A Service-Oriented approach for integrating MultiAgent Systems with Web Services in a Collaboration application.

Eduardo H. Ramírez and Ramón F. Brena

Tecnológico de Monterrey
Centro de Sistemas Inteligentes
{eduardo.ramirez,ramon.brena}@itesm.mx

Abstract. Service-oriented architectural paradigm (SOA) improves the abstraction and flexibility on which information systems may be designed and integrated, thus allowing agent-based and other AI applications to be gradually and seamlessly adopted by large organizations. Web-services standards for semantics and interoperability realize the SOA approach and are key enablers for MultiAgent systems enterprise integration. In this paper we show how the SOA paradigm and Web Services technologies may be used for designing and integrating agent-based applications in a flexible and robust way. A case-study for a collaborative application is presented.

1 Introduction

The Service-Oriented architecture paradigm (SOA) improves the abstraction and flexibility on which information systems may be designed and integrated. SOA foundation ideas were introduced by Arsajani[2] and sharply defined as: "the architectural style that supports loosely coupled services to enable business flexibility in an interoperable, technology-agnostic manner. SOA consists of a composite set of business-aligned services that support a flexible and dynamically re-configurable end-to-end business processes realization using interface-based service descriptions[5]".

There exists three roles or actors in a SOA that are shown on figure 1. A *service consumer*, that locates a *service provider* in a *service registry* or broker, then binds the service description and finally performs an invocation to the provider. The loose coupling principle suggest the operations between actors to be realized interchanging as little information as possible, usually through message passing.

Even though the SOA architectural style is not bound to any particular implementation technology, the Web Services standards are becoming a natural and common choice. For the purposes of this work, we understand a Web Service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [24]). Other systems interact with the Web Service

© A. Gelbukh, R. Monroy. (Eds.)
Advances in Artificial Intelligence Theory
Research on Computing Science 16, 2005, pp. 107-116

in a manner prescribed by its description using SOAP-messages [25], typically conveyed using HTTP with an XML [23] serialization in conjunction with other Web-related standards."[26].

Close relationships between Web Services and Software agents[14] exist at the semantic and interoperability level. Since the conception of the Semantic Web[4] due to its semantic processing capabilities, agent technologies were strongly proposed as Intelligent Web Service consumers[10], being able to discover, compose and verify Web Services[27]. On the other side, agents are also a candidate technology to deliver knowledge-intensive Web Services[1][11]. However, in order to allow agents to become intelligent Web Service providers, a number of technical issues should be addressed.

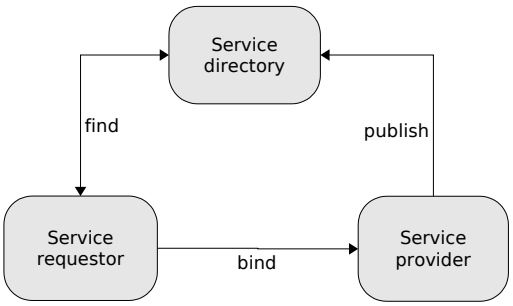


Fig. 1. Roles in a service-oriented architecture

2 SOA for MultiAgent Systems

The SOA approach and Web Services technologies could be used to overcome the technical limitations required to integrate a MultiAgent System into an enterprise service-oriented environment. In a similar way, the SOA ideas can be translated to the application internals to allow agents to provide a stable framework for building robust Agent-based applications. A number of recent works report how the SOA approach have been applied in several AI and multi-agent systems and components[17][19][27].

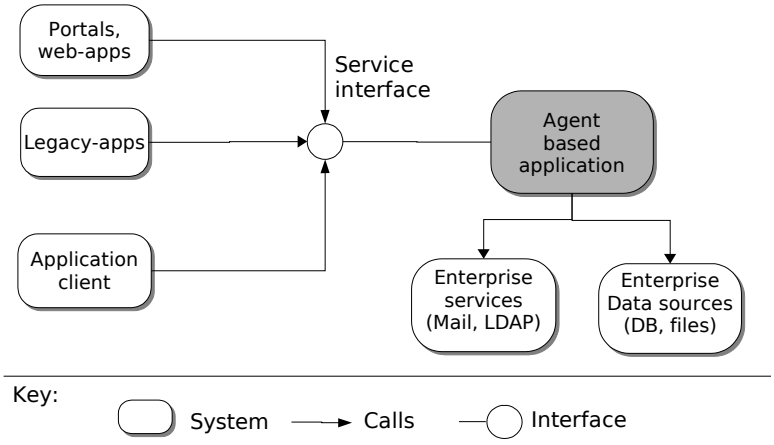


Fig. 2. Decoupled architecture top-level view

2.1 SOA for agent-based application integration

In conformance with the SOA ideas, the main architectural principle for integrating a MultiAgent system into an enterprise environment consists of decoupling applications through the exposure of "coarse-grained" service interfaces. As shown in figure 2, the underliying metaphor used to determine the design strategy was the aim to create a "black-box" in which agents can live and perform complex knowledge-intensive tasks. Neither enterprise applications nor its developers should be aware that a service is provided by agents if the system offers a standard SOAP endpoint as interface.

Several integration architectures for agent-based application exists[22],[7], however a particular approach, namely the "Embedded Web Service Architecture (EWSA)" has proven to be enough simple and effective for this purpose with a clear focus on agent-to-application communication. The proposal's main idea suggests embedding a Web Server into an agent-based application and defining a interaction model upon which software agents and Web-components[20] may communicate in order to provide services. Details on the approach can be found on related work[18].

Once the integration problem is solved, the task of designing an adequate set of services is a classic software engineering problem and remains a particular challenging activity by itself. Each agent-based application, may offer a different set of agent-based web services. Some examples of knowledge-intensive services from one of our agent projects[6] are:

Recommendation services A user's profile is represented by a set of points in the taxonomies, as each user could have many interests and could be located

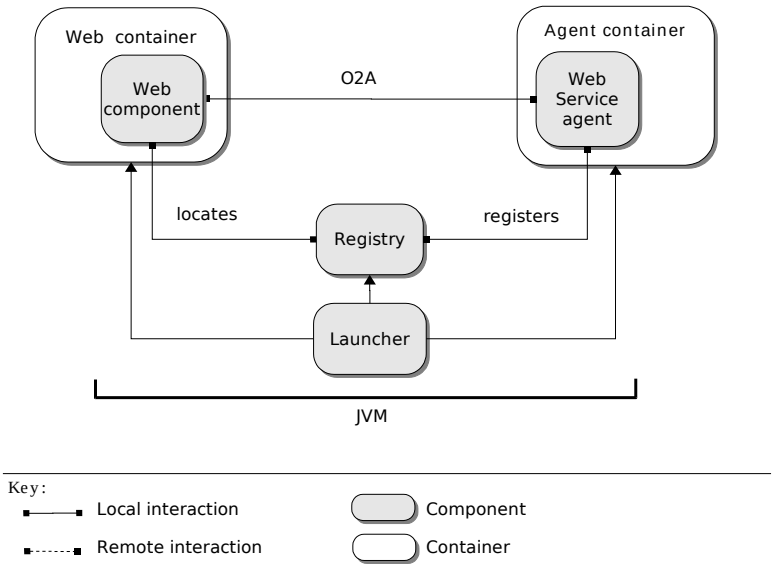


Fig. 3. Integration architecture component view

at different parts of the organizational structure. As the system keeps track of user interests and preferences it is able to recommend content to users on demand. Recommended content may be used in Portals or Web applications.

Content search and classification A service agent that searches the most relevant documents on particular data-sources can be constructed. The knowledge that guides the search is handled by the ontology agent where the keywords with which the search engine is invoked are defined. The documents obtained by the search are qualified by a fuzzy system and then the best ones are delivered to the users.

Subscription and alert services The system allows users to subscribe to changes in specific areas. Also, users may customize the media and frequency of notifications. Rules may be defined so as messages relative to certain topics are handled with higher priorities. A rule may state that several alerts may be sent to their cell-phone via SMS, and also define that interest-area messages be sent in a weekly summary via email.

Content distribution services Enterprise applications may deliver content to the system using its semantic-based content distribution services. When new content is received it is classified and distributed to users who may be interested. Users receive the notifications of new content as specified by their own rules.

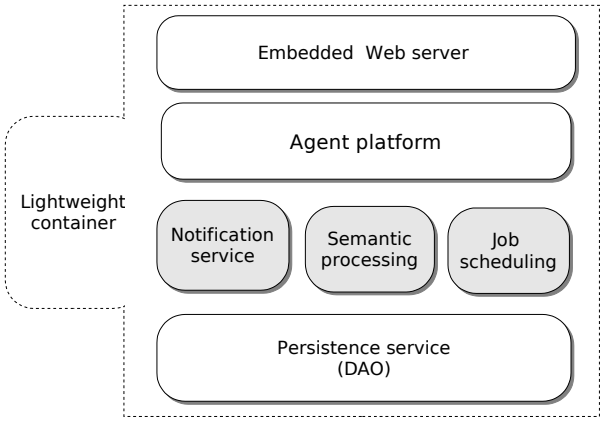


Fig. 4. Platform services framework, layered component view

2.2 A Service-Oriented framework for MultiAgent System design

Besides application integration, the service-oriented approach have gained momentum for intra-application component communication. The SOA paradigm have been reinforced by a new generation of non-invasive lightweight frameworks[16],[15] that leverage contemporary software design principles such and allow application developers to produce enterprise-level, yet simple applications. The use of a lightweight framework like Spring[12] helps agent project teams to decompose application in simple components that access and reuse a collection of platform services, thus simplifying programming and allowing a greater abstraction to the specific agent development.

A generic application architecture is shown in figure 4, where some general-purpose services may be provided by robust third-party tools. Each of the boxes in figure4 represents a platform service that is configured and instantiated by the lightweight container, the container also fulfills the "service registry" role described in section 1. Each platform service is composed of a particular kind of application components concerning a specific technical domain. The layer metaphor gives an insight about the abstraction level of each application tier, where high-level service components (i.e. web-components and agents), consume the services of low-level components (i.e. data-access objects).

In the presented example (figure 4), there exists generic (Web, Agent, Persistence) and custom services (shaded) like the Scheduling and Notification services. For this specific case they were defined and implemented as follows:

Embedded Web Server . Implemented using the Tomcat Web Server[13], it provides application with the capability of receiving HTTP requests and turning them to a particular Web-component or Servlet in conformance with the JSR-154[20] specification.

Agent platform . The execution environment for the agents provided by the agent platform in conformity with FIPA[8] specifications. Particularly implemented with the JADE[3] platform.

Scheduling service . Allows the agents and users of the applications to schedule of periodic or time-triggered execution of custom jobs using the Quartz Scheduler[21]. Job components are defined using an application specific hierarchy of tasks and events (i.e. CheckMail, CheckAppointments, StartWorkflow etc.).

Notification service . Allows the agents and web-components to access the enterprise notification infrastructure, namely the mail service using the JavaMail API and several SMS systems. Notification components are the drivers for the distinct enterprise notification providers.

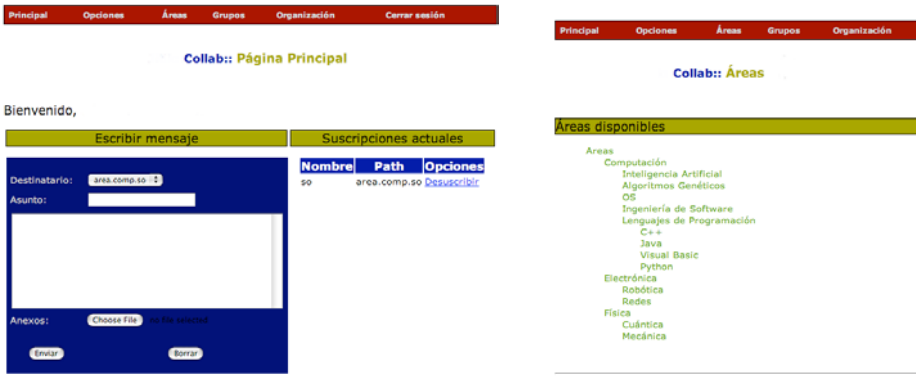
Semantic processing service . Provides access to a set of classification and inference tools like the Jess rule engine[9]. Allows agents to perform simplified queries to rule bases using native Java calls.

Persistence service . Implemented using the JDBC API and the components in conformance with DAO pattern, it is a low-level platform service that provides a simplified object access to enterprise data-sources to the servlets, agents, jobs and other components in the application. It also offers access to directory services such as LDAP for user authentication and profile synchronization.

Following this approach, the particular agent-model of the system is designed and implemented on the Agent platform, enabling agents with the capability of programatically invoking any of the underliying platform services. We believe that the use of such a framework becomes critical in agent projects where the agent developers don't share the same skill-set and concerns of enterprise application developers, then, the use of service-oriented principles (loose coupling, coarse grained interfaces) helps heterogeneous development teams to achieve a clear division of work. One major drawback of the model is that imposes some learning curve to the development teams which could lead to delays in the initial development cycles. Even tough the individual component developer learning scope gets reduced, all developers should have a clear understanding of the overall system architecture beyond de agent model, and of the libraries and interfaces that should be reused in order to interoperate with the underliying infrastructure services.

3 A collaborative application case study

Web-applications are widespread tools for collaboration support in large organizations, however, its traditional style of development and architecture hardly allows for a more sophisticated intelligent service to be included. For this kind of cases, the SOA-integrated agent system may be a knowledge service provider. A prototype groupware application was developed including the following standard sets of features:



(a) Message and file sharing (b) Group browsing

Fig. 5. Collaborative application prototype

- Users may create groups and join to existing groups. See figure 5(b).
- Users may browse the member list and conversations history of each group.
- Within groups users may exchange messages and share files. See figure 5(a).

3.1 Implementation of a recommendation service

Trivial features of a collaborative application could be enhanced by agent-based web services, for example, consider the simple scenario in which a user joins a workgroup using the collaborative application interface. After this event, the system will send recommendations about documents related to group activity and interests on a daily basis. Some of the internal services interaction needed to fulfill this case are:

- Once the user joins the group, the collaborative application will invoke the agent-based service using a standard coarse-grained interface.
- Periodical updates to the document base will be programmed to be triggered upon certain conditions using the agent-platform scheduling service.
- Information gathered from data-sources and repositories will be classified and filtered against the interests of the relevant groups using the semantic-processing and classification services.
- Relevant information will be distributed to users using the agent-based notification services, the system may leverage the use of several distribution media (Email, SMS) to alert users of new documents and other application events according to its relevance.

The components and services interactions for the use-case are shown in detail in figure 6. It should be noticed that the use of generic services for Scheduling, Notification and Classification allows the task-specific code (read documents,

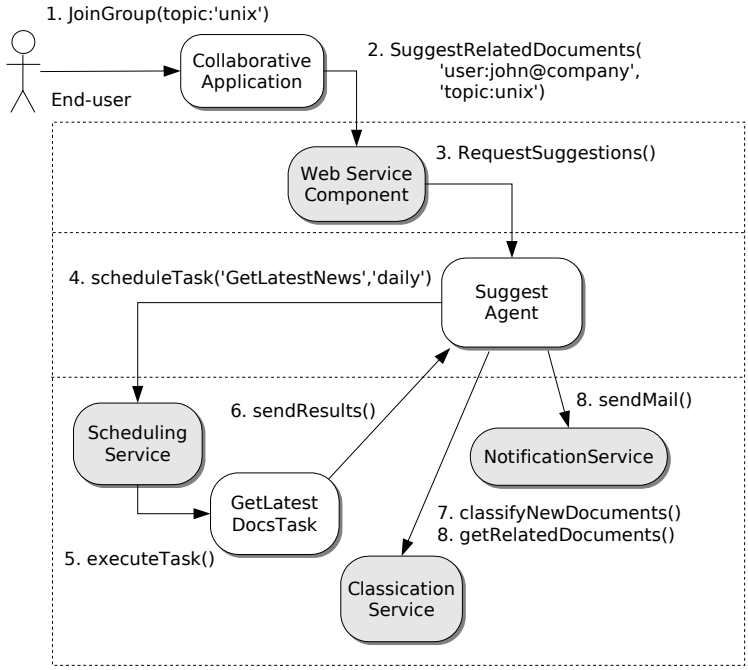


Fig. 6. Use case enhancement using agent based services, detailed view

send mail) to be encapsulated in the lower layers of the system, thus enhancing the abstraction level for the agent layer.

4 Conclusions

In this paper we have shown how the SOA paradigm and Web Services technologies may be used for designing and integrating agent-based applications as service providers in enterprise software environments. Also, using the same design principles we have described a generic framework for intra-platform services for implementing robust and reliable MultiAgent systems reusing high-quality open source components like Spring[12], Jade[3], Quartz[21] and Tomcat[13]. Also we have presented a case study on a simple collaborative application that shows the kind of feature improvements that can be done by the integration of agent-based web services.

Future works we intend to do on collaborative features enhanced by agent-based web services are:

– Taxonomy-based user profiles: User profiles are defined using several standard taxonomies like the organizational structure roles (which are assigned to user by management); workgroups that may be freely created upon project

requirements; and interest areas to which users get subscribed according their professional or personal interests
- User-defined messages: Users of a workgroup or a content area may send custom messages. The message receivers are not selected by name rather than by their role in the organization or their interest areas. A semantic matching of message receivers may be produced using the web services.
- Semantic file sharing: Users of the system may share documents to taxonomy-based workgroups via the web-based interfaces. When a document is uploaded or updated, users subscribed to taxonomy child nodes will be notified according to their preferences. Example: A Unix tutorial is uploaded, so users in the general Computing area, the more specific Operating Systems sub-area, and the specific Unix area are notified of the addition. Also, when a new file is submitted, it may be classified and automatically distributed to a number of relevant groups outside the original receiver.

References

1. Alun Preece and Stefan Decker. Intelligent web services. *IEEE Intelligent Systems*, 17(1), Ene-Feb 2002. `http://www.csd.abdn.ac.uk/~apreece/research/download/ieeeis2002.pdf`.
2. A. Arsanjani. A domain-language approach to designing dynamic enterprise component-based architectures to support business services. In *39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39*, pages 130 – 141, Aug. 2001.
3. Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - A FIPA-compliant agent framework, `http://jade.cselt.it/papers/PAAM.pdf`. In *Proceedings of PAAM'99, London*, 1999.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):35–43, 2001. Essay about the possibilities of the semantic web.
5. Bernhard Borges and Kerrie Holley and Ali Arsanjani. Service-Oriented Architecture, Aug. 2004. `http://webservices.sys-con.com/read/46175.htm`.
6. R. Brena and J. Aguirre. Just-in-Time Knowledge Flow for Distributed Organizations using agents technology. In *Knowledge Technologies 2001 Conference, Austin Texas*, Mar 2001.
7. Dick Cowan and Martin Griss et al. Making Software Agent Technology available to Enterprise Applications, `http://www.hpl.hp.com/techreports/2002/HPL-2002-211.pdf`. Technical Report HPL-2002-211, HP Labs Technical Reports, 2002.
8. Foundation for Intelligent Physical Agents. FIPA Abstract Architecture Specification, `http://www.fipa.org/specs/fipa00001/`, 2002.
9. Ernest Friedman-Hill. Jess, the rule engine for the java platform, 2005. `http://herzberg.ca.sandia.gov/jess/`.
10. J. Hendler. Agents and the Semantic Web, `http://citeseer.ist.psu.edu/hendler01agents.html`. *IEEE Intelligent Systems*, 16(2), 2001.
11. Michael Hunhs. Agents as Web Services. *IEEE Internet Computing*, 6(4):93 –95, Jul-Ago 2002.
12. Interface21 Limited. The Spring Java/J2EE application framework, `http://www.springframework.org/`, 2005.

13. Jakarta Project - The Apache Software Foundation. The Tomcat Web Server v. 4.1, http://jakarta.apache.org/tomcat, 2003.
14. N. Jennings and M. Wooldridge. Software agents. *IEE Review*, 42(1):17 –20, 18 Jan. 1996.
15. Rod Johnson. *Expert One-on-One J2EE Design and Development*. Wrox Press Ltd., Birmingham, UK, UK, 2002.
16. Rod Johnson. J2EE development frameworks. *Computer*, 38(1):107–110, 2005.
17. G.F. Laforga and R.F. Romero. A services-oriented architecture applied to artificial neural network. In *9th International Conference on Neural Information Processing, ICONIP '02*, volume 5, pages 2650 – 2654, Nov, 2002.
18. E. Ramirez and R. Brena. Web-enabling MultiAgent Systems. *Lecture Notes in Computer Science, IBERAMIA 2004*, 3315, 2004.
19. H. Schuschel and M. Weske. Automated planning in a service-oriented architecture. In *13th International IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2004*, pages 75–8, June 2004.
20. Sun Microsystems, Inc. JSR-000154 Java(TM) Servlet 2.4 Specification (Final release), http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html, 2003.
21. The OpenSymphony Group. Quartz Enterpise Job Scheduler, http://www.opensymphony.com/quartz/, 2005.
22. Whitestein Technologies, A.G. Web Services Agent Integration Project, http://wsai.sourceforge.net, 2003.
23. World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3c.org/TR/2000/REC-xml-20001006, 2000.
24. World Wide Web Consortium (W3C). Web Services Description Language (WSDL) 1.1, http://www.w3c.org/TR/wsdl, 2001.
25. World Wide Web Consortium (W3C). Simple Object Access Protocol, http://www.w3.org/TR/SOAP, 2003.
26. World Wide Web Consortium (W3C). Web Services Glossary, Working Draft, http://www.w3c.org/TR/ws-gloss/, Aug 2003.
27. S.J.H. Yang, B.C.W. Lan, and Jen-Yao Chung. A New Approach for Context Aware SOA. In *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE '05*, pages 438 – 443, March 2005.