Multi-Agent Systems Integration in Enterprise Environments Using Web Services

Article *in* International Journal of Intelligent Information Technologies · July 2006

DOI: 10.4018/jiit.2006070105 · Source: DBLP

2 authors:

Eduardo H. Ramírez
Ensitech de Mexico
19 PUBLICATIONS **120** CITATIONS
SEE PROFILE

Ramon Brena
Instituto Tecnológico de Sonora
120 PUBLICATIONS **1,631** CITATIONS
SEE PROFILE

Multi-Agent Systems Intergration in Enterprise Environments Using Web Services

Eduardo H. Ramírez, Tecnológico de Monterrey, México
Ramón F. Brena, Tecnológico de Monterrey, México

ABSTRACT

In this article, we present a decoupled architectural approach that allows Software Agents to interoperate with enterprise systems using Web Services. The solution leverages existing technologies and standards in order to reduce the time-to-market and increase the adoption of agent-based applications. We present case studies of applications that have been enhanced by our proposal.[1]

Keywords: software agents; Web Services

INTRODUCTION

Software Agents (Jennings & Wooldridge, 1996) and Web Services (W3C, 2003b) have become key research areas for a growing number of organizations and are expected to bring a new generation of complex distributed software systems (Jennings, 2000). Even if agent technology is finding its way little by little into the mainstream, Web Services have been adopted much more widely and rapidly (Barry, 2003).

Several authors have pointed out some overlapping areas between agents and Web Services' semantic capabilities (Hunhs, 2002; Preece & Decker, 2002). However, issues regarding how they may be competing or complementary technologies remain open (Petrie, 1996). Because of that, research involving agents and Web Services is focused mainly on building improved semantics (Dickinson & Wooldridge, 2003; Hendler, 2001), communication languages and interaction protocols (Labrou, Finin, & Peng, 1999).

We assume that in order to impact real-world organizations, a greater emphasis should be made on interoperability between agent-based applications and enterprise information systems. Moreover, we believe that the adoption of agent technologies will grow by leveraging existing industry standards and technologies. Therefore, the problem we address is an instance of the legacy software integration problem (Nwana & Ndumu, 1999; Genesereth & Ketchpel, 1994).

In this work, we present a decoupled architectural approach and design principles called Embedded Web Services Architecture (EWSA) that allows agent-based applications to be integrated into enterprise application environments (Peng et al., 1998) using Web services, thus allowing them to interoperate with robust conventional systems such as the following:

- Web Applications, Portals, and Content Management Systems (CMSs)
- Enterprise Resource Planning (ERP)
- Manufacturing Execution Systems (MES)
- Workflow Engines and Business Process Management Systems (BPMSs)

This integration allows agents to publish XML (W3C, 2000) Web Services (W3C, 2003b) or standard HTML, thus providing a convenient interface for other distributed components. The Web Service architecture is understood widely 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) (W3C, 2001). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages (W3C, 2003a), typically conveyed using HTTP with an XML (W3C, 2000) serialization in conjunction with other Web-related standards (W3C, 2003b).

Regarding the external behavior of the agent-based application, our approach fits into the design paradigm identified as Service Oriented Architecture (SOA). The SOA foundation ideas were introduced by Arsajani (2001); he defines SOA 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 flexible and dynamically reconfigurable end-to-end business processes realization using interface-based service descriptions (Borges, Holley, & Arsanjani, 2004).

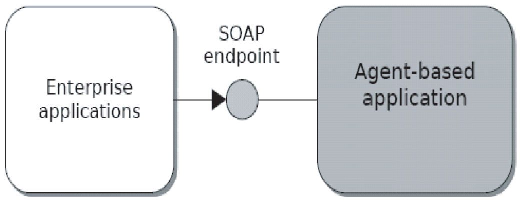
This article also discusses the kind of agent-based applications we have found to be suitable for this approach and the nature of Web Services that agents can provide. The rest of the article is organized as follows. In the next section, we provide an overview of our solution approach. Specifically, we discuss the proposed Embedded Web Server Architecture for integrating agent-based applications and enterprise applications and its implementation. Then, we discuss its evaluation and application to some example domains. Finally, the article concludes with some expected business results.

SOLUTION OVERVIEW

Instead of making an agent-based application look different, compared to other applications from the outside, which is, indeed, a “religious” point of view frequently in the agent research community, we intend to hide the agentness of a group of agents from the outside.

We contend that agents should solve problems for which they are well suited and

Figure 1. Decoupled architecture top-level view



should relate to other software components just as another software component. This is especially true when a set of technologies for gluing software components is maturing, such as Web Services and Service-Oriented Architectures. So, our approach relies much more on hiding the agents than on exposing them to the outside world.

Architecture

As shown in Figure 1, the underlying metaphor used to determine the design strategy is the aim to create a black box in which agents can live and perform complex tasks. The main architectural principle consists of decoupling agent-based applications through the exposure of Web Service interfaces. Enterprise applications should not be aware that a service is provided by agents if the system offers a standard SOAP endpoint as interface, appearing to the world as a conventional Web Service or application.

An agent-based application that exposes Web interfaces requires the interoperability of Web components and agents and their respective containers, since they are built on different programming models, each following different sets of specifications. The relevant components and containers in a system combining agents and servlets would be the following:

Web Container. Also called the Servlet container, it is the application that provides the execution environment for the Web components and implements the Java Servlet API in conformity with the JSR-154 specification (Sun Microsystems, Inc., 2003). Web containers usually are built within Web Servers and provide network services related to HTTP request processing.

Web Component. Servlets are the standard user-defined Web components written in Java. JSR-154 defines them as a "Java technology-based web component, managed by a container that generates dynamic content" (Sun Microsystems, Inc., 2003). They follow a synchronous processing model, since they are designed to handle the content of the HTTP requests. The dynamic content delivered in the request may be HTML for Web pages or XML (W3C, 2000) for Web Services.

Agent Container. The execution environment for the agents provided by the agent platform in conformity with FIPA (2002) specifications.

Web Service Agent. A Java thread that periodically executes a set of behaviors containing the agent tasks. For the purposes of this work, we could say that an agent is a Web Service agent, if it receives and processes requests formulated by a human user or an application in collaboration with a Web component. The requests may be synchronous or asynchronous.

Our proposed solution (Figure 2(a)) is designed around the idea of embedding a Web container into the agent-based application. This approach makes it easier to communicate with both containers, because they are placed in the same memory space.

When the agent and Web containers are started on the same Java Virtual Machine operating system process, agents and Web components may communicate by sharing object references in a virtual object-to-agent (O2A) channel. The resulting execution model is shown in Figure 2(b).

As an intermediate result, the Embedded Web Server Architecture (EWSA) provides intra-container integration and allows the agent-based application to process HTTP petitions in a simple and efficient way.

However, in order to leverage the achieved integration, it is necessary to provide some additional artifacts to simplify the interactions between the agents and the Web components; namely:

- An agent registry, where all the agents with Web Services capabilities register in order to be located for Web component invocations.
- A platform launcher, a boot system that initializes and configures the main containers.

Moreover, from the structural view of the architecture as well as the behavioral or dynamic perspective, we notice that when the agents and Web components exist in the same space, synchronization becomes an issue. So, in the embedded architecture, the complexity of interactions is handled by the following two core framework components:

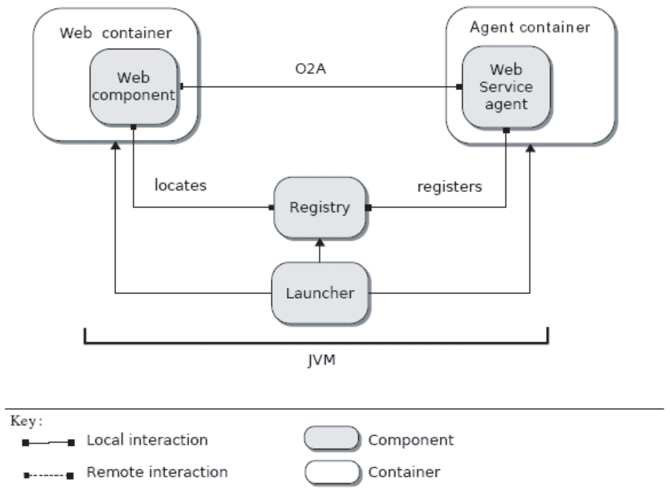
- A call monitor object that serves as a shared memory space between the agent and the Web component and handles synchronization among them.
- A service behavior object used by agents for exposing internal tasks as Web Services.

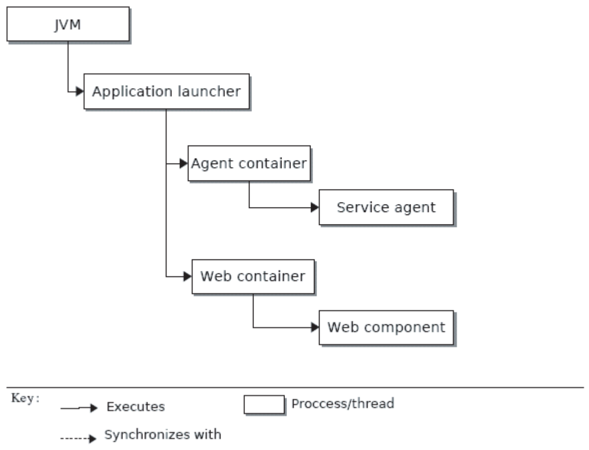
The call monitor is a simple implementation of the classic synchronization construct (Hoare, 1974) used to encapsulate the concurrent access to shared resources. In the EWSA framework, when a Web component requires an agent service, a call monitor object is created to handle the agent response, which may be synchronous or asynchronous.

When a synchronous Web service is invoked, a Web component attempts to access the agent results (the shared resource, in this case) calling the monitor entry method getResult, then the monitor makes the caller wait until the agent thread finishes its work and releases the monitor through the exit method setResult. In the case of asynchronous invocations to agent services, the call monitor releases the Web

Figure 2. EWSA decoupled architecture

(a) Component view





(b) Execution model

component process immediately. As the synchronization may involve a busy wait time, the monitor optionally can handle a maximum invocation timeout. Details of these interactions are shown in Figure 4(a).

On the other hand, as the agent request processing is asynchronous by nature, the service behavior allows agents to unqueue received message objects (each containing a monitor) and to transparently translate them into internal service method invocations. In the case of synchronous requests; that is, when the Web component remains blocked until agent results are calculated, the service behavior object is responsible of delivering them to the call monitor object and thus releasing the lock. The service behavior object also implements a round-robin service discipline within the agent, allowing it to serve many Web component requests simultaneously.

Implementation

Among FIPA platforms, JADE (Bellifemine, Poggi, & Rimassa, 1999) was selected because it is considered well-suited for large-scale deployments, mainly due to its thread-per-agent programming model and the support of virtual channels that allow agents to interact with regular Java components (Rimassa, 2003).

In this particular implementation, the Launcher program initializes and starts an instance of the JADE platform besides an embedded version of the Tomcat Web Server (Jakarta Project, 2003). The aforementioned Registry is nothing but a data structure that holds references to the running Service Agents, implemented as a Singleton pattern (Gamma, Helm, Johnson, & Vlissides, 1995).

Access to the agent's source code is required, since the agent needs to be recompiled to include the Web Service capability, which is encapsulated in a platform-specific library. In JADE's particular case, agents are enhanced with a custom behavior class, which only requires the addition of one line of code.

The architecture is applicable to FIPA platforms other than Jade; however, it would be necessary to port the framework components (Registry and Launcher) using its particular libraries and program interfaces and to add missing components provided by the platform like virtual channels between objects and agents.

Evaluation and Comparison

Our proposal is not the first solution that allows agents to interoperate with Web-based components. In fact, such an architecture was defined by developers (Berre & Fourdrinoy, 2002) of the Jade platform and later implemented on the WSAI Project (Whitestein Technologies, 2003) as a contribution to AgentCities initiative (Dale, Willmott, & Burg, 2002).

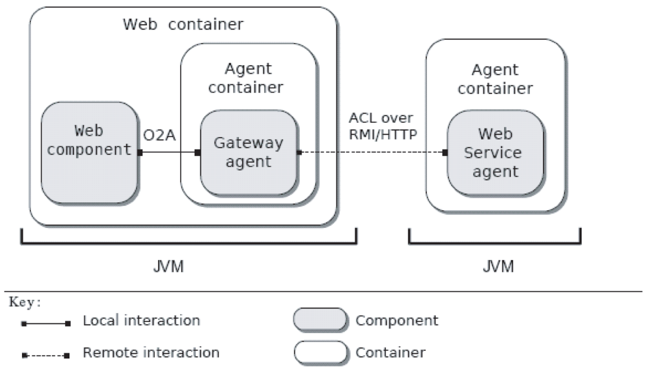
The WSAI solution assumes the existence of two agent containers: one standalone, which we call the main container, and another one contained within the Web container. Each container is executed in a separate JVM system process. WSAI introduces the concept of Gateway Agent as an agent living in the Web container, which is responsible for translating HTTP requests into ACL messages. The general gateway architecture components are shown in Figure 3(a).

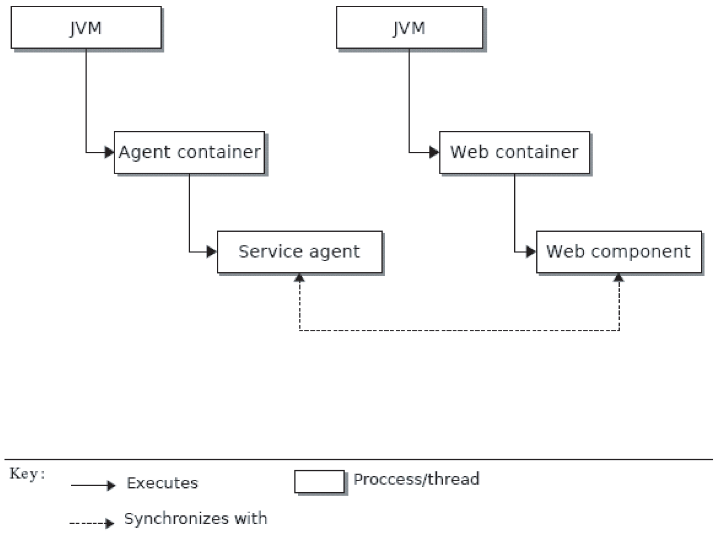
One of the major drawbacks of the approach resides in the existence of several processes that should synchronize using remote method invocations, even if both of them are deployed on the same machine. The synchronization problem is addressed by instantiating the Gateway Agents and Caller objects on a request basis. Callers implement the Java synchronization and time out handling, as shown in Figure 4(a). Additional complexity comes from the fact that it is able to interoperate with any running FIPA-compliant agent platform (even non-Java-based ones) without access to its source code.

An alternative interesting proposal by Cowan and Griss (2002) is BlueJADE, a connector that allows the management of a JADE platform by a Java application server and, therefore, its interaction with the enterprise applications deployed inside it. BlueJADE's main strengths are manifested at the container level. Although the containers are running separately, the connector eliminates the need for remote calls, which

Figure 3. Gateway architecture

(a) Component view





(b) Execution model

enables the use of the object-agent channels. However, BlueJADE does not define any interaction model for the components inside the application and is highly dependent on the particular application server product running the applications.

We believe that in order to build enterprise-class, agent-based applications, it is not critical to provide Web interoperability to an indefinite number of FIPA platforms. Therefore, we trade off this flexibility in favor of good integration with the chosen agent platform, even though the architectural principles remain useful for them. As a result, our framework implementation is simple and provides good performance.

In the EWSA framework, interactions are reduced to a minimal subset, which does not require any remote method invocation. The model shown in Figure 4(b) considers interactions similar to the ones in the gateway architecture, until the HTTP request is delivered to the Gateway Agent.

In a benchmark comparison between WSAI and the EWSA decoupled architecture, an important performance and scalability gain was observed. A currency exchange Web service is provided in the WSAI platform. The service implementation is trivial, since it only consists of a simple mathematical conversion performed by an agent. As shown in Figure 5(a), we may notice that not only are EWSA's response times better, they also increase at a slower rate with respect to the number of concurrent requests, which leads to better scalability. The performance gain in the embedded architecture can be interpreted as an effect of the elimination of network calls overhead between agents and Web components.

A complementary experiment was performed in order to measure the relative performance overhead of the agent-based architecture against a non-agent-based solution implemented as a regular Axis Web Service. Results shown in Figure 5(b) indicate that by using the embedded architecture, the performance of the agent system follows a behavior pattern close to that of the pure Java solution. As a preliminary conclusion, we may state that the embedded architecture solves the potential performance limitations for using agents in simple scenarios.

Additionally, by means of simple metrics, we could show some of the desirable properties of the proposed architecture. As seen in Table 1, a functional implementation for embedding Web Services into agent-based applications is several times smaller than the implementation of a gateway framework from the WSAI Project.

Although the number of classes is not an absolute metric of software complexity, from the interaction models, we can deduce that the interactions present in the embedded model (Figure 4(b)) are nothing but a subset of the interactions required in a gateway approach (Figure 4(a)). Consequently, the internal complexity of the embedded Web Services Architecture (EWSA) is not significantly different from the gateway architecture. In the embedded architecture, only one abstract class is provided as an extension point for service developers, which consequently simplifies the agent and Web Service implementation, as shown in Table 2.

This comparison is provided as a reference resulting from the programming models. Considering that even if a code generation tool actually could simplify the development process, the additional interactions remain with a significant performance penalization.

APPLICATIONS

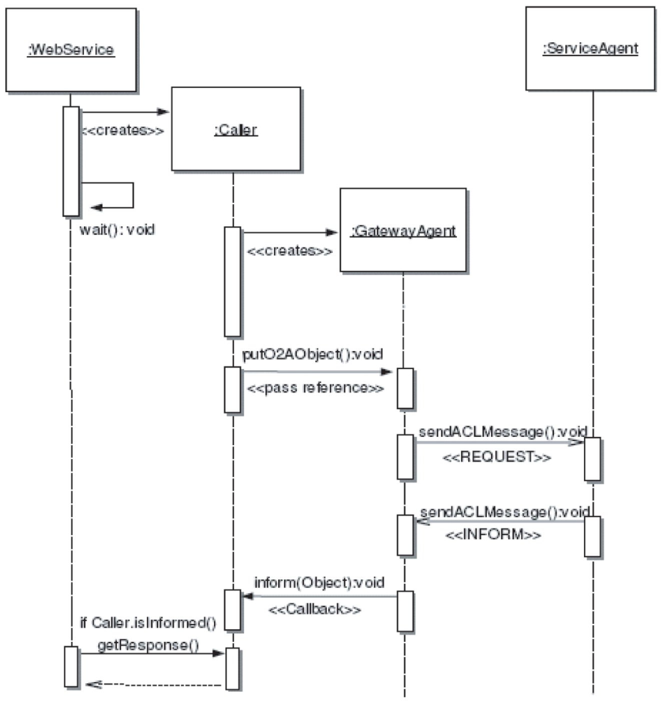
In general, we believe that the proposed integration model is useful in order to allow agent-based applications to provide knowledge-intensive services, such as the following:

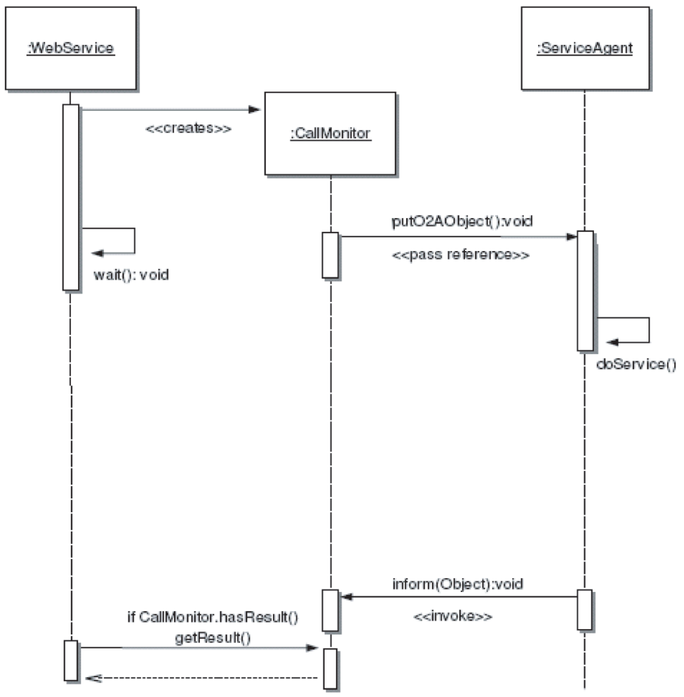
- Search and automatic classification
- User profile inference
- Semantic-based content distribution

Web-enabled agent systems may serve in a variety of domains. As presented in the Just-in-Time Information and Knowledge (JITIK) case study, they are well suited to support knowledge distribution in enterprise environments.

Figure 4. Interactions sequence diagrams

(a) WSAI interactions

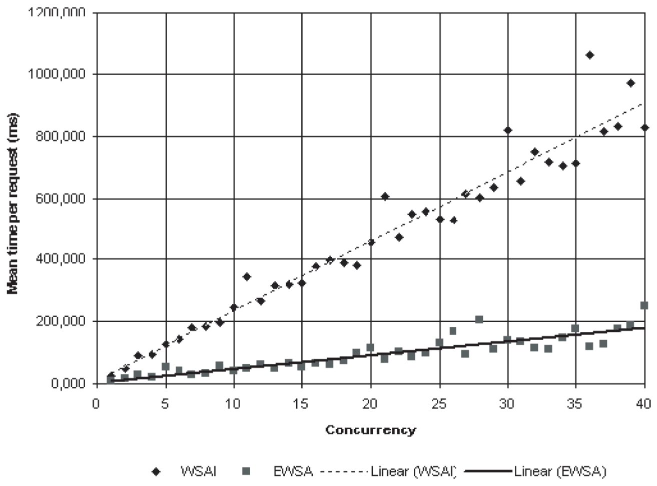


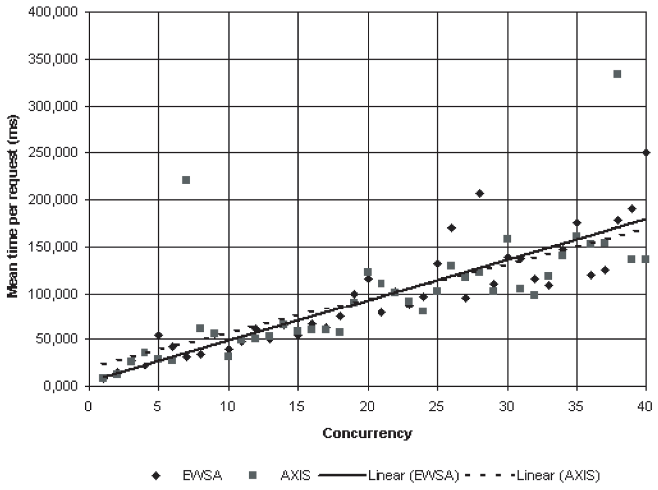


(b) EWSA interactions

Figure 5. Mean service time for concurrent requests

(a) EWSA vs. WSAI





(b) EWSA vs. non-agent service

Just-in-Time Information and Knowledge

The proposed model has been implemented successfully for the JITIK environment that may be defined as a Web-enabled, agent-based intelligent system capable of delivering highly customized notifications to users in large distributed organizations (Brena et al., 2001). JITIK is aimed to support collaboration within organizations by delivering the right knowledge and information to the appropriate people just in

Table 1. Integration framework size comparison

Package	Total Classes	Abstract	Concrete
Embedded	6	1	5
Gateway	52	6	46

Table 2. Example implementation size comparison

Package	Total Classes	Abstract	Concrete
Embedded	2	0	2
Gateway	4	1	3

time. JITIK is designed to interoperate with enterprise systems in order to retrieve and distribute contents in a flexible way.

The JITIK agent model is shown in Figure 6. Personal Agents work on behalf of the members of the organization. They filter and deliver useful content according to user preferences. Personal agents are provided with information by the Site Agent, who acts as a broker between them and service agents. For the purposes of this work, the most relevant agents of JITIK are the so-called service agents that collect and detect information and knowledge pieces that are supposed to be relevant for someone in the organization. Examples of service agents are the Web Service agents that receive and process external requests as well as monitor agents that continuously are monitoring sources of information and knowledge (Web pages, databases, etc.).

The ontology agent contains the knowledge about the interest areas of the members of the organization and about its structure (Brena & Ceballos, 2004). This knowledge is described hierarchically in the form of taxonomies, usually one for interest areas and one describing the structure of the organization. For example, in an academic institution, the interest areas could be the science domains in which the institution is specialized and the organizational chart of the institution that gives the structure of the organization.

JITIK Web Services

JITIK is an example of an agent-based application that is able to provide knowledge-intensive services that may be grouped as follows:

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 at different parts of the organizational structure. As JITIK 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.

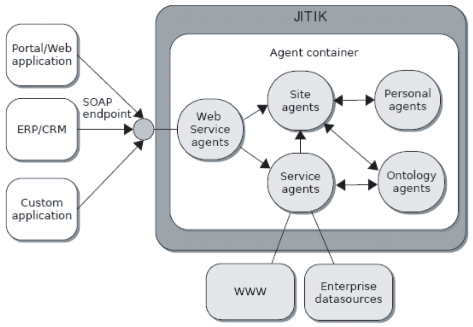
Content Search and Classification. One of the main difficulties for Web users is obtaining relevant information. In normal conditions, people waste a lot of time searching for documents on the Web, because the user must examine the documents in detail to determine if they are really relevant for the search purpose. In the context of JITIK, a service agent that searches the most relevant documents on the Web can be constructed. The knowledge that guides the search is handled by the ontology agent in which 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 presented to the user.

Subscription Services. JITIK allows users to subscribe to changes in specific areas. Also, users may customize the media and frequency of JITIK notifications using simple Web-based interfaces. Rules may be defined so that messages relative to certain topics are handled with higher priorities. A rule may state that several alerts may be sent to a cell phone via SMS and also define that interest-area messages be sent as a weekly summary via e-mail. Organization managers may set high-level distribution rules.

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.

As previously shown, the EWSA decoupled architecture allows an agent-based application like JITIK to provide enterprise communities with a number of knowledge-oriented Web Services, especially useful in large organizations in which performance and scalability attributes become critical.

Figure 6. JITIK and enterprise systems interaction



Implementation of a Recommendation Service

Standard collaborative applications 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 the group activity and interests on a daily basis. Some of the internal services interaction needed to fulfill this case are the following:

- 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 (e-mail, 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 7. It should be noted that the use of generic services for scheduling, notification, and classification allows the task-specific code (read documents, send mail) to be encapsulated in the lower layers of the system, thus enhancing the abstraction level for the agent layer.

Implementation of a Classification Service

Another use case that serves as an example to our integration approach is a news delivery and classification service. Interaction between agents and services causes the flow of events between services and agents, as shown in Figure 8. The sequence of interaction is as follows:

- Once the user of the content management system (CMS) publishes a new article, it triggers a notification to the JITIK service. The CMS application will invoke the content-distribution service using a standard coarse-grained interface.
- After that, the classification service will attempt to match a set of topics for the article, according the contents of the ontology of the organization.
- When the content is classified, the request is passed on to the agent layer, where agents determine the set of content receivers considering their interest profiles and personal preferences.
- Finally, relevant information will be delivered to users through the internal notification services; the system may leverage the use of several distribution media (e-mail, SMS) to alert users of new documents and other application events.

CONCLUSION

We have presented an architectural approach aimed to allow integration of multi-agent systems as Web Services components. Besides its simplicity, the advantage of this approach is that it provides an efficient way of interoperating agent-based subsystems with Web-centric, loosely coupled systems. We think that this solution is a good compromise, given the current status of technology, and it allows rapid integration of modular systems conforming to open standards. As expected business results, we hope the solution to be helpful to do the following:

- Reduce the time to market of agent-based applications
- Deliver good performance for a large volume of users
- Improve the code base maintainability

It is to be noted that our solution assumes that the agent and Web component source code is available. With respect to the architectures presented in the literature,

Figure 7. Recommendation service-detailed view

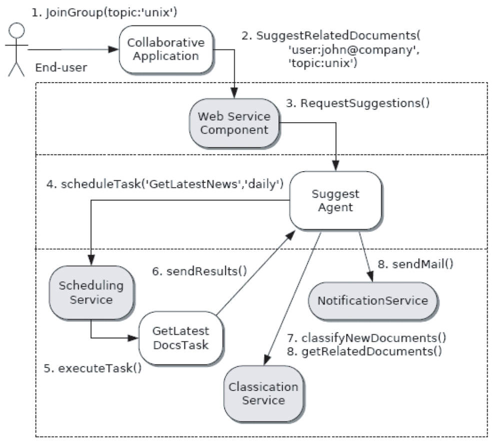
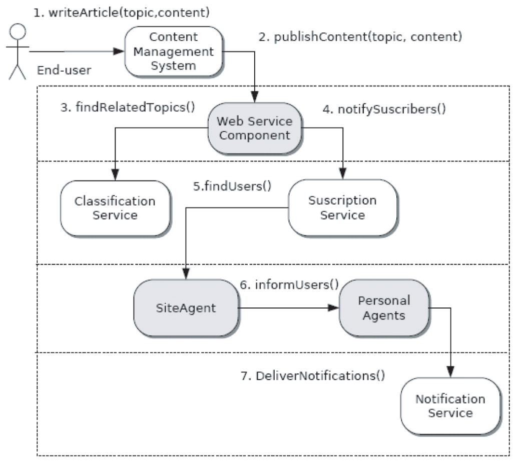


Figure 8. Classification service detailed view



this solution mostly trades off flexibility in favor of simplicity and maintainability.

We have presented experimental evidence to support our claim of efficiency. We have also presented a case study, which is the application of our architecture to the JITIK system. It is a multi-agent system to deliver information items to a distributed community of users.

In the near future, we intend to test our architecture with other real-world systems integrating agents in a Web-based framework. We currently are studying the methodological issues in order to guide the development of hybrid agent Web systems, since current agent development methodologies need to be strongly enhanced in order to suit our architecture.

REFERENCES

Barry, D.K. (2003). *Web services and service-oriented architectures: The savvy manager's guide*. Morgan Kaufmann.

Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE-A FIPA-compliant agent framework. In *Proceedings of the PAAM'99*, London.

Berre, D.L., & Fourdrinoy, O. (2002). Using JADE with Java server pages. http://sharon.cselt.it/projects/jade/doc/tutorials/jsp/JADE4JSP.html

Borges, B., Holley, K., & Arsanjani, A. (2004). Service-oriented architecture. http://webservices.sys-con.com/read/46175.htm

Cowan, D., & Griss, M. (2002). Making software agent technology available to enterprise applications [technical report HPL-2002-211]. HP Labs. http://www.hpl.hp.com/techreports/2002/HPL-2002-211.pdf

Dale, J., Willmott, S., & Burg, B. (2002). Agentcities: Building a global next-generation service environment.

Dickinson, I., & Wooldridge, M. (2003). Towards practical reasoning agents for the semantic Web. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, (pp. 827-834).

FIPA. (2002). FIPA abstract architecture specification. http://www.fipa.org/specs/fipa00001/SC00001L.html

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Genesereth, M.R., & Ketchpel, S.P. (1994). Software agents. *Communication of the ACM, 37*(7), 48-55.

Hendler, J. (2001). Agents and the semantic Web. *IEEE Intelligent Systems, 16*(2), 30-37.

Hoare, C. (1974). Monitors: An operating system structuring concept. *Communications of the ACM, 17*(10), 549-557.

Hunhs, M. (2002). Agents as Web services. *IEEE Internet Computing, 6*(4), 93-95.

Jakarta Project - The Apache Software Foundation. (2003). The tomcat Web server v. 4.1.

Jennings, N., & Wooldridge, M. (1996). Software agents. *IEE Review, 42*(1), 17-20.

Jennings, N.R. (2000). On agent-based software engineering. *Artificial Intelligence, 177*(2), 277-296.

Labrou, Y., Finin, T., & Peng, Y. (1999). Agent communication languages: The current landscape. *IEEE Intelligent Systems, 14*(2), 45-52.

Nwana, H.S., & Ndumu, D.T. (1999). A perspective on software agents research. *The Knowledge Engineering Review, 14*(2), 1-18.

Peng, Y., et al. (1998). A multi-agent system for enterprise integration. *In Proceedings of the 3rd International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM98)*, London, (pp. 155-169).

Petrie, C.J. (1996). Agent-based engineering, the Web and intelligence. *IEEE Expert, 11*(6), 24-29.

Preece, A., & Decker, S. (2002). Intelligent Web services. *IEEE Intelligent Systems, 17*(1), 15-17.

Rimassa, G. (2003). *Runtime support for distributed multi-agent systems* [doctoral thesis]. University of Parma.

Sun Microsystems, Inc. (2003). JSR-000154 Java(TM) servlet 2.4 specification (final release).

W3C. (2000). Extensible markup language

(XML) 1.0 (2nd Ed.). Retrieved from http://www.w3.org/TR/2000/REC-xml-20001006

W3C. (2001). Web services description language (WSDL) 1.1. Retrieved from http://www.w3.org/TR/2001/NOTE-wsdl-20010315

W3C. (2003a). Simple object access protocol (SOAP) 1.2. Retrieved from http://www.w3.org/TR/soap12-part1/

W3C. (2003b). Web services glossary [working draft]. Retrieved from http://www.w3.org/TR/2003/WD-ws-gloss-20030808/

Whitestein Technologies AG. (2003). Web services agent integration project.

ENDNOTE

1 An earlier summarized version of this work was published in ICEIS'05 conference selected papers book titled *Enterprise Information Systems VI* (ISBN: 1-4020-3674-4, Springer, 2006).

Eduardo H. Ramírez holds an MSc degree in information technology from the Tech of Monterrey, México, where he collaborated as a research assistant and staff engineer at the Center of Intelligent Systems. He is involved in the development and enterprise implementation of the JITIK Project. He is cofounder and CTO of Ensitech, S.C. a high-tech startup with consultancy and research activities on distributed computing and Web technologies. His current research work and interests involve agent-oriented software engineering, Web services and service oriented architectures, Semantic Web, rich Internet applications and collaborative knowledge management.

Ramón F. Brena is a full professor at the Center of Intelligent Systems, Tech of Monterrey, Mexico, since 1990, where he is head of a research group in distributed knowledge and multiagent systems. Dr. Brena holds a PhD from the INPG, Grenoble, France, where he presented a doctoral thesis related to knowledge in program synthesis. His current research and publication areas include: intelligent agents and multiagent systems, knowledge management, representation and distribution, Semantic Web, and artificial intelligence in general. Past research includes: program synthesis and software reuse, as well as automated reasoning. Dr Brena is member of the SMIA (AI Mexican Society), the AAAI and the ACM, and is recognized as an established researcher by the official Mexican research agency, CONACyT.