Wireless Web Services using Mobile Agents and Ontologies

Vasileios Baousis, Elias Zavitsanos, Vasileios Spiliopoulos,
Stathes Hadjiefthymiades, Lazaros Merakos, Giannis Veronis

*University of Athens, Department of Informatics & Telecommunications,
Communication Networks Laboratory, Panepistimioupolis, Ilisia, 157 84 Athens, Greece.*

e-mails: {bbaous| std00025|std00108| shadj|merakos|std00024}@di.uoa.gr
Tel: +30 2107275425 / Fax: +30 2107275601

Abstract--**We discuss the integration of two contemporary service technologies: Web Services and Mobile Agents. We exploit the capabilities offered by Mobile Agents to query and invoke semantically enriched Web Services without the need for simultaneous, online presence of the service requestor. Such service setting is ideal for wireless/mobile computing, where user terminals are not necessarily online during their entire session. To improve the capabilities of Service registries met in the Web Services reference architecture, we exploit the advantages of the Semantic Web framework. Specifically, we use enhanced registries enriched with semantic information that provide semantic matching to service queries and published service descriptions.**

***Index Terms*- Mobile agents, Ontologies, OWL-S, Semantic Web services.**

I. INTRODUCTION

Nowadays the ability to access services and obtain information anywhere and anytime, irrespective of the network and terminal, is imperative to meet users' requirements. However, most of the services available on the Web are designed to be accessed from desktop PCs, with a fixed, low error connection to the network. The main pursuit of most research efforts is to extend current services and applications designed for fixed networks to mobile users in a seamless and transparent way. This task is cumbersome given the problems encountered when using a hand-held device to access such services. Scarce bandwidth availability, temporary disconnections, high latency, limited battery and constrained processing capabilities are often met in wireless environments.

Web Services (WS) are designed for fixed networks. WS provide a loosely coupled infrastructure for service description, discovery and execution. In the traditional WS model, service requestors find the appropriate service by placing a request to the service registry, often implemented with UDDI (Universal Description, Discovery and Integration), obtain the result(s) - public interfaces of the chosen service(s) (expressed in WSDL - Web Services Description Language) and, finally, send SOAP (Simple Object Access Protocol) messages to WS provider(s).

The main problems experienced in these interactions are:

- UDDI guarantees syntactic interoperability, and it does not provide a semantic description of its content. UDDI is characterised for its lack of semantic description mechanisms, such as semantic interoperability, explicit semantic models to understand the queries and inference capabilities. UDDI service discovery is performed primarily by service name (keyword matching), not by service attributes/capabilities. UDDI tModels may be regarded as a vocabulary where service descriptions are unstructured and intended for human comprehension. Different services with the same capabilities can, thus, be categorized in different business categories.
- SOAP messages overhead is often greater than the service parameters/results exchanged between communicating parties.
- WSDL is XML-based and used to specify the interface of a WS. It describes what information is exchanged (structure of the SOAP messages), how that information is exchanged via interactions with the WS (transport protocols) and where the service is located. However, WSDL does not contain any information about the capabilities of the described service.

Several efforts have been made to address the lack of expressiveness in WSDL in terms of semantic description that fall into the area of the Semantic Web (SW). SW is a vision [1] in which web pages are augmented with semantic information and data expressed in an unambiguous manner and can be understood and interpreted by machine applications and humans alike. This requires means to represent the semantics of the exchanged data so that it could be automatically processed. This requirement is met with the use of ontologies. Ontologies facilitate knowledge sharing among heterogeneous systems, through explicit formal specifications of the terms used in a knowledge domain and relations among them [2]. Ontologies are machine-understandable and, as such, a computer can process data, annotated with references to ontologies. Through the knowledge encapsulated in the ontology, a computer can deduce facts from the originally provided data. The use of ontologies enables systems to share common understanding of the structure of information and reuse of domain knowledge, make domain assumptions explicit and separate domain knowledge from the operational knowledge.

OWL-S (Web Ontology Language) [3] is a SW description language for defining and instantiating Web ontologies. An OWL-S ontology implicitly defines message types (as input/output types of processes) in terms of OWL classes, which allows for a richer, class-hierarchical semantic foundation. With OWL-S, WS are described in an unambiguous manner allowing for a potential service requestor to place a capability search in a service registry rather than a keyword search in UDDI registries.

In this paper, we propose the integration of Mobile Agent (MA) technology with WS that are expressed in OWL-S. A MA has the unique ability to autonomously transport itself from one system to another. The ability to travel allows a MA to move to a system that contains an entity (-ies) with which the agent wishes to interact and take advantage of being in the same host or network with the collaborating entity. MA operate asynchronously and are equipped with the appropriate intelligence to dynamically accomplish their task. MA are not trying to replace traditional ways of communication but to enhance the functionality and operation of the involved service entities. Researchers agree that MAs are not always the best solution and a combination of the MA, client-server and remote execution paradigms delivers the best performance with respect to network operation metrics like bandwidth, response time, and scalability.

This paper introduces a novel framework for dynamic discovery and integration of semantically enriched WS with MA. Specifically, the proposed framework is mostly intended for wireless environments where users access WS in the fixed network. The system uses an enhanced WS registry enriched with semantic information that provides semantic matching to incoming service queries and the published WS descriptions. The framework enhances the fixed network with the intelligence needed in order to dispatch the service requests of the wireless user in an efficient, reliable and transparent manner. Furthermore, as shown in the performance evaluation section, the MA framework can be quantitatively compared with systems implemented without MA. However, our objective is not to outperform the existing systems, but rather provide a framework that solves the problems encountered in wireless environments. The proposed approach enables users to execute multiple services with minimum interaction, without the requirement of being online during the entire session. Additionally, the proposed framework provides better fixed network utilization since unnecessary communication overhead is avoided and reliable delivery of the service results is provided.

The rest of this paper is structured as follows. In Section II we discuss relevant prior work. In Section III, we present an overview of the proposed architecture. In Section IV, we study the performance of the proposed framework and in Section V we discuss our findings. Finally, conclusions and directions for further work are included in Section VI.

II. RELATED WORK

In this section, we provide an overview of the related work performed in the areas of semantic WS and multi-agent systems and, especially, on research activities that integrate these two technologies.

In [4] [5] BPEL (Business Process Execution Language) is used to form simple rules to describe MA physical behaviour (e.g., migration and cloning). Such simple rules are separated from the integration logic, allowing for addition or change of physical behaviour without modification of the BPEL description. A MA is composed of three parts of descriptions: (1) a BPEL process description expresses the integration logic (2) behaviour rule descriptions that drive the physical behaviour of the agent, and, (3) packed services that are carried by the agent and invoked locally by BPEL process. The separation of the agent's physical behaviour from integration logic is considered helpful in dealing with the dynamic environment of WS, however, the discussed framework supports actions only in case of predefined events. Such events are agent migration failures, agent cloning and task allocation to agent clones. The implemented rules do not consider dynamic events that might be generated during WS invocation and MA roaming. Moreover, directory services and multicast protocols are assumed pre-existing. Finally, the system has not been implemented yet, hence benchmarking is not possible.

In [6], the authors present a technique for providing agents with dynamically configured capabilities, described with DAML-S, which can represent atomic or orchestrated WS. According to this technique, MA implemented in LEAP reside in mobile devices and obtain the descriptions of the WS they wish to invoke from a repository server. Such descriptions are then forwarded to the Home Server where they are transformed to executable programs and results are communicated back to the MA through JXTA.

There are several proposed models that adopt BPEL4WS (Business Process Execution Language for Web Services) as a specification language for expressing the social behaviour of multiagent systems and adapt to changing environment conditions [7], [8]. However, these models do not provide for the semantic description of the WS involved in the system. Other proposed frameworks adopt DAML-S for describing the WS, thus, allowing for service capability search and matching [14],[15]. However, they do not provide any mechanism for combining these services.

In [10], [11] the authors propose a policy based framework for flexible management and dynamic configurability of agent mobility behaviour in order to reduce code mobility concerns and support rapid mobile code-based service provisioning. Policies specify when, where, how and the parts of the agent that will perform a given task (e.g., migrating to a host and invoke a service).

In [9], the authors suggest a model that provides the WS client runtime information pertinent to its execution and business logic. The client decides autonomously to bind to the best service that currently meets his requirements (such as server load, QoS, response time, etc). Two mechanisms are considered to gather this information: Remote Procedure Call (RPC) and MAs. The concept of the circulating agent is also introduced (i.e., to periodically travel through the network and retrieve updated information). This approach provides better response times and exhibits improved behaviour in wireless environments.

An agent based approach for composite mobile WS is proposed in [12], where three methods for compositions are discussed: parallel, sequential and a hybrid of these two. The service composition scenario is that a user with a wireless device places a request to execute a WS and a MA executes the service on the behalf of the user by moving to the service registry, query the registry, get service description (in WSDL), and finally invoke the service. Service execution, depending on the WS itself, is performed with one of the aforementioned composition methods. This approach does not consider semantic information describing the involved WS, thus, services are selected by simple keyword queries to the UDDI registry. Additionally, it does not include mechanisms to decide which composition approach to follow. Integration depends upon the nature of the WS (if the service is a composition of other services it must be accessed sequentially, if not, then in parallel). A similar approach is proposed in [13], with the difference that a personal and a service agent are used to perform the task of the MA described in the previously mentioned approach.

III. FRAMEWORK ARCHITECTURE

Figure 1 depicts a general outline of our approach. The proposed framework consists of the mobile user that uses WS, the MA representing the user in the fixed network, the service registry and the WS provider. The last two entities may be implemented as stationary agents. According to the service implementation scenario, a mobile user accesses the proposed system and places service requests specifying some criteria. Subsequently, the system creates a MA that migrates to the registry to find the WS that best meets the user requirements. Service registry allows for a capability search to be performed, since it is enriched with semantic information. The MA, after acquiring the WS listing and technical details, migrates to service provider(s), invokes the WS, collects the results and returns to the service requestor to deliver the results to the user.

The route of the agent may vary, depending on the service requestor preferences and the network topology. As explained below, the user may dynamically force his MA to send its clones to the providers, invoking the services in parallel, rather than serially migrate to each one. Moreover the user may force the MA to implement different service execution strategies (e.g., execute all services locally or remotely, change timeout limit), during its itinerary and execution of service(s).

Our framework consists of the following functional components: (1) User Service Requestor (USR) and the Client System, (2) Mobile Agent (3) Provider Stationary Agent (PSA), (4) Registry Stationary Agent (RSA), (5) Semantic Web Services Registry (SWSR) (6) Web Service Provider (WSP). Their structure and functionalities are described below. In the end of this section we provide a service implementation scenario, presenting all possible supported service invocation alternatives.

A. *User Service Requestor (USR)*

USR is the client that invokes a WS. USR logs into the Client System, which communicates with the agent platform using IIOP. The agent platform is responsible for creating and handling MA, according to user specifications. The Client System is implemented in JSP/Servlet technology, and many users can be accommodated without having java runtime environment (JRE) or the MA platform (MAP) installed on their device. The only requirement is a browser to access the Client System. The Client System offers services to clients like: account creation, user login/logout, service invocation policies profile editing, and control of existing agents. Moreover, the administrator is allowed to add/remove/edit user properties/profiles. Finally, users' service invocation policy profiles are serialised and stored into the server's database that enables the seamless and transparent provision of services.

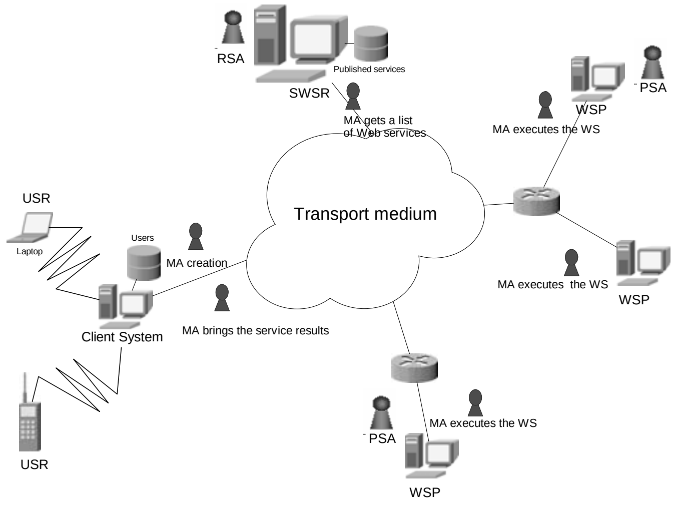

Figure 1: Framework Architecture

B. *Mobile Agent (MA)*

The MA is the representative of the user in the fixed network and is capable of roaming, finding and executing services and delivering results to the user. The MA may also spawn clones that execute the selected WS in parallel to minimize the total processing time. Clones can migrate and invoke simultaneously the chosen WS and return to the service requestor with the results. The MA has the following components: (1) data state, (2) code, (3) migration and cloning policies, (4) matching engine, and, (5) policy management component (Figure 2).

The data state component contains the information carried by the MA during migrations. The policies component specifies the autonomous behaviour of the MA. It should be noted that the social behaviour of the MA (migration, cloning) is separated from integration logic and code implementation. This separation is accomplished with user's specified invocation policies that govern the behaviour of MA, being external and independent of its code and integration with the WS. The policy management component is responsible for the MA external communication and the transparent installation of policies into the agent's repository.

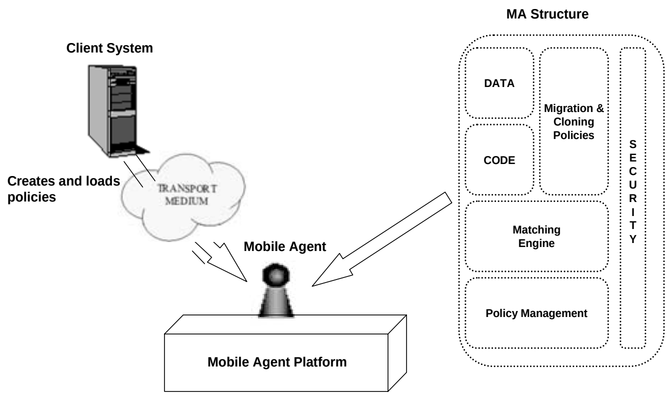

Figure 2: Mobile Agent Structure

As shown in Figure 3, the policy management component provides four services, namely communication, trigger, specification and policy repository. Specifically, the communication service enables the MA to interact with the client and other network entities. Such functionality is accomplished through the monitoring service which filters the messages coming from the client system and through the event service which handles events concerning policy changes. When a policy change occurs, trigger service is notified to update the policy repository. Specification service is responsible for fulfilling this task. Finally, the matching engine component is responsible for post-processing the service registry query results, i.e. confirm the availability of the service providers prior to agent migration.

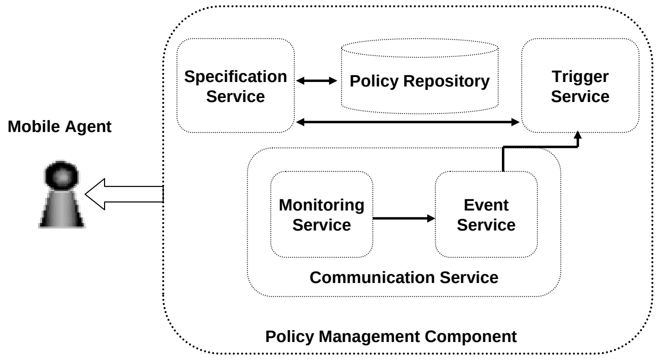

Figure 3: MA Policy Management Component

As mentioned above, the agent's policies determine its physical behaviour while roaming in the network and executing WS. Currently, the MA considers the policies (some of them described in Table I), which are Boolean (marked **B**) and numerical variables (marked **N**). Agent policies are expressed in XML and stored in a serialised format into the Client System database. For each registered user there is an associated policies file, to provide personalized WS access.

Table I: Policy names and their respective meaning

Policy name	Description
<Migrating> and <cloning>	MA's ability to migrate to another host and spawn clones respectively. (**B**).
<retryTimes>	The number of attempts that the MA will perform when a WS is unavailable. (**N**)
<timeBetween-Reattempts>	The time that MA will wait between consecutive reattempts. (**N**).
<suspend-WhenFinished>	States if the user wishes (dis)-connected operation. (**B**).
<maxNum-berOfHits>	The maximum number of services to be invoked. (**N**)
<minNum-berOfResults>	The minimum number of results when querying the semantically enriched registry. (**N**)
<pingServer>	MA will check if the targeted service provider is alive, before its migration.(**B**)
<migrate-ToServer>	Specifies if the service will be invoked locally or remotely. (**B**).
<remoteCall>	MA invokes the chosen WS using SOAP/RPC (remotely from other host) without migrating to each provider. (**B**).
<callThrough-Stationary>	Indicates if communication between WS and MA will take place with or without the Provider's Stationary Agent (PSA). (**B**))
<clone-ToServer>	Allows the MA either to serially migrate to each service provider or sent clones to accomplish the task in parallel. (**B**).

C. *Provider Stationary Agent (PSA)*

PSA is a stationary agent that resides in the host offering a certain WS. The purpose of the PSA is to wrap the functionality of the WS. The PSA communicates with the service providers through protocols specified for WS invocation and interaction (e.g., SOAP). When the MA migrates to a host offering a WS with a PSA, it obtains the results through the PSA. This communication is performed with RMI (Remote Method Invocation), instead of the resource-consuming SOAP. With this approach, the MA need not be SOAP fluent, thus, leading to a lightweight implementation. Figure 4 presents the PSA structure. The PSA interface exposes the available methods of the WS as they are described in OWL-S. PSA consists of two parts: (1) its data state, and, (2) its code. PSA methods are multi-threaded to accommodate and simultaneously serve multiple MAs.

D. *Registry Stationary Agent (RSA)*

RSA is a stationary agent that acts as a broker between the MA and the service registry (Figure 5). RSA implements part of the registry's functionality and serves MA's requests. By using RSA in the WS registry, the MA does not have to be aware of the implementation specific functionalities of the registry, and, as such, different service registries can be used as long as RSA acts between WS registry and MA. The proposed framework can be used with many different registries that are currently available.

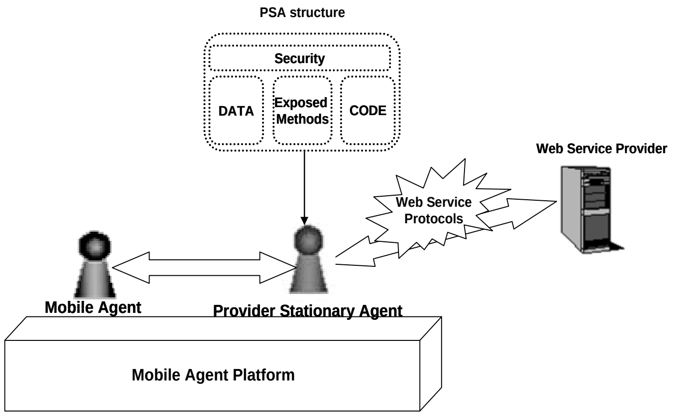

Figure 4: Provider Stationary Agent Logic

E. Semantic Web Services Registry (SWSR)

The SWSR (Figure 5) consists of the RSA, the matchmaking tool and the UDDI registry. The matchmaker [16] is a tool which enhances the UDDI server by adding capability-based discovery. In combination with Racer [20], it processes the OWL ontologies. Service advertisements are first processed by the UDDI server and, if any semantic information is contained in them, they are passed to the OWL-S matchmaking engine. Finally, the engine processes service queries and returns the results to the UDDI server, which in turn, communicates with the requesting service client.

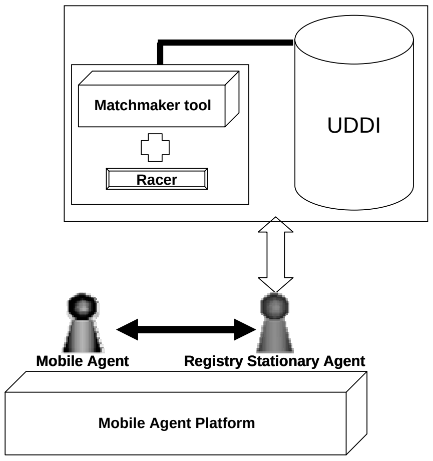

Figure 5: Semantic WS Registry

Matchmaker is a tool that integrates seamlessly with registries such as UDDI. In our system we used a local implementation of UDDI, called JUDDI [21]. JUDDI is a web application for Apache Tomcat that has the ability to deploy the functionality of the classic UDDI locally. The matchmaker tool is responsible for the mapping of the OWL-S service description to JUDDI. Matchmaker is plugged in JUDDI and is available in two versions, a Web-based and a standalone version. The standalone version provides a matching engine and a client API for invoking this engine. An extensive description of matchmaker can be found in [17][18][19].

F. Web Service Provider (WSP)

The WSP provides the WS to interested clients. It maintains a description of the WS expressed in WSDL and OWL-S (Web Ontology Language). Figure 6 depicts the WSP and their supported functionalities. Service invocation by the MA depends on the OWL-S description of the service. In our framework, service invocation by MA can be performed either directly or through the PSA. In the direct access case, the agent has to be SOAP fluent, a fact that increases the size of the MA when moving over the network. Inside the OWL-S description of the WS, it is indicated if a PSA wraps the functionality of the service to allow the roaming MA to interact with the PSA instead of the service.

As mentioned above, OWL-S is used to enhance the expressiveness of WSDL in terms of semantic information. For this reason, in our framework, WS are described both in WSDL and OWL-S. WSDL is used to describe the technical details (Service grounding) and OWL-S is used to specify the input and output ontologies, thus, enabling an advanced service capability search (Service profile and model). When the desired service is retrieved from the registry, the WSDL description is used to find the necessary definitions for its successful invocation.

As already mentioned, the WS provider can expose a PSA to act as his delegate and interact with the user's MA. This is revealed to the MA through the OWL-S description. If this is not the case, the MA infers that no PSA is offered and the service should be accessed directly.

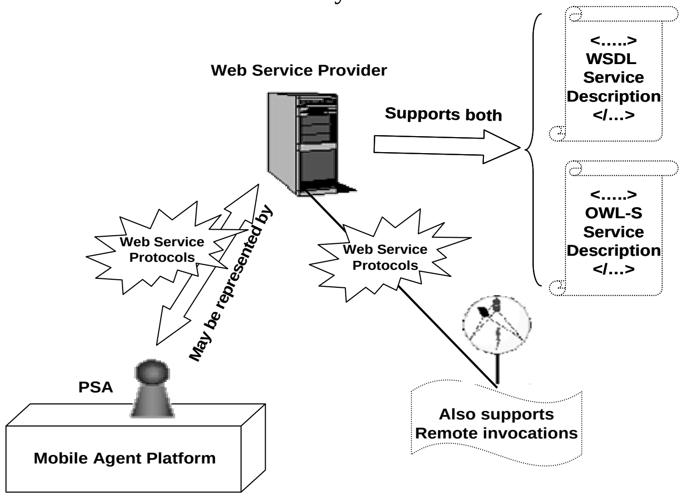

Figure 6: Web Service Provider

G. Service Description

In this section we provide a functional description of our framework, using a service scenario. As shown in the use case view (Figure 7), the USR wishes to find and invoke a certain WS, using a mobile device. Therefore, he connects to the Client System, the platform front-end. After a successful registration, the USR sets the desired criteria for the WS. The user may also define the MA service invocation policies and force the latter to follow a certain policy while roaming throughout the network. Subsequently, a MA is created to represent the user in the fixed network and dispatch his service requests.

The created MA is equipped with the user's unique ID, service invocation and agent behavioural policies. Such policies are passed to the MA in XML format and stored into his policy repository. The trigger service has the authority to change these policies, according to the messages that the event service may receive from the USR or other network entities.

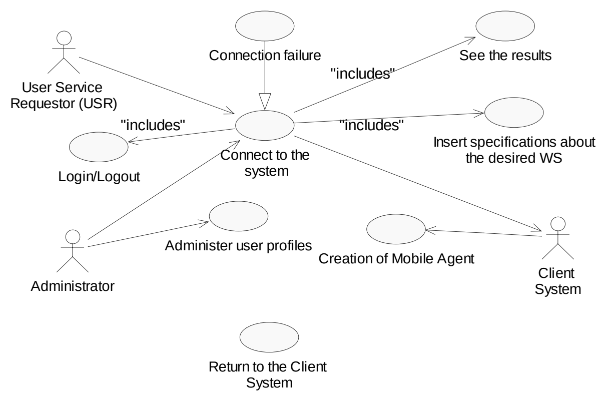

Figure 7 Use Case 1 : USR perspective

The SWSR (Figure 8) provides WS descriptions and allows service capability search. The MA, after creation, migrates to the SWSR. When the MA arrives at the registry, it communicates with the RSA, which will query the registry on behalf of the MA. The latter finds the service(s) that meet the user needs. The MA, after acquiring the results decides on the next step according to its specified service invocation and agent behavioural policies.

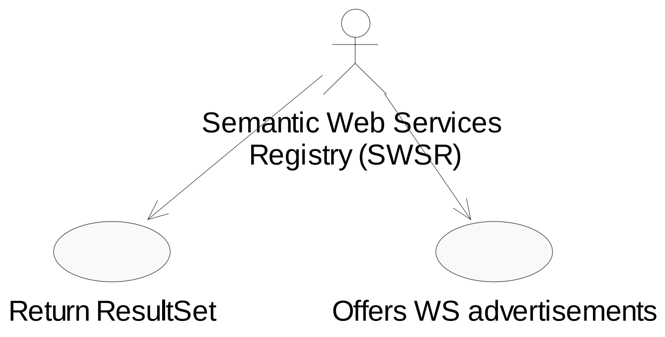

Figure 8: Use Case 2: SWSR perspective

The MA (Figure 9) may follow several WS invocation alternatives and these are listed below:

1. May poll the servers where the services are located to check their availability, in order to migrate only to those that are alive. In this way, the MA is released from the burden of migrating to a malfunctioning remote server. This strategy improves the overall performance of the framework, by avoiding unnecessary migrations.
2. May try to invoke the services from remote and not migrate to the provider. Remote invocation or migration of MA is specified in the MA policies. Specifically, depending on the size of the MA or the distance between its current location and the provider, it may be preferable not to migrate, but remotely invoke the WS.
3. May migrate to the WSP and collaborate with the PSA. The MA invokes the service and obtains the results through the PSA.
4. May migrate to the WSP and directly invoke the WS. This option requires the MA to carry additional code libraries. The implementation of the WSP is much simpler and straightforward, since there is no change in the traditional WS implementation model.
5. Finally, the MA may send clones to each WSP, instead of migrating serially to each one. This scenario results to a parallel invocation of the WS, with each MA clone to invoke one WS. In this way, the overall service invocation time is reduced, as expected, in comparison to the previous service invocation alternatives.

All these service invocation alternatives are decided at runtime, according to the user's specified service invocation and agent behavioural policies. When the MA(s) have collected the results, there are two scenarios depending on the selected policies:

1. When the MA invokes all the services, it migrates back to the Client System. If the user is logged in the system, the MA passes the results to the user, and, if the user is not logged in then the MA waits for the user to login and ask for the service results.
2. When clones had been used for service invocation, the MA clones return to the Client System and deliver service results to the father MA. After this interaction, MA clones are destroyed. The father MA, as in the previous case, delivers the services results to the user.

When the USR obtains the results, he may ask the MA to repeat one of the above scenarios by changing, if necessary, its policies, or he may cancel the execution of the agent. The USR may also, at any time, search for the agent, instruct him to return or cancel its execution at runtime.

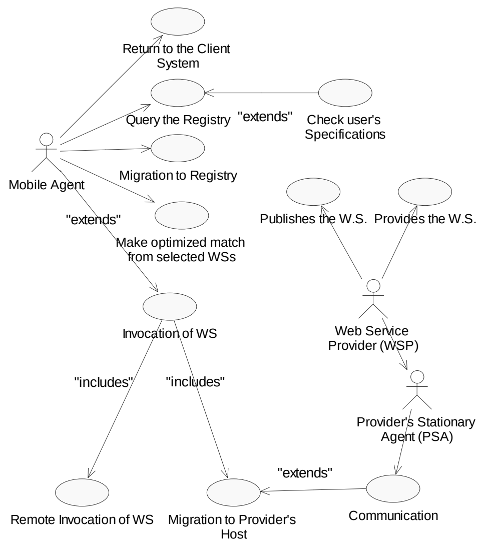

Figure 9 : Use Case 3: MA perspective

IV. PERFORMANCE EVALUATION

In this section we discuss the performance evaluation of the proposed system. Specifically, we compare the performance for different configurations settings, against the traditional business model of WS provision. In the following description, the term “conventional WS business model”, refers to the model where a user requests a service to be executed and the system dispatches (either automatically or with user intervention) the request by discovering the appropriate service(s) from the service registry, and then, sequentially, invokes these WS, receives and forwards/presents to the user the service results. Communication between the involved network entities is performed with SOAP. This model is a direct implementation of the WS architecture as described in [22]. Moreover, in our framework, mobile agents were implemented on the Grasshopper platform [23].

We have developed and tested the following four system variants:

a. A WS system implemented with the “Conventional WS Business Model” (**WSBM**).

b. Our MA framework <u>with</u> stationary agents in Service registry and Service providers - (**WITH PSA.**)

c. Our MA framework <u>without</u> stationary agents in Service registry and Service providers - (**NO PSA.**)

d. A hybrid system, where some Service Providers accommodate a Stationary agent, while others do not (**Hybrid.**)

The WS logic implemented in our experiments is quite simple. The WS have an extensive service description, stating unambiguously their capabilities in OWL-S. Such description is published in the registry (SWSR). However, the WS functionality is fairly simple, returning a pre-specified data volume subject to the service request. In our trials, service results are 1 KB, 10 KB, 100KB and 1 MB. Moreover, we have implemented 6 WS and distributed them in our testing network.

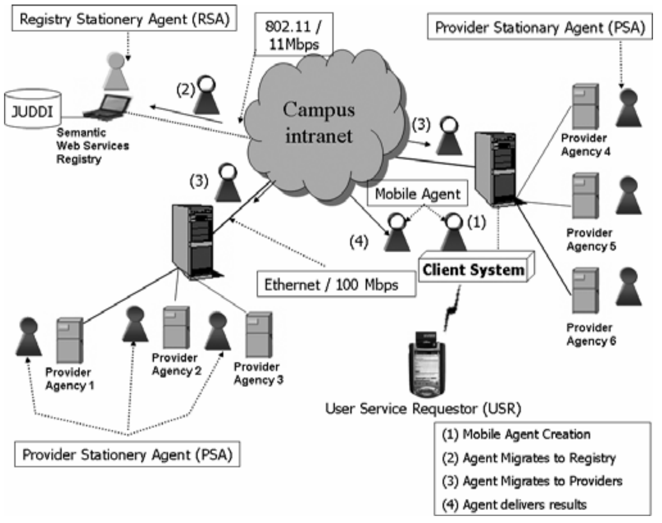
Registry Stationery Agent (RSA)
802.11 / 11Mbps
Provider Stationary Agent (PSA)
JUDDI
Semantic Web Services Registry
(2)
Campus intranet
(3)
Provider Agency 4
(3)
Mobile Agent
(1)
(4)
Provider Agency 5
Client System
Ethernet / 100 Mbps
Provider Agency 6
Provider Agency 1
Provider Agency 2
Provider Agency 3
User Service Requestor (USR)
(1) Mobile Agent Creation
(2) Agent Migrates to Registry
(3) Agent Migrates to Providers
(4) Agent delivers results
Provider Stationery Agent (PSA)

Figure 10: Performance evaluation network topology

In the performance evaluation run, a user requests a service, specifies some preferences and each of the above four systems dispatches this request. Furthermore, the service registry in each service request, replies with the same number of services (6). With this approach, each user request results to all the 6 WS to be executed in the same sequence.

The testing platform that we used is depicted in Figure 10. The system is a LAN composed of two workstations and a portable PC, all connected to the Internet through the University’s MAN. We measured the time of MA migration from one Service Provider to another and the interaction time of the MA with each service. In the “Conventional WS business model” we measured only the interaction service time (service registry query and service request and result times).

Below, we elaborate on the metrics that we have adopted in order to assess the performance of the system (variants). In Equation (1), Total Service Time (*TSTMA*) (for the MA platform) is the sum of Registry Interaction Time (*RIT*), Migration of MA to the i-th Service Provider Time ($MSPT_i$) and the Interaction Time with the i-th Service Provider ($ITSP_i$):

$$TST_{MA} = RIT + \sum_{i=1}^{N} (MSPT_i + ITSP_i) \qquad (1)$$

where N is the number of WS that are executed in the itinerary of the MA (6 in our tests). In the WSBM system, Equation (1) takes the form:

$$TST_{\mathrm{WSBM}} = RIT + \sum_{i=1}^{N} ITSP_i \qquad (2)$$

In (2) $ITSP_i$ is the time between service request submission and service results reception.

V. PERFORMANCE EVALUATION RESULTS

In Figure 11 we plot the migration time of the MA from one WS provider to another, against the service result size with the help of 2nd order polynomial interpolation lines. The line labelled “With PSA” indicates that WS results are delivered through the PSA. The line labelled “NO PSA”, means that no such agents are provided. Finally, the line “Hybrid” denotes the case where some providers are equipped with PSAs while others are not. We can observe, from Figure 11, MA in the “With PSA” system have constantly less migration time from the hybrid system and the latter has constantly less migration time from the “No PSA” system. Moreover, we observe that as the size of the service results increases the previously stated difference becomes more obvious, and the “With PSA” system performs better than the other systems. The MA migration behaviour of these systems can be justified because in the “With PSA” system the MA agent does not have to be SOAP fluent which means that it does have to carry extra code in order to support such communication. In “With PSA” system the MA is “lighter” than in the other two systems while the MA has to be equipped with extra libraries in order to exchange SOAP messages.

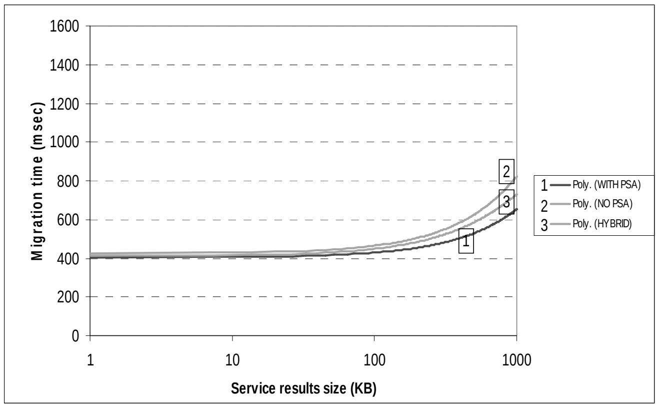

Figure 11: Migration Times vs. Service result size

In Figure 12, we plot the Interaction Time of the MA with the Service Provider (ITSP) against the service result size (presented through 2nd order polynomial fitting lines). The CL (Cloning) means that the MA that roams in the fixed network instead of invoking sequentially the WS, sends its clones to invoke each WS (parallel service invocation).

We notice that the WSBM system demonstrates smaller interaction time in all cases, and follows the "With PSA", the "Hybrid" and, finally, the "No PSA" system. Next, with poorer performance, follow the aforementioned three systems with CL enabled. The *ITSP* is relatively high in cases where cloning is enabled. This can be attributed to the small number of service providers used in our tests. With agent cloning the platform is saturated by the additional clones, thus resulting to considerable performance degradation.

Moreover, the system having stationary agents (PSA) to encapsulate the WS functionality, communicate faster with the MA than the system where the MA communicates through SOAP. The better inter-agent communication is attributed to the Grasshopper platform, where agents communicate with synchronous inter-agent message passing.

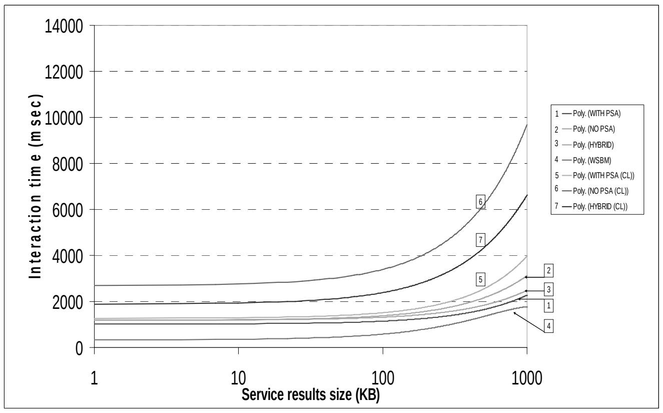

Figure 12: ITSP vs. Service result size

Finally, in Figure 13, we plot the Total Service Time (*TST*) against the service result size. We observe that, apart from the WSBM system that shows the lowest *TST*, the system with the smallest service time is that having PSA and MA cloning enabled (service execution in parallel). We notice that the MA cloning increases the interaction time between the MA and the WS but, eventually, entails considerable improvement to the system, due to the fact that WS are executed in parallel whereas in cases where the MA cloning is not used the WS are executed sequentially.

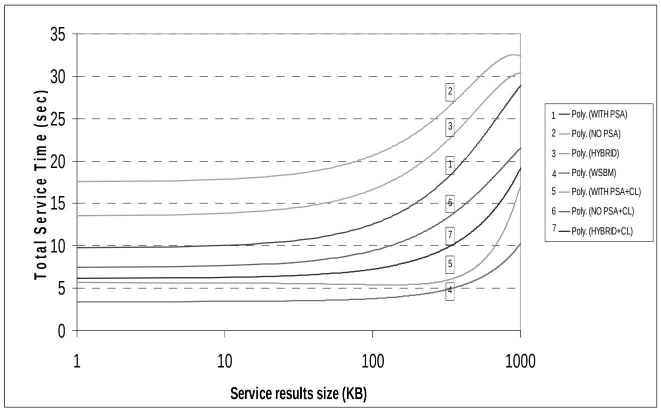

Figure 13. Total Service Time (TST) vs. Service result size

VI. CONCLUSIONS AND FUTURE WORK

In this paper we presented a framework that provides wireless access to WS using MA to find and execute WS in the fixed segment. The WS are semantically enriched and are expressed in OWL-S. Furthermore, the proposed system adopts an enhanced WS registry enriched with semantic information that provides semantic matching between service requests submitted and the service description published to them. The advantages of the system are: (1) Users may invoke a set of services with only one interaction with the fixed network (post the request and receive the results), (2) Users do not have to be connected during the service discovery and invocation and the results are downloaded to mobile devices after their network session re-establishment, (3) Service invocations are performed locally or according to the user's specified policies, and unnecessary information is not transmitted over the network leading to better resource utilization, (4) the framework ensures the delivery of the service results to the user, (5) the MA dynamic behaviour improves system robustness and fault tolerance, (6) New services, agents, users and service registries can be easily integrated to the framework thus providing an expandable, open system.

Future work includes the study of agent mobility. We have already designed a MA infrastructure that takes network events into account. Network events (e.g., node failures) occurring while the service invocation is underway, may force the MA to dynamically reschedule its itinerary accordingly. The MA will implement routing algorithms that generate itineraries by considering network information published in the WS description, network status and topology. Presently, we plan to integrate our framework with SNMP agents to report network monitoring events and develop the necessary agent infrastructure.

REFERENCES

[1] Tim Berners-Lee, James Hendler, Ora Lassila "The Semantic Web" Scientific American 2001

[2] Gruber"A Translation Approach to Portable Ontology Specification"Knowledge Acquisition 5, 1993

[3] http://www.daml.org/servcies/owl-s/

[4] Fuyuki Ishikawa, et al "Mobile Agent System for Web Services Integration in Pervasive Networks", (IWUC 2004), pp.38--47, April, 2004

[5] Fuyuki Ishikawa, et al, "Behavior Descriptions of Mobile Agents for Web Services Integration", ICWS 2004, pp.342--349, July, 2004

[6] Paul A. Buhler, Jose M. Vidal. "Semantic Web Services as Agent Behaviours" In B. Burg et al., editors, *Agentcities: Challenges in Open Agent Environments*, pages 25-31. Springer-Verlag, 2003.

[7] Paul A. Buhler, Jose M. Vidal "Enacting BPEL4WS specified workflows with multiagent systems". Proceedings of the Workshop on Web Services and Agent-Based Engineering, 2004

[8] Paul A. Buhler, et al "Adaptive Workflow = Web services + Agents". International Conference on Web Services, pages 131-137. CSREA Press, 2003

[9] Amir Padovitz, et al "Towards Efficient Selection of Web Services." WSABE2003 Australia 7/2003

[10] Rebecca Montanari et al "A Policy-based Mobile Agent Infrastructure." SAINT'03, 2003: p.370-379

[11] Rebecca Montanari, et al "Policy-based Separation of Concerns for Dynamic Code Mobility Management." COMPSAC'03

[12] Wassam Zahreddine, Qusay H. Mahmoud "An agent-based approach to composite mobile web services" AINA05 p185-188.

[13] Sheng-Tzong Cheng et al "A new framework for Mobile Web Services" (SAINT'02w).

[14] Nicholas Gibbins, et al "Agent based Semantic Web services" Journal of Web Semantics, 2004.

[15] Lagana Kagal et al "Agents making sense of the semantic web", WRAC 2002, USA, January 16-18,

[16] Matchmaker :http://www.daml.ri.cmu.edu/ matchmaker/

[17] Katia Sycara, et al. "Automated discovery, interaction and composition of semantic web services". Journal of Web Semantics, July 2004.

[18] Massimo Paolucci, et al. "Semantic matching of Web services Capabilities" International Semantic Web Conference 2002: 333-347.

[19] Naveen Srinivasan, et al. "Adding OWL-S to UDDI, implementation and throughput", SWSWPC 2004

[20] RACER: www.racer-systems.com

[21] JUDDI: http://ws.apache.org/juddi/

[22] Web service Architecture : http://www.w3.org/TR/ 2004/NOTE-ws-arch-20040211/

[23] Grasshopper: http://www.grasshopper.de (last accessed 10/2003)