measures in real application environments need to be planned and performed in order to evaluate availability, performance and scalability.

REFERENCES

[1] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *Mobile agents for distributed and dynamically balanced optimization applications,* Proc. of the 9th International Conference on High-Performance Computing and Networking (HPCN Europe 2001), London, UK, Springer-Verlag (2001) 161–172.

[2] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *Mobile agent programming for clusters with parallel skeletons,* Proc. of 5th International Conference on High Performance Computing in Computational Sciencesm, VECPAR'2002, Porto, Portugal (2002).

[3] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *A resource discovery service for a mobile agents based grid infrastructure,* Concurrent Engineering—Enhanced Interoperable Systems (2003) 1011–1017.

[4] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *Terminal-aware grid resource and service discovery and access based on mobile agents technology,* Proc. of 12th Int. Conference on Parallel and Distributed Processing (PDP 2004), A Coruna, Spain, IEEE Computer Society (2004) 40–45.

[5] R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque, MAGDA: *A Mobile Agent based Grid Architecture, Journal of Grid Computing, Springer Netherlands,* ISSN 1570-7873 (Print) 1572-9814 (Online), pp. 395–412, December 2006, Vol. 4 Num. 4.

[6] C. Wang, G. Wang, A. Chen, and al: *A policy-based approach for qos specification and enforcement in distributed service-oriented architecture,* In Proceedings of the 2005 IEEE International Conference on Services Computing IEEE Press, 2005.

[7] *I. O. for Standardization,* Iso/iec. international standard 13236 technology—quality of service: Framework, Dec. 1998.

[8] Z. Chen, C. Liang-Tien, B. Silverajan, and L. Bu-Sung: *Ux an architecture providing qosaware and federated support for uddi,* In Proceedings of the 2003 International Conference on Web Services, 2003.

[9] A. S. Ali, O. F. Rana, R. Al-Ali and D. W. Walker: *Uddie: An extended registry for web services,* In Proceedings of the Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference. IEEE Press, 2003.
[10] A. Kumar, A. El-Geniedy, and S. Agarwal: *A generalized framework for providing qos based registry in service oriented architecture,* Proceedings of the 2005 IEEE International Conference on Services Computing. IEEE Press, 2005.
[11] A. Keller and H. Ludwig: *The wsla framework: Specifying and monitoring of service level agreements for web services,* Research report RC22456, IBM, 2002.
[12] D. D. Lamanna, J. Skene, and W. Emmerich: *Slang: A language for defining service level agreements,* Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems. IEEE Press, 2003.
[13] L. Taher, R. Basha, and H. E. Khatib: *Establishing association between qos properties in service oriented architecture* Proceedings of the Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference. IEEE Press, 2003.
[14] V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian: *Supporting qos based selection in service oriented architecture,* Proceedings of the International Conference on Next Generation Web Services Practices. IEEE Press, 2006.
[15] G. Wang, A.Chen, C. Wang, C. Fung, and S. Uczekaj, *Integrated quality of service (qos) management in service-oriented enterprise architectures,* Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf. IEEE Press, 2004.
[16] Xu C. Z., Wims B.: *A mobile agent based push methodology for global parallel computing,* Concurrency—Practice and Experience **12** (2000) 705–726.
[17] Kreger H.: *Web services conceptual architecture(wsca 1.0)* (2001).
[18] Brittenham P.: *P. an overview of the web services inspectin language* (2002).
[19] R. Chinnici, M. Gudgin, J. M. S. W.: *Web services description language (wsdl) version 1.2* (2003).
[20] OASIS: *Uddi technical white paper,* Technical report, `http://www.uddi.org` (2000).
[21] Maximilien E., Singh M.: *Agent-based architecture for autonomic web service selection,* Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia (2003).
[22] Lyell M., Rosen L., Casagni-Simkins M., Norris D.: *On software agents and web services: Usage and design concepts and issues,* Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia (2003).
[23] cGroup W. W. S. A. W.: *Web service architecture recommendation (2004)* `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211`
[24] Maamar Z., Sheng Q. Z., Benatallah B.: *Interleaving web services composition and execution using software agents and delegation,* Proc. of Workshop on Web Services and Agent-based Engineering (AAMAS2003). (2003).
[25] Maamar Z., Sheng Q. Z., Benatallah B.: *On composite web services provisioning in an environment of fixed and mobile computing resources,* Information Technology and Management **5** (2004).
[26] Ragusa C., Liotta A., Pavlou G.: *Dynamic resource management for mobile services,* Proc. of 5th International Workshop on Mobile Agents for Telecommunication Applications (MATA-03), (2003).
[27] Bohoris C., Pavlou G., Liotta A.: *Mobile agent-based performance management for the virtual home environment,* Journal of Network and Systems Management **11** (2003).
[28] Kifer M., Lara R., Polleres A., Zhao C., Keller U., Lausen H., Fensel D.: *A logical framework for web service discovery,* Proc. of Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications (ISWC 2004). Volume 119, Hiroshima, Japan (2004).
[29] Buhler P. A., M. Vidal J.: *Semantic web services as agent behaviors,* Burg B. et al. (eds), Agentcities: Challenges in Open Agent Environments, Springer-Verlag (2003) 25–31 `http://jmvidal.cse.sc.edu/papers/buhler03a.pdf`
[30] L. Bettini, R. De Nicola M. L.: *Software update via mobile agent based programming,* Proc. of SAC 2002, Madrid, Spain, ACM 1-58113-445 (2002).
[31] *3rd Generation Partnership Project Technical Specification Group Services, Aspects, S.:* Mobile station application execution environment (mexe); functional description; stage 2; arib std-t63-23.057, Technical report, 3GPP (2001).

Edited by: Dana Petcu
Received: July 15, 2007
Accepted: August 6, 2007