

Faculty of Information and Communication Technologies  
Centre for Intelligent Agents and Multi-Agent Systems

# WS2JADE: Integrating Web Service with Jade Agents

Technical Report: SUTICT-TR2005.03

Thang Xuan Nguyen  
Swinburne University of Technology  
Ryszard Kowalczyk  
Swinburne University of Technology  
30 July 2005



SWIN  
BUR  
NE

SWINBURNE UNIVERSITY  
OF TECHNOLOGY

# Table of Contents

<b>ABSTRACT</b> .....	<b>1</b>
<b>INTRODUCTION</b> .....	<b>1</b>
<b>RELATED WORK</b> .....	<b>2</b>
<b>WS2JADE APPROACH</b> .....	<b>3</b>
<b>Ontology Generation and Management</b> .....	<b>5</b>
<b>Interaction Mapping</b> .....	<b>6</b>
<b>Service Assignment Management and Service Discovery</b> .....	<b>7</b>
<b>WS2JADE IN ACTION</b> .....	<b>8</b>
<b>CONCLUSION AND FUTURE WORK</b> .....	<b>9</b>
<b>ACKNOWLEDGMENTS</b> .....	<b>10</b>
<b>REFERENCES</b> .....	<b>10</b>

## WS2JADE: Integrating Web Service with Jade Agents

### Abstract

Web services have gained popularity today for enabling universal interoperability among applications. In many scenarios, allowing software agents to access and control Web Services is important and hence the integration between these two platforms. In this paper, we focus on technical aspects of an integration framework of Web Services and Jade agent platform. The mismatch between description and communication of FIPA compliant agent platform versus Web Services are two key challenges that must be addressed. Our implementation, WS2JADE, is described and compared with WSDL2JADE - a previous implementation on the same topic, and WSIGS – a recent proposal of Web Service and Agent integration architecture. In contrast to other solutions, WS2JADE provides facilities to deploy and control Web services as agent services at run time for deployment flexibility and active service discovery.

### Introduction

Web services have gained popularity today for enabling universal interoperability among applications. In many scenarios, allowing software agents to access and control Web Services is important and hence the integration between these two platforms. In this paper, we focus on technical aspects of an integration framework of Web Services and Jade agent platform. The mismatch between description and communication of FIPA compliant agent platform versus Web Services are two key challenges that must be addressed. Our implementation, WS2JADE, is described and compared with WSDL2JADE - a previous implementation on the same topic, and WSIGS – a recent proposal of Web Service and Agent integration architecture. In contrast to other solutions, WS2JADE provides facilities to deploy and control Web services as agent services at run time for deployment flexibility and active service discovery.

With the emergence of the Web services standards, universal interoperability between applications is becoming reality. Following a loosely coupled integration model and common service access standards, Web services enable flexibility in the integration of heterogeneous systems. However, how to achieve the automation of Web services discovery, composition and invocation, or how to perform Web service execution monitoring and management are still open issues. Agent software is a promising technology to solve such problems. Unfortunately, Web services and agents were originally developed separately with different standards and specifications. Therefore, integrating between these two platforms becomes important in this context.

There has been many research on the topic of Web services and agent integration, to provide access to Web services from agent platforms and vice versa. This issue has also been addressed in the AgentCities project [2]. Main identified obstacles are the description mismatch and communication mismatch between Web services and Agents. A proxy approach has been recommended as the most suitable solution for the current stage of Web services and FIPA compliant agent platforms.

In this paper, we propose an enhanced solution for Web services and FIPA compliant Multi Agent System integration that offers many advantages over existing solutions. Specifically, we discuss different ways how Web services can be visible to Agents and how they can be accessed and used by Agents. We first present some background on the issue and then propose an approach for the integration followed by a proof-of-concept implementation for a specific case of Web services and JADE agent platform. We

contrast our tool with WSDL2JADE and WSIGS to show that WS2JADE, compared to WSDL2JADE, has more advanced features including dynamic Web Service deployment at runtime, improvements of more capable proxy agents and WSDL parsing functionality.

In the next section the paper discusses the related work. The WS2JADE approach is proposed in Section 3, followed by an example of its functionality in Section 4. Finally, Section 5 concludes the paper and outlines future work.

## Related Work

In the area of theoretically related work, a symmetric integration of Web Services and FIPA-compliant agents has been proposed in [2] as a high-level architectural recommendation from the AgentCities. The reason of this symmetry is that Web services were developed without the concept of agents (i.e. FIPA agents) and can exist without agents. Web Services can be architected to monitor and control other Web Services without interventions from agents [15]. The symmetric architecture takes into consideration that many Web Service clients, though may have autonomous characteristics of agents, are not strictly conformed to some specifications like FIPA. A proxy-based approach allows the two platforms to be evolved in parallel without imposing any restrictions on each other. This approach accepts the equity between the roles of agents and Web services, which is different to the traditional view that agent platforms are considered one level up from Web services, and agents take solely the roles of Web services providers and consumers.

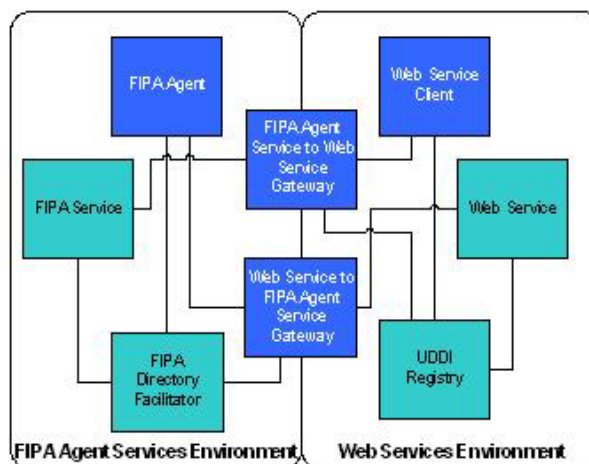


Figure 1: Integration Architecture [1]

As can be seen from Figure 1 taken from [2], a FIPA agent service environment is assumed to exist in parallel with a Web Service environment. The “FIPA Agent Service to Web service Gateway” on the border between the two environments is to allow FIPA agents to access Web Services by translating ACL messages to Web Service invocations. In the reverse order, the “Web Service to FIPA Agent Gateway” exposes and registers agent services in UDDI Registry Server so that any Web Service client can use them. Following AgentCities recommendations, two separate pioneer implementations have been proposed to solve two ends of the problems for FIPA compliant Jade Agent system: Exposure of Web Services to Jade agents by Sztaki [14] and exposure of Jade agent services to Web Services by Whitestein Technology [21]. Whitestein has released their tool WSAI (Web service Agent Integration) as an open source code in its first version. Another tool from Whitestein, WSIGS (Web Service Integration Gateway Service), is under development and its architecture has **been published in [5] and [6]**. WSDL2JADE has been released as an online program that converts WSDL file to Jade classes. It takes

an input of a Web service address and generates outputs of Jade agent code and agent ontology for the Web service. There is no run-time deployment capability. Based on some test cases we have carried out with Sztaki's WSDL2JADE, there are also some problems we notice with that tool. For example some XML enumeration values and type information are missing in the generated ontology files with XML Enumeration data type. It prevents client agents to use operations related to data type. Also a default single threaded agent in WSDL2JAVA and 1:1 mapping from Web service to agent prevent multiple agents to access a Web service simultaneously.

WSAI and WSIGS have been proposed by Whitestein Technology [5][6]. WSAI [21] allows Web Service clients to use JADE agents' services. In order to do this, WSDL files are generated for these agent services. Technically, at this stage WSDL files are created manually from these agents' behaviors. It also requires manual programming of "interface agents" to communicate with the target agent. These "interface agents" are created and destroyed per Web service client invocation of the service. However it appears that the applicability of WSAI in practical situations is limited because of two major obstacles. Firstly, Jade agent service specification is specified loosely and not based on message passing levels. This makes it difficult to automatically generate WSDL interfaces. Secondly, the default single-threaded mode of Jade agent, and the asynchronous and stateful nature of agent communication do not fit well in the Web Service communication model. There have been discussions of asynchrony versus synchrony in [2]. However, how to translate stateful communications of agents, in which conversations and history of past interactions with other agents are remembered, to stateless communications of Web Service is not discussed. We believe that the WSRF (WS-Resource Framework) specification [9] where a stateless service can have stateful resources could be applied here for a solution.

WSIGS is under development at the writing time of this paper. WSIGS proposes an architecture for bi-directional integration with no special agents for handling Web Service exposure [5]. WSIG is a set of codecs that do the translation between agents' ACL (Agent Communication Language) and Web Services' calls. To be visible in both environments, WSIG is registered as a special agent service in FIPA DF (Directory Facilitator) and a special Web Service endpoint in UDDI directories. When an agent wants to invoke a service (Web Service) registered in WSIG registry, the request is passed on to WebServiceInvocation, a component of WSIG, to perform the actual Web Service invocation. The requirement for services in one environment to be registered by their owners before they can be seen in other environments reflects the assumption that Web services need to be registered before they can be discovered. This is true for a model like UDDI but in more recent P2P models the assumption is no longer hold.

## **WS2JADE Approach**

This section describes a proposed approach for the integration of Web Service and Jade agents with WS2JADE. From an architectural perspective, WS2JADE, in accordance with [2], forms a Web service to FIPA Agent Service gateway. There are two distinct layers in WS2JADE: interconnecting layer and management layer. The interconnecting layer contains interconnecting entities that glue Web Service and agents together.

The management layer, being static, creates and manages those dynamic interconnecting entities. In WS2JADE, the interconnecting entities consist of special agents, ontology and protocol specifications. We call these special agents WSAG (Web Service Agents). WSAG are the agents capable of communicating with and offering Web Services as their own services. The combination of a static and a dynamic layer is a distinct feature of WS2JADE as compared to different tools mentioned in the previous

part. The WS2JADE static management layer is capable of active service discovery and automatically generating and deploying WSAG at runtime. It is an advancement over WSDL2JADE since it can automate the agent deployment process. Instead of being passive like WSIG, the WS2JADE is designed to actively discover Web services and registers them on DF if required. Also, each WSAG is multi-threaded and has its own Web service invocation module.

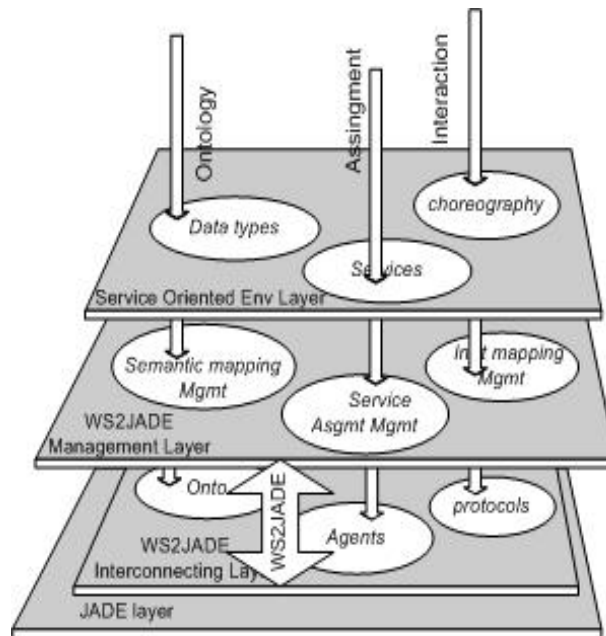


Figure 2: WS2JADE layer mapping

Figure 2 shows that WS2JADE management can be looked at from a different perspective as a layer which is capable of projecting Web Service or any Service Oriented environment layer into JADE agent layer. The output of this projection is the interconnecting entities. As depicted in this figure, three mappings are carried out by WS2JADE during the projection: ontology mapping, interaction mapping, and assignment mapping. These mappings are handled by three main components that construct the WS2JADE management layer: ontology generator and management, interaction generator and management, and service assignment management. These components, together with WS Discovery, compose the WS2JADE management part as shown in figure 3.

Figure 3 presents different components in WS2JADE system and how they are linked to JADE. The vertical rectangular box is WS2JADE, the horizontal one is JADE. Note that the overlap between WS2JADE and JADE is the components in WS2JADE interconnecting layer: generated interaction protocols, ontologies, and WSAGs. Figure 3 also illustrates a scenario for WS2JADE operation, in which a client agent searches for some service on DF. The DF can trigger WS2JADE to look up for available services in the Web service environment. If some Web services are found, their corresponding ontology and interaction models are generated. Also, a WSAG capable of accessing the Web service is generated. This WSAG registers the Web service as its service on DF and communication between the client agent and this WSAG can start if the client agent wants the service. The following will discuss each WS2JADE component in turn.

## Ontology Generation and Management

The ontology generator is responsible for ontology generation and ontology management. It translates data and its structure from Web service WSDL interfaces into meaningful information for Agents. A detailed explanation of a WSDL document can be found in WSDL specification [7]. WSDL describes abstract concepts and concrete entities. Abstract concepts are port type, operation, message and data type. Concrete entities are data encoding style, transport protocol and network address. In WS2JADE, the abstract concepts are relevant for the ontology mapping management as Agents need to know how to invoke operations of a Web service. The concrete entities are handled by the interaction translator and management component.

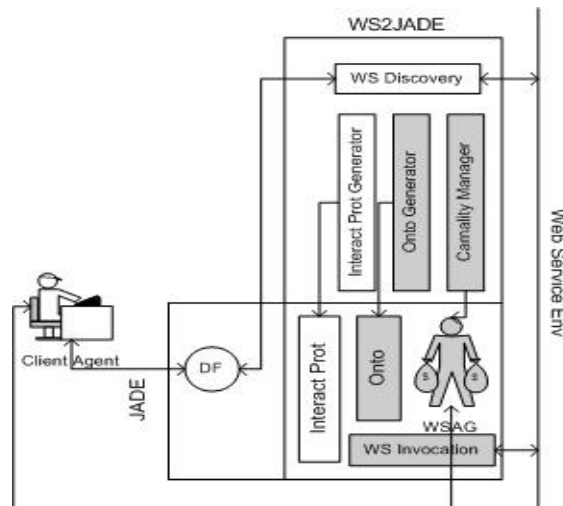


Figure 3: WS2JADE system components

WS2JADE ontology translator and management component converts Web services' data types and operation inputs and outputs into agent ontologies. The corresponding WSDL port type is tagged in the structure of the ontologies. According to JADE specification [16], JADE ontologies can be represented as Java classes, which are convenient for Jade agent's manipulation and processing. Alternatively, it can be in other formats such as RDFS and OWL for interoperability with other FIPA compliant agent platforms. Our WS2JADE toolkit supports Jade native ontology and OWL. To generate ontologies in Java, a WSDL data type is converted to a concept in agent ontologies. Two concepts are generated for each WSDL operation. One is for the operation input message and the other is for the output message. WSDL data types can be built-in XML types. The list of built-in simple XML data types are defined in the XML schema specification [16]. We map these built-in simple data types to Java primitives that are supported by Jade ontology representation. For XML data types that are not built-in, special customized Java classes are used, for examples, Beans, Enumeration Holders and Facet classes.

Generating ontologies in the OWL format is simpler than in Java classes because OWL and WSDL both use XML. Similar to Jade ontology generation approach, data types and messages in WSDL are mapped to concepts in OWL. There is a one-to-one relationship between concepts in ontologies generated in Java and OWL. OWL is still very new and subjected to changes; however we share the belief that it will continue to play an important role in Semantic Web with an increasing support from agent communities.

In addition to ontology generation, ontology management is important in WS2JADE. WS2JADE organises generated ontologies in an efficient way. For data types that can be shared among different Web services, the corresponding generated ontology concepts are shared and form a

common ontology base. It means that every time a new Web service is presented as an agent service, part of the existing ontology base and domain knowledge can be reused for this new service. Also, this allows the ontologies to be structured in a manageable way.

## Interaction Mapping

To translate Web Service transport messages (commonly SOAP) into agent ACL messages, the SOAP envelope is first projected into Java languages and then into ACL. We do not translate SOAP into ACL directly because of two reasons. Firstly, we want to reuse our generated ontologies and existing Java implementations of SOAP. Secondly, we make an assumption of the agent's intelligent capabilities to understand and process the messages according to its own logic in addition to language translation before forwarding the messages. This is best done by translating SOAP and ACL into Java – the native language for JADE.

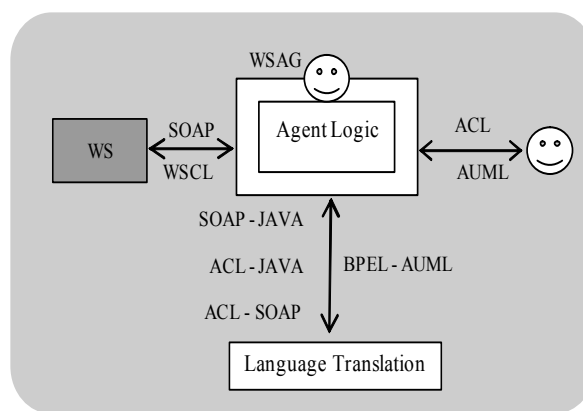


Figure 4: Interaction Translation

Figure 4 indicates that when a WSAG receives a SOAP message, it uses the Language Translation component to convert this message directly to ACL and send to the client agent. It can also perform some reasoning and modification on the message by converting the message to Java classes before any translation into ACL. In the Language Translation part of the interaction management component, Axis' JAX-RPC (Java API for XML based Remote Procedure Call) [19] implementation and JADE support for content languages and ontologies are used for translations between XML and Java, and between ACL and Java. Axis is one of the most popular open source implementations of SOAP today. On one hand, JAX-RPC, led by Sun, is a specification of Web Service Invocation framework in Java. In JAX-RPC specification, at the client side, Java to XML translation in remote method call is done through a mapping from Java client stubs to the SOAP message representation. On the other hand, in JADE, information represented in Jade ontology-supported classes (Java objects) can be converted to different ACL content languages, including SL and LEAP. As it can be seen from Figure 4, language translation is leveraged by the reuse of existing technologies instead of reinventing the wheel. SOAP-ACL translation is done by piping SOAP-JAVA and ACL-JAVA translation together. The main task of the language translation component is to map Axis stubs to Jade ontologies. However, we found out that due to the restrictions of Jade ontology and JAX-RPC classes it is not easy to convert data between them. In particular, an automation of the conversion process for any data types is difficult. We use special classes which represent the

ontology facets to preserve precisions in the conversion process. There has been a similar discussion in [14] for Sztaki WSDL2JADE. Complex data mapping in WS2JADE (for example mapping of Axis Holder and Enumeration types to Jade ontology concepts and classes) is done recursively through simple data type.

Table 1: Interaction Mapping

WS Interaction patterns	Agent protocols
Request-response	FIPA Request Interaction Protocol
Solicit-response	FIPA Request Interaction Protocol
Subscribe-Notification	FIPA Subscribe Interaction Protocol

In the interaction pattern translation component, we focus on choreography. By “choreography” we mean the required patterns of interactions among parties. It is in contrast to “orchestration” that describes how a composite Web service is constructed from other atomic services. For a composite Web Service, choreography is obtained by looking from an outsider’s perspective. It tells the Web service clients different steps of how to use a composite service.

We have mapped simple interactions implicitly described in WSDL documents into standard FIPA interaction protocols. Web service (WSDL version 1.2) provides four types of operation: one-way, request-response, solicit-response, and notification. In the one-way operation, a Web service client sends a request without receiving any response from the Web service. In the request-response, the client sends a request and receives a response synchronously. In the solicit-response, the Web service sends a solicit request to the client and receives a response. In the last type, notification, the Web service notifies the client without receiving any response. These four types of Web service operations lead to three common interaction patterns in practice: request-response, solicit-response, and subscribe-notification. The request-response and solicit-response interaction patterns correspond to those of Web service operation types. The subscribe-notification interaction describes the conversation style in which a client registers to the Web service in order to receive notifications when some event occurs. Table 1 summaries the mapping between these interactions styles into agent protocols.

More information on FIPA Request Interaction protocol, Request Interaction protocol, and Subscribe Interaction protocol can be found in [8].

### Service Assignment Management and Service Discovery

The Service Assignment Management component is responsible for cardinal mapping and service deployment management. The cardinal mapping management manages M:N relationship between Web services and WSAG. In WS2JADE, a number of Web services can be offered as services of different WSAGs. In other words, a WSAG can offer more than one Web Service and a Web Service can be offered by more than one WSAG. This cardinal relationship is managed through a registry that keeps records as triples of a Web service, a WSAG that offers the Web service, and a new name of the Web Service in the agent platform. The service assignment management also provides a tool for deploying and destroying WSAGs, and assigns new Web services to a WSAG. It informs the WSAG which ontologies should be used for a

particular newly assigned Web Service. If an assigned Web service is reported to be no longer available, the service deployment management removes the service from the list of the offered services of WSAGs and from the DF.

The Service Discovery component is designed to discover Web services. It is essentially a piece of software that can use Web service discovery protocols and translate the received information into agent service description for DF. As mentioned above, we prefer an active discovery model rather than waiting for services to be registered. At the time of this writing, Web service discovery protocol is complex and subject to change with the latest revised version of WS-Discovery specification which uses multicast protocols. Traditional Web service discovery mechanism of UDDI shares a common model with agent DF in a sense of accessing the directory. However, UDDI has evolved away from the concept of a “Universal Business Directory” that represented a master directory of publicly available services as DF still is. Most P2P based and multicast discovery protocols prove that requesting service providers to register the services is not always the case.

## WS2JADE IN ACTION

WS2JADE version 1.0, released in December 2004, implements the core components with a gray color in Figure 3. From a user perspective, WS2JADE provides two distinct tools: a tool for ontology generation and management and a tool for deploying and controlling Web Services as agent services. The ontology generation and management tool is offered through the combination of the ontology translator and the interaction pattern management components. The Web Service deploying and controlling tool is provided through combination of the service assignment mapping and the interaction pattern management components. The ontology generation and management tool can be used alone from a command line with an input of a WSDL address and several user options. The options include whether the user want to generate ontologies for the immediate imported XML types which are defined in other documents, and where to store Web Service stubs and agent ontologies. Another option is to specify the name for the agent service. The tool has been developed under `ciamas.wsjade.wsdl2jade` package with Java main class `ciamas.wsjade.wsdl2jade.WSDL2Jade`.

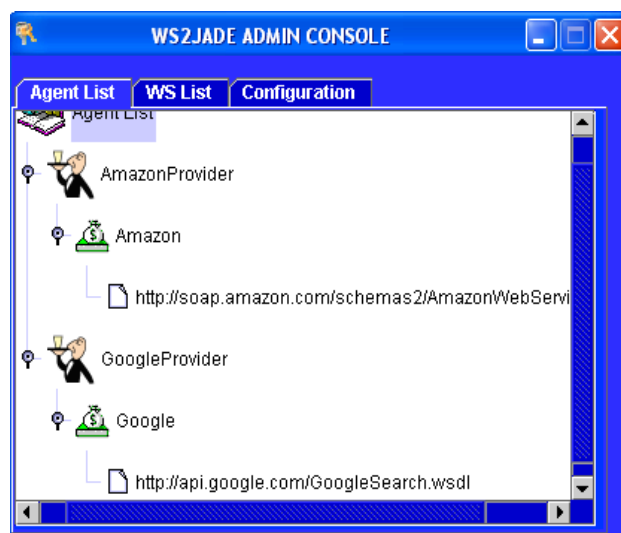
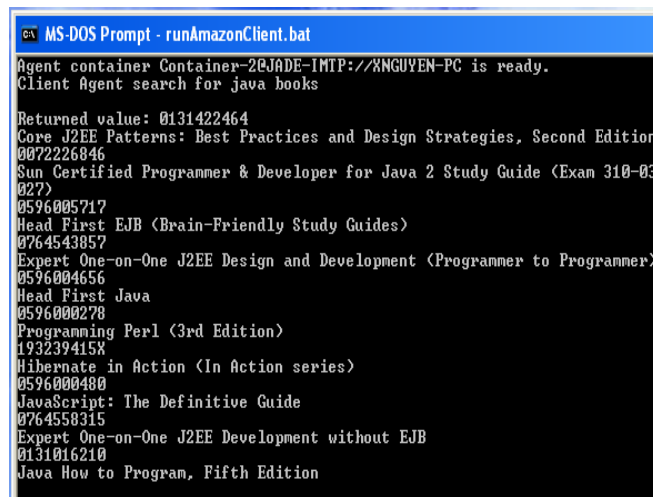


Figure 5: Service Deployment

In WS2JADE, the basic functionalities of a WSAG have been implemented in `ciamas.wsjade.wsdl2jade.utils.WSAgent`. The users can reuse the code or extend this base class to implement new functionalities, agent reasoning for example; since the base class supports message translation only. If the agent is designed with some complex behaviors in which single threaded mode is preferred, the base class can then be modified so that the agent is single threaded without difficulties. WS2JADE's Web Service deploying and controlling tool allows Jade agents to deploy Web Services on the fly. The tool has been developed in `ciamas.wsjade.management` package with the main class for graphical administration interface at `ciamas.wsjade.management.utils.Admin`. WS2JADE operates as follows. First important parameters in the configuration panel need to be set. The gateway agent container is then started and new agents are created from the "Agent List" tab. After deploying the new agents, new WSAG agents appear on the Jade platform and ready to be assigned with Web Services. Once the Web Services are assigned they have to be activated in order to be used by client agents. The deployment process can be done simply as shown in Figure 5, in which "Amazon" and "Google" services are deployed with a few clicks and inputs of the real Amazon and Google Web Service addresses. The WSAG who provide these services are "AmazonProvider" and "GoogleProvider". Ontology packages are generated and compiled on the fly, and the Web Services are now available as services of these WSAG agents. The generated ontology package is stored under the "Jade output folder" and can be sent to a Web Server for downloading and using by the client agents.



```
MS-DOS Prompt - runAmazonClient.bat
Agent container Container-2QJADE-IMTP://XNGUYEN-PC is ready.
Client Agent search for java books

Returned value: 0131422464
Core J2EE Patterns: Best Practices and Design Strategies, Second Edition
0072226846
Sun Certified Programmer & Developer for Java 2 Study Guide (Exam 310-035027)
0596005717
Head First EJB (Brain-Friendly Study Guides)
0764543857
Expert One-on-One J2EE Design and Development (Programmer to Programmer)
0596004656
Head First Java
0596000278
Programming Perl (3rd Edition)
193239415X
Hibernate in Action (In Action series)
0596000480
JavaScript: The Definitive Guide
0764558315
Expert One-on-One J2EE Development without EJB
0131016210
Java How to Program, Fifth Edition
```

Figure 6: Amazon Client Agent

## Conclusion and future work

This paper presents WS2JADE toolkit for integrating Web Services and Jade agent platform that allows Jade agents to offer Web Services as their own services at runtime. Comparison with another pioneer implementation from Sztaki is discussed to demonstrate the advanced features of the WS2JADE implementation.

Although WS2JADE offers many advantages over other existing tools its current version has also some aspects that could still be improved. One-way integration is one of them. At the moment we are reluctant in any implementations of the other end as we believe that, as discussed in the previous sections, there is still a lack of substantial theoretical work on the topic of agent to Web Service integration. This is a subject of our on-going research. As a contribution to Agent OpenNet, we are working on adding our WS2JADE deployment node into the agent network. Our current and future work also involves

improvements of the semantic processing capability of the ontology management component. It also includes an implementation of auto-translation of Web Service choreography to agent protocol for dynamic mapping, invocation, and monitoring composite Web Services. This will support more dynamic and robust integration between the two platforms.

## ACKNOWLEDGMENTS

This work has been partially supported by the Adaptive Service Agreement and Process Management (ASAPM) in Services Grid project (AU-DEST-CG060081) and the EU FP6 Integrated Project on Adaptive Services Grid (EU-IST-004617). The ASAPM project is proudly supported by the Innovation Access Programme -International Science and Technology established under the Australian Government's innovation statement, Backing Australia's Ability.

## REFERENCES

- [1] AgentLink, European Co-ordination Action For Agent-Based Computing, <http://www.agentlink.org/>
- [2] Agentcities Web services Working Group. Integrating Web services into Agentcities Technical Recommendation, <http://www.agentcities.org/rec/00006/>
- [3] Amazon.com, Inc. , Amazon Web Service [www.amazon.com/gp/aws/landing.html](http://www.amazon.com/gp/aws/landing.html)
- [4] B. Bauer, J.P Muller, and J. Odell. Agent UML: formalism for specifying multiagent software systems, in Agent-Oriented Software Engineering, Ciancarini, P. and Wooldridge, M., Editors. LNCS, Vol 1957, 2001: Spinger, pp. 207-221
- [5] D. Greenwood, M. Calisti, "An Automatic, Bi-Directional Service Integration Gateway", , IEEE Systems, Cybernetics and Man Conference; 10-13 October, 2004, the Hague, Netherlands
- [6] D. Greenwood, M. Calisti, "Engineering Web Service – Agent Integration", IEEE Systems, Cybernetics and Man Conference; 10-13 October, 2004, the Hague, Netherlands
- [7] E. Christensen et al. Web Service Description Language (WSDL 1.1), <http://www.w3.org/TR/wsdl>
- [8] Foundation for Intelligent and Physical Agents. Interaction Protocol Specification, <http://www.fipa.org/repository/ips.php3>
- [9] Globus Alliance, IBM, and HP, "Web service Resource Framework", <http://www.golubs.org/wsrfr>
- [10] Google, Google Web APIs <http://www.google.com/apis/>
- [11] J. Cardoso. Quality of Service and Semantic Composition of Workflows. Ph.D. Thesis, University of Georgia, 2002.
- [12] J. Dale. Exposing Web Services, Agentcities.NET iD2, Agentcities Lisbon, 9-10 September, 2002
- [13] C. Li, L. Li, "An agent-based approach for grid computing", IEEE Parallel and Distributed Computing, Applications and Technologies, 2003, pp 608-611.
- [14] L. Zs. Varga,Á. Hajnal: "Engineering Web Service Invocations from Agent Systems". Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Prague, Czech Republic, pp. 626-635, June 16-18, 2003. Lecture Notes in Computer Science, Volume 2691, V.
- [15] Marik, J. Müller, M. Pechoucek (Eds.), Multi-Agent Systems and Applications, Springer-Verlag Heidelberg, ISSN: 0302-9743, pp. 626-635, 2003
- [16] N. Catania, et al. Web Service Management, <http://devresource.hp.com/drc/specifications/wsmf/WSMF-WSM.jsp>, 2003
- [17] N. Cavantzas et al. Web Services Choreography Description Language Version 1.0, <http://www.w3.org/TR/2004/WD-wscdl-10-20041012/>
- [18] P. V. Biron, K. Permanente, and A. Malhotra, XML Schema Part 2: Datatypes Second Edition, <http://www.w3.org/TR/xmlschema-2/>
- [19] S.P. McIlraith, T. Cao Son, and H. Zeng. Semantic web services. IEEE Intelligent Systems, 16(2):46-53, March/April 2001
- [20] Sun Microsystems, Inc. Java API for XML-Based RPC (JAX-RPC), <http://java.sun.com/xml/jaxrpc/index.jsp>
- [21] Telecom Italia Lab. JADE (Java Agent Development Framework), <http://sharon.csel.it/projects/jade/>
- [22] Whitestein Information Technology Group AG. Web services Agent Integration Project, <http://wsai.sourceforge.net/index.html>
- [23] WS2JADE, Web services to Agents <http://www.it.swin.edu.au/centres/ciamas>

[24] Y. Kalfoglou, Y. Schorlemmer, Ontology mapping: the state of the art. *The Knowledge Engineering Review* 18(1):1-31, 2003