Integration of Mobile Agents and Web Services

Jan Peters

Fraunhofer Institut für Graphische Datenverarbeitung
Fraunhoferstraße 5, 64283 Darmstadt, Germany
jan.peters@igd.fraunhofer.de

Abstract. The web service specification represents an open standard for distributed service oriented architectures, already impacting a broad range of commerce and industry. Web services are widely used in current web-based business applications, and form the basis of emerging frameworks in Grid computing. Current mobile agent systems implement similar (often proprietary) mechanisms for service description, invocation and discovery. Furthermore, mobile agents provide very specific advantages with respect to dynamic and concurrent service composition resp. execution, as well as the extension of server-sided services with client-side intelligence and functionality. This short paper introduces an architecture which seamlessly connects mobile agents with web services in a transparent and fully automated manner by means of a specialized Web Service Engine. This Web Service Engine has been implemented and integrated into the Secure Mobile Agents (SeMoA[1]) platform.

1 Introduction

Similar to web services, software agents can encapsulate business or application logic. Rather, software agents can dynamically discover, combine and execute such processes, and further offer multiple services or behaviors, that can be processed concurrently. In order to move from one system to another, or even to communicate with each other, mobile agents currently need a common platform on which they operate. Thus, they are useful for business partners only if these actually share a common platform. The consistent use of web service standards for description of capabilities, communication, and agent discovery would establish interoperability not only between different agent platforms but also between agent platforms and traditional web services. Thus, the advantages of two worlds can be combined.

To integrate mobile agent and web service technology in a seamless manner, components have to be designed, which map between the different mechanisms for *service description*, *service invocation*, and *service discovery*, in both worlds. In other words, messages resp. representations from the according web service protocols (WSDL, SOAP, UDDI) have to be translated into corresponding requests resp. data types of the agent system, and vice versa. When I talk of a *web service engine* in the remainder of this paper, I mean the summary of these components, which can be integrated into an existing agent system, and thereby extend this system with web service abilities (cf. Fig. 1).

[1] SeMoA - Secure Mobile Agents. http://www.semoa.org/

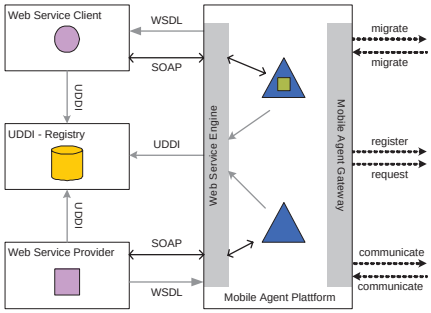

Fig. 1. WSE Integration Architecture

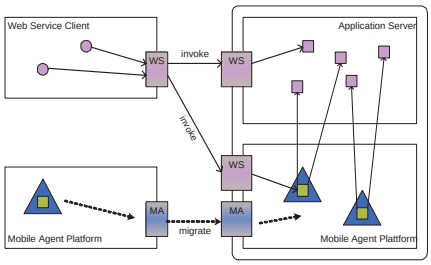

Fig. 2. Example Scenario

This web service engine enables both, agents offering encapsulated functionality as web service (service provider) as well as agents utilizing web services (service client). In a compound scenario, one agent might offer a functionality provided as web service which in turn can be utilized by another agent acting as web service client. Assuming *transparent integration*, as I want to define it, the service as well as the client agent should not recognize any "agent service to web service" resp. "web service to agent service" mappings during the service description, discovery, and invocation process. In other words, they should only need to know and utilize the usual mechanisms of their specific agent environment. Of cause, a traditional external web service client should be able to directly use the interfaces of the web service engine to search for and utilize agent based services as web services.

Both web service and mobile agent technology have different attack scenarios and thus ask for different solutions. The main concern in web service security is to prohibit unauthorized access to resources from (malicious) clients. As we talk of mobile agent based web services, i.e. program code is transported to another host and executed locally within a foreign environment, we must not neglect the corresponding *security aspects.* Within the mobile agent community there has been elaborate discussion about security concerns in the past. A lot of solutions and protocols have been proposed which prevent or at least reveal malicious behavior dependent on the specific application. When integrating web services and mobile agents it has to be assured that the new mean of service interaction does not bypass or override existing security mechanisms of the agent system. In my approach I combine the security mechanism of our SeMoA platform with security mechanisms from web service technology.

2 Example Scenario

Based on such an architecture, many interesting use cases are conceivable. In this section I want to give one concrete example scenario. As shown in Fig. 2, a mobile agent (△) embedding a compound service migrates from its origin to a provider's agent platform, and subsequently accesses services (□) within a locally running application server to fulfill its task. By registering the compound service within the agent platform this service with added value is provided as web service, not only to its owner, but to every

regular web service client (○). This scenario distinguishes the following three entities within a network community:

Provider A *provider* allows access to locally hosted resources through web services, implemented as components within a regular application server. Furthermore, he runs a mobile agent platform which accepts agents from community members and grants them local access to the provided web services.

Community Member The *community member* invests his knowledge to implement a service with added value which is based on functionality of the provider's web services. Embedding this compound service in a mobile agent, and subsequently migrating this agent to the provider's agent platform, it is able to benefit from the optimized access to local services.

Regular client If the community member's agent in turn registers the API for its compound service within the agent platform, this service can subsequently be invoked as web service by the community member and other *regular clients* through the agent platform's web service engine.

In this provider-based network community (e.g. a gaming platform), a community member is able to improve the provider's web service framework for himself and other community members. The community member does not need to setup its own public web service framework to provide his services with added value. The compound service further benefits of local access to resources. The provider is still able to select and encapsulate incoming agents by enforcing its security policy through the agent platform. Thereby, he benefits of community members improving the web service framework he provides, and is in control of the additional features at the same time.

3 Related Work

The work described in this paper has been strongly influenced by resp. is evolved from several existing contributions to the field, whereas the following discusses existing frameworks which build upon static agent systems, and approaches integrating web services and mobile agents.

Most of the frameworks based on *static agent systems* [8, 3, 11, 6] presuppose FIPA-compliant communication between agents, and a FIPA-compliant mechanism for service description and discovery (cf. [5, 4]). Since interaction is based on the exchange of messages, all the approaches lead to components (a generic gateway, or single wrapper agents) which map incoming FIPA-ACL messages and service register/deregister/search requests into messages of the corresponding web service protocol, and vice versa. These components further have to bridge the gap between asynchronous behavior of FIPA-ACL, and synchronous behavior of SOAP-based service invocation. Except the latest and most advanced framework described by Greenwood et al., which aims to implement all mechanisms for bidirectional system integration in one single gateway component, the approaches need manual configuration steps for each integration process, which is partly automated but cannot be done during runtime.

In contrast, the existing contributions based on *mobile agent systems* have more diverse goals, and thereby do not completely follow transparent integration of agents and

web services. While some approaches make use of specific advantages of mobile agents in the field of load balancing [1], or the efficient and concurrent task execution in distributed and mobile environments [9], others focus on dynamic and efficient selection of web services through mobile agents [10]. Instead of integrating web service support into an existent mobile agent system, Cooney et al. extended the .NET framework for web services with the ability of service migration [2] based on a rudimentary agent mobility paradigm. Since they mainly created a new service platform from the scratch, there was barely obligation to map existing mechanisms for service description and invocation to the web service world. The realization of mobile web services described in [7] represents a high level approach of system integration. Due to an interpreter agent encapsulating web service logic, this approach is mostly independent of the underlying mobile agent system and the given interaction paradigms. As consequence, existing services of the underlying system cannot be reused transparently as web services, whereas the developer of mobile web services has a new workflow based programming model to implement composite services.

The above summarized frameworks dealing with mobile agent systems, either enrich the web service world with specific advantages of mobile agent technology, or bind the mobile agent world to web service technology. In contrast, the WSE architecture aims to provide bidirectional integration of both technologies in a seamless way.

4 Architecture

Derived from [6], the following *assumptions* were made when designing the web service engine:

- The agent runtime environment provides mechanisms for service description, invocation, and discovery, which are based on direct (or remote) method invocation in Java resp. sharing of Java-APIs as service interfaces.
- All web services are assumed to use the standard web service stack consisting of WSDL for service description, SOAP for service invocation, and UDDI for service discovery.
- The web service engine is transparently integrated into the agent server by means of an extended service management layer, which *implicitly* transforms locally registered services into web services accessible from external web service clients (and vice versa).
- The web service engine is further visible as service within the agent server, which can *explicitly* be used by other services and agents (to search for and invoke resp. to register web services).
- The web service engine is visible from web service clients as gateway supporting various transport protocols (mainly HTTP and HTTPS). Service endpoints within this gateway are dynamically generated and publish via UDDI.
- If the agent system supports semantic agent service descriptions, these are available to the web service engine.

In case of SeMoA, the only mean of interaction between mobile agents and/or services is based on direct method invocation through shared Java interfaces: Services are

registered locally as entities within a hierarchical namespace (the service *environment*). Thus, a service requester searches for an appropriate service interface within a defined sub hierarchy of the environment. If successful, a Java object is returned which implements the given interface. As consequence no asynchronous protocol behavior has to be mapped to the synchronous behavior of SOAP. Since automated runtime integration of web services is provided, the framework has to cope with optimized and dynamic web service stub provisioning, on the other hand.

The proposed solution to implement the *Web Service Engine* (WSE) as introduced above covers several individual components and a service management layer which can easily be adapted and plugged into the existing service management of the specific mobile agent platform:

Stub Generator SOAP messages transported through the network as result of a web service invocation are typically exchanged between a client and a server stub. With respect to the WSE architecture, the server stub processes incoming messages and triggers the associated service object by means of direct method invocation. Vice versa, the client implements the service's interface and starts communication with the associated server stub, upon local method invocation. Thereby, the task of the *stub generator* is twofold. On the server side, it extracts the specific Java interface from a given service object, automatically generates a corresponding syntactic WSDL description and creates a new server stub, which is then associated with the service object. On the client side, it transforms a given WSDL description into the corresponding Java interface, and creates a client stub implementing this interface. To realize automated and transparent integration for Java-based systems, the stub generator must be able to dynamically generate new stub objects during runtime.

Web Service Gateway Server stubs created by the stub generator have to be exposed by means of web service endpoints accessible over the network. The *web service gateway* thereby implements the specific transport protocols and serves as both, web server enabling access to server stubs (e.g. over HTTP and HTTPS) as well as web client used by client stubs as transport layer for the transmission of SOAP messages.

Registry Service To make agent services visible by means of web service discovery, the *registry service* transforms WSDL descriptions created by the stub generator from a given service object into appropriate UDDI business entities. These business entities are subsequently be registered at a UDDI-compliant registry. Furthermore, this service can be used to search for a web service which is syntactically compatible to a given Java interface.

WSE Service The *WSE Service* wraps the above described functionality and provides it via a simple interface which can *explicitly* be used by mobile agents to either search for web services, or to deploy and undeploy encapsulated service objects. In both cases, the agent does not need to know anything about the traditional web service stack: deployment is done by providing a Java object implementing an arbitrary Java interface which is automatically exposed by means of a web service, then; a search request with a given Java interface directly returns the reference to a client stub implementing this interface, if successful.

Service Management Layer The *service management layer* transparently activates the above described processes by automatically forwarding appropriate requests (to register or lookup an agent service within the agent infrastructure) to the WSE service. Since web service deployment and undeployment is subsequently done implicitly, the administrator of the local agent server can configure this layer in advance, and select the types of services to automatically expose as web services.

Acknowledgements

This paper was written while the author was working within SicAri, a project funded by the German Ministry of Education and Research. The web service engine has been implemented in cooperation with Jan Oetting, usd.de ag, Germany.

References

1. J. Cao, Y. Sun, Y. Wang, and S.K. Das. Scalable Load Balancing on Distributed Web Servers Using Mobile Agents. *Journal on Parallel and Distributed Computing*, 63(10):996–1005, October 2003. ISSN:0743-7315.
2. Dominic Cooney and Paul Roe. Mobile Agents Make for Flexible Web Services. In *Proceedings of The Ninth Australian World Wide Web Conference*, Queensland, Australia, July 2003.
3. Jonathan Dale, Akos Hajnal, Martin Kernland, and Laszlo Zsolt Varga. Integrating Web Services into Agentcities Recommendation. Technical report, Agentcities Task Force, November 2003.
4. FIPA. FIPA Agent Discovery Service Specification. Preliminary FIPA document PC00095A, Version 1.2e, Foundation for Intelligent Physical Agents, October 2003. http://www.fipa.org/specs/fipa00095.
5. FIPA. FIPA Agent Management Specification. Standard FIPA document SC00023K, Foundation for Intelligent Physical Agents, March 2004. http://www.fipa.org/specs/fipa00023.
6. Dominic Greenwood and Monique Calisti. Engineering Web Service - Agent Integration. In *IEEE International Conference on Systems, Man and Cybernetics (SMC 2004)*, The Hague, The Netherlands, October 2004.
7. Fuyuki Ishikawa, Nobukazu Yoshioka, Yasuyuki Tahara, and Shinichi Honiden. Toward Synthesis of Web Services and Mobile Agents. In *Proceedings of the AAMAS'2004 Workshop on Web Services and Agent-based Engineering (WSABE)*, New York, USA, July 2004.
8. M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris. On Software Agents and Web Services: Usage and Design Concepts and Issues. In *The 1st International Workshop on Web Services and Agent-based Engineering*, Sydney, Australia, July 2003.
9. Zakaria Maamar, Quan Z. Sheng, and Boualem Benatallah. Interleaving Web Services Composition and Execution - Using Software Agents and Delegation. In *The 1st International Workshop on Web Services and Agent-based Engineering*, Sydney, Australia, July 2003.
10. Amir Padovitz, Shonali Krishnaswamy, and Seng Wai Loke. Towards Efficient Selection of Web Services. In *The 1st International Workshop on Web Services and Agent-based Engineering*, Sydney, Australia, July 2003.
11. Laszlo Zsolt Varga, Akos Hajnal, and Zsolt Werner. An Agent Based Approach for Migrating Web Services to Semantic Web Services. In *11th International Conference on Artificial Intelligence: Methodology, Systems and Applications (AIMSA 2004)*, Lecture Notes in Computer Science, pages 371–380, Varna, Bulgaria, September 2004. Springer Verlag Heidelberg.