WS2JADE: A Tool for Run-time Deployment and Control of Web Services as JADE Agent Services

Xuan Thang Nguyen, Ryszard Kowalczyk,
Mohan Baruwal Chhetri and Alasdair Grant

Abstract. Web services and software agent technologies are two areas that have attracted substantial research and industry interests in recent years. On the one hand, the Web services technology is gaining popularity because of its well-defined infrastructure aiming at enabling interoperability among heterogenous applications. On the other hand, the agent technology aims at providing intelligent autonomous capabilities for distributed components. A combination of these two technologies could create an environment where Web services and agents can employ and compliment each others' strengths. In this chapter, we propose a framework called WS2JADE for integrating Web services and the JADE agent platform. In particular, the technical aspects of run-time deployment and control of Web services as agent services with WS2JADE are presented. We relate our framework to other solutions in the area and show how new emerging Web services management technologies can be used with WS2JADE for enabling Web services management with agents. The management capabilities are demonstrated with simple examples of using WS2JADE for service discovery, composition and deployment with JADE agents.

1 Introduction

With the emergence of Web service standards, the universal interoperability between distributed applications is fast becoming a reality. Web services follow a loosely coupled integration model and use industry-standard protocols to facilitate the seamless integration of heterogeneous systems within and across organisations. While the Web service technology offers many distinctive advantages and benefits, it still has certain limitations potentially hindering its broader adoption in more complex applications. In particular it involves the limited support for the management of the discovery, composition and execution of Web services (e.g. [1, 18, 24, 25]).

The agent technology offers abilities of intelligent operations, interactions and cooperation between autonomous components that can be used in automating management tasks and business processes, and has also been recognized as a promising technology for managing Web services [9, 10, 17, 25]. However, because Web services and agents were originally developed separately with different standards and specifications their integration is not straightforward. Realising the benefits of integrating these two technologies, significant research has been carried out in this direction. The Agentcities Web Service Working Group's project [4] is an example of such an effort aiming at addressing the issue of Web services and agents integration. More recently, Agentcities has also created openNet [26] that provides a test-bed environment for integrating software agents, Web services and Semantic Web services. The integration of software agents and services in general has been proposed by Luck et al. [20] as one of the major tasks for the agent community. The main obstacles in integrating Web services and agents are the mismatches in description and communication used by these two technologies. One way to overcome these obstacles can be a proxy-based integration approach that allows the two technologies to evolve in parallel without imposing any restrictions on either, providing the gateway to bridge the Web services and agents.

Web services can work across organizations and be composed to create new Web services. There are business scenarios in which many Web services from different administration domains need to be tied together in cross-organisational business processes or complex composite applications. Consequently, Web services management becomes a complex task that requires a high level of intelligent capabilities and automation support. The emerging Web services management technologies, such as WS-Management [1] and Web Services Distributed Management (WSDM-MUWS [24], WSDM-MOWS [25]), define additional interfaces for Web services needed for their management. However they do not specify the management mechanisms, i.e. 'how', 'when' or 'why' these Web services can be managed. The agent technology is well poised to fulfil this role and support Web services with the required management mechanisms. If agents were able to interact with Web services, the agent technology could be integrated into the Web service management model effectively and used to manage Web services.

In this paper, we propose a framework for integrating Web services and FIPA compliant software agents, which offers many advantages over previous solutions. Specifically it enables run-time deployment and control of Web services as agent services. The management of Web service discovery, composition and execution can be performed both at the agent service and the Web service levels. Section 2 overviews the related work in the area of integrating Web services and agent technologies. Section 3 presents the proposed framework called WS2JADE and its technical aspects. Section 4 discusses the need for Web service management and current standards available for Web service management. It also discusses how and why agent technology, and in particular WS2JADE can support Web service management. Section 5 demonstrates the specific capabilities of WS2JADE with simple examples of Web service management including service discovery, composition and deployment with JADE agents. Finally, concluding remarks and an outline of the future work are presented in Section 6.

2 Related Work

The agents and Web services communication, and a synthesis of the agents and Web services have been addressed in a number of works. In [11], the authors discuss the "agentification" for Web services, in which the legacy systems could be re-engineered into agent-based Web services. In [10] an agent-based architecture is proposed for the selection and composition of Web services. The trend of using agents to monitor and control Web services composition has been increasing recently, evidenced by a number of other publications (e.g. [12, 27, 28]). In most of those works however, agents do not conform to any specific standard and details of using agents to invoke Web services are not formally described. The general assumption is that any agent can request any Web service by acting as the Web service's client. In practice, if this assumption held then the agents' code would need to contain the Web service invocation code. For FIPA-compliant agent systems, this also means that in addition to agent communication languages, the agents' programmer need to consider the low level details of Web services invocation. There have been attempts to provide a framework in which agents and Web services, with separation of concerns in their implementations, can communicate with each other. A symmetric integration of Web services and FIPA-compliant agent platforms has been proposed in [4] as a high-level architectural recommendation from the Agentcities [3]. It is the fact that Web services were developed without the concept of agents (i.e. FIPA agents) and can exist without agents. The symmetric architecture takes this into consideration and also that many Web service clients may have autonomous characteristics of agents without conforming to the FIPA specifications. A proxy-based approach allows the two platforms to be evolved in parallel without imposing any restrictions on each other.

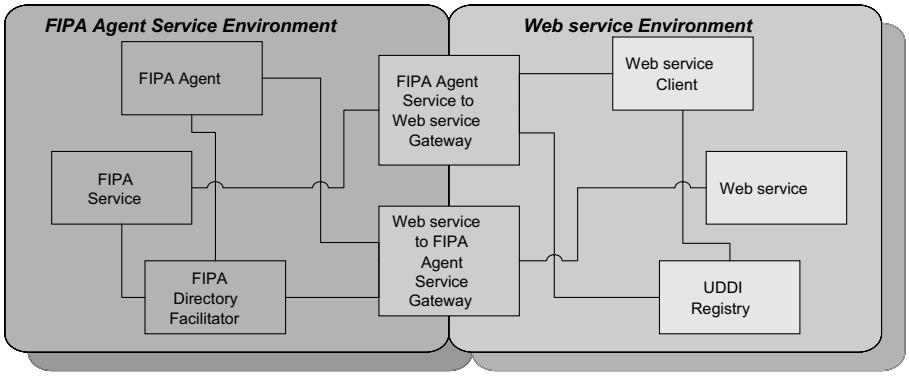

Fig. 1. FIPA Agents – Web services Integration Architecture [4]

A FIPA agent service environment can exist in parallel with a Web service environment as depicted in Figure 1 [4]. The "*FIPA Agent Service to Web service Gateway*" on the border between the two environments allows a FIPA agent to access Web services by translating ACL messages into Web service invocations. In the other direction, 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 the Agentcities recommendations, two separate solutions have been proposed to solve two ends of the problems for the FIPA compliant JADE agent system. One solution exposures Web services to JADE agents [19] and JADE agent services to Web services [7, 8, 33]. Whitestein Technology has proposed WSAI (Web Service agent Integration) [33] and WSIGS (Web Service Integration Gateway Service) [7], and has already released the tool WSAI as an open source code in its first version. WSIGS is under development and its architecture has been published in [7] and [8]. WSDL2JADE has been released by Sztaki [19] as an online program that converts WSDL file to JADE classes. It takes a Web service address as an input and generates outputs of JADE agent code and agent ontology for the Web service. However, there is no run-time deployment capability.

WSAI [33] 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 the agents' behaviors. It also requires "interface agents" to communicate with a target agent. These "interface agents" are created and destroyed per Web service client invocation of the agent service. However it appears that the practicability of WSAI is limited. This is because of the default single-threaded mode of JADE agent, and the asynchronous and stateful nature of agent communication do not fit well in the current implementation stage of Web service communication model, which is stateless and mostly synchronous. There have been discussions of asynchrony versus synchrony in agents and Web services in [4]. WSIGS [7] is under development at the writing time of this paper. WSIGS proposes an architecture for the bi-directional integration without special agents, and provides a set of codecs that do the translation between the agents' ACL (Agent Communication Language) and Web services' calls. To be visible in both environments, WSIGS is registered as a special agent service in FIPA DF (Directory Facilitator) and as a special Web service endpoint in UDDI directories. When an agent wants to invoke a service (Web service) registered in WSIGS registry, the request is passed on to WebServiceInvocation, a component of WSIGS, to perform the actual Web service invocation. There is no active discovery mechanism provided in WSIGS. Services in one environment need to be registered by their owners in a public directory before they can be seen in the other environment.

The area of Web services management has attracted substantial effort from industry and significant research among academic communities. Web services-based applications can work across enterprise boundaries more now than any other types of applications. However, being distributed and dynamic in nature, Web services require an efficient management model that can integrate seamlessly and work dynamically in a distributed environment. There have been two main approaches in tackling this problem and they are complimentary to each other. In the first approach, management is done through Service Level Agreement (SLA) or service contracts. This approach assumes that control over Web services is not visible to external managers and service management is enforced through agreement terms (rewards, penalty, preference, etc) specified in the contracts. Current work on this approach can be found in [15] and [37]. In the second approach,

external mangers can exert direct control over a Web service. For this to happen, the Web service that needs to be managed must expose some manageability interfaces. This approach is the latest emerging management model from industry, supported for example by two specifications, Web Services Distributed Management (WSDM) [22] originally from HP and WS-Management [1] from Microsoft. At this initial stage, WS-Management and WSDM appear to overlap in a number of aspects.

WS2JADE proposed in this paper aims at enabling seamless and dynamic Web services and agent technology integration, and is geared for agent-based management of Web services. To do this, it utilizes useful features from emerging Web service management models; especially Web services based management models like WSDM and WS-Management. It provides facilities to integrate with future implementation of WSDM and WS-Management, and offers the deployment of Web services as agent services at run-time. It is the first step towards the ultimate goal of automation of Web service management using agents. WS2JADE does not automatically provide intelligent algorithms for Web services management. However, it provides a gateway for the existing management algorithms to be employed in the multi-agents environment. With WS2JADE, an overlay management network of agents can be formed to manage the Web services network and communicate with the underlying management of Web services.

3 WS2JADE: Web Services-Agents Integration

This section describes the proposed approach for the integration of Web services and JADE agents with WS2JADE. From an architectural perspective, WS2JADE, in accordance with [4], forms a gateway between a Web service and FIPA agent service. As depicted in Figure 2 there are two distinct layers in WS2JADE: the interconnecting layer and the management layer. These two layers provide facilities to connect the Service Oriented Layer (Web services) and the JADE agent layer together.

The layer which contains entities that directly and dynamically interconnect Web services and agents is the Interconnecting Layer. 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 the static and dynamic layers is a distinct feature of WS2JADE as compared to earlier tools mentioned in the previous section. The WS2JADE management layer is capable of active service discovery, and automatically generating and deploying WSAG at runtime. It is an improvement 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 generate equivalent Web services if required. Also, each WSAG is multi-threaded to take an advantage of supporting concurrent requests of Web services and has its own Web service invocation module.

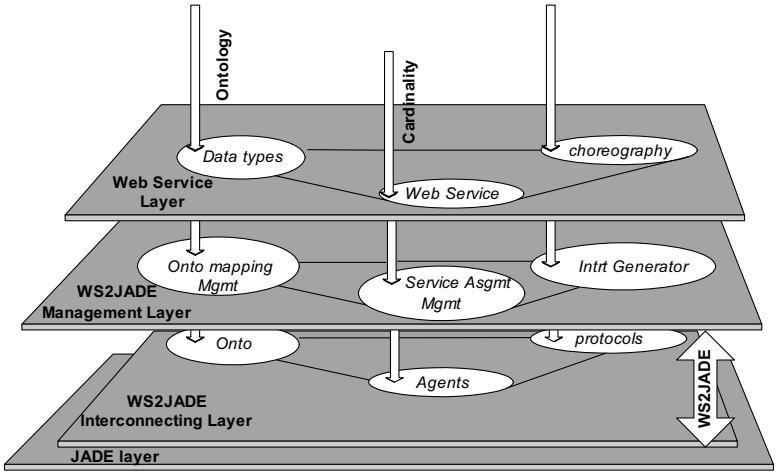

Fig. 2. WS2JADE layer mapping

Figure 2 shows that WS2JADE management can be looked at from a different perspective as a layer which is capable of projecting the Web services (or any service-oriented environment) layer into the JADE agent layer. The result of this projection is represented by the interconnecting entities. As depicted in that 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

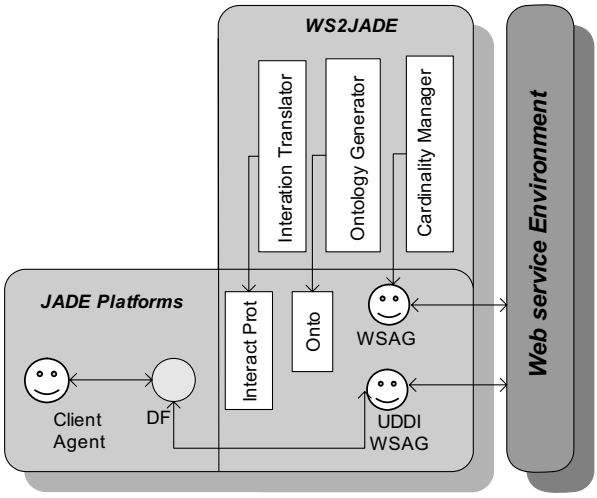

Fig. 3. WS2JADE components

that form the WS2JADE management layer: *ontology generation and management* component, *interaction translation* component, and the *service assignment management* component. These components interact with each other and provide the WS2JADE management part as shown in Figure 3.

Figure 3 presents the different components within WS2JADE system and how they are linked to JADE agent system. The vertical rectangular box depicts WS2JADE, the horizontal one depicts JADE. Note that the overlap between WS2JADE and JADE consists of components in the WS2JADE interconnecting layer: generated interaction protocols, ontologies, and WSAG. 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 subsections discuss each WS2JADE component in more details.

3.1 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 WSDL can be found in WSDL specification [9]. WSDL describes abstract concepts and concrete entities. The abstract concepts are port type, operation, message and data type. The 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 the operations of a Web service. The concrete entities are handled by the interaction generator and management component. WS2JADE ontology generator and management component converts Web services' data types, and the operation inputs and outputs into the agent ontologies. The corresponding WSDL port type is tagged in the structure of the ontologies. Since JADE is implemented in Java, JADE ontologies are often represented as Java classes, which are convenient for JADE agent's manipulation and processing. Alternatively, it can be in other formats such as RDFS [34] and OWL [35] for interoperability with other FIPA compliant agent platforms provided that there are suitable codec plug-ins. 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 [29]. 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.

The following is an example of generated ontology for a XML type Enumeration defined as:

```
<s:simpleType name="Mode">
        <s:restriction base="s:string">
        <s:enumeration value="On" />
        <s:enumeration value="Off" />
        </s:restriction>
</s:simpleType>
```

The generated ontology concept in a Mode class extends from JADE Concept class:

```
public class Mode implements jade.content.Concept{
ModeFacet facet=null;
java.lang.String enumEle;
        public Mode(){}
        public java.lang.String getEnumEle(){return enumEle;}
        public void setEnumEle(java.lang.String enumIn){
          this.enumEle = enumIn;
        }
}
```

This Mode class has a facet defined as in ModeFacet class, which restricts the value of enumEle slot in Mode concept to one of values in the xml Enumeration.

```
public class ModeFacet implements jade.content.schema.Facet{
public void validate(jade.content.abs.AbsObject abs,
jade.content.onto.Ontology onto) throws
jade.content.onto.OntologyException {
try {
        jade.content.abs.AbsPrimitive p =
jade.content.abs.AbsPrimitive) abs;
        boolean valid = false;
        java.lang.String    obj=p.getString();
        if(obj.equals("On"))
        valid = true;
        if(obj.equals("Off"))
        valid = true;
        if (!valid) {
```

```
            throw new jade.content.onto.OntologyException("Facet
restriction violated");
              }
            }
            catch (Exception e) {
            throw new
jade.content.onto.OntologyException("Invalid Facet Object",
e);
              }
            }
}
```

Generating an ontology 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 the 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 the ontology generation, the 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. This 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.

3.2 Interaction Translation

The interaction translation component handles the conversion from Web service communication into agent communication. Specifically, it converts Web service transport messages into ACL envelopes, and Web service interaction patterns into agent protocols. These correspond to two sub-functionalities: language translation and interaction pattern conversion.

3.2.1 Language Translation

To translate Web service transport messages (commonly SOAP) into agent ACL messages, the SOAP envelope is first projected into the Java language and then into ACL. We do not translate SOAP directly into ACL for two reasons. Firstly, we want to reuse our generated ontologies and the existing Java implementations of SOAP. Secondly, we want to make use of the agent's capabilities to understand and process the messages according to its own logic (in addition to language translation) before forwarding them.

This is best done by translating SOAP and ACL into Java – the native language for JADE.

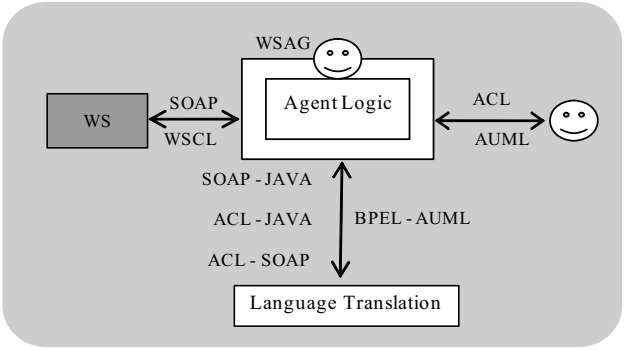

Fig. 4. Language Translation

Figure 4 shows 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 translation component, Axis' JAX-RPC (Java API for XML based Remote Procedure Call [30]) implementation and an extension of JADE ontology package are used to support SOAP to JAVA translation. Axis is one of the most popular open source implementations of SOAP today. On the 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. We can see from Figure 4, that language translation is leveraged by the reuse of many 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 the Axis stubs to JADE ontologies. However, 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. Hence, we use special classes which represent the ontology facets to preserve precisions in the conversion process. There has been a similar discussion in [19] for Sztaki's 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.

3.2.2 Interaction Pattern Translation

In the interaction pattern translation component, WS2JADE focuses 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 [14]. Web service (WSDL version 1.2) provides four types of operations: 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 summarizes the mapping of these interactions styles to agent protocols. More information on FIPA Request Interaction protocol and Subscribe Interaction protocol can be found in [14].

Table 1. Interaction pattern mapping

Web Service Interaction Patterns	Agent Protocols
Request-Response	FIPA Request Interaction Protocol
Solicit-Response	FIPA Request Interaction Protocol
Subscribe-Notification	FIPA Subscribe Interaction Protocol

3.3 Service Assignment Management and Service Discovery

The Service Assignment Management component is responsible for the cardinality mapping and service deployment management. The cardinality mapping manages M:N relationship between Web services and WSAG. Offering the same Web services on different WSAG allows better load balancing and reduces probability of service access failure when some WSAG are down. Offering more than one Web service on a proxy agent allows related Web services to be grouped together. The cardinality mapping of M:N permits a number of Web services can be offered and duplicated as services of different WSAG. This relationship is managed through a registry that keeps records as triples of the Web service, a WSAG that offers this Web service, and a new name for the Web Service in the agent platform. The service assignment management also pro-

vides a tool for deploying and destroying WSAG. It assigns Web services to a WSAG informing it which ontologies should be used for the 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 WSAG 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 descriptions for the DF. As mentioned earlier, 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 [17] specification which uses multicast protocols. Traditional Web service discovery mechanism of UDDI shares a common model with agent DF in the 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. Because UDDI implementations are widely available at this stage, in WS2JADE version 1.0 the Web services discovery is available as a UDDI proxy agent. This agents supports special discovery services which can be configured to proxy to any UDDI version 2 servers, including Microsoft and IBM UDDI inquiry servers.

3.4 Remarks

As mentioned earlier and discussed in [4], the main difficulties in integration of Web services and FIPA compliant agent platforms are the mismatches in communication and descriptions. These are summarized in Tables 2 and 3, respectively. For translation from a Web service to an agent service, WS2JADE handles these mismatches through its different components. Although the current version of WS2JADE is operational and offers many advantages over other tools, it can still be improved in a number of areas.

For example to increase the ontology reuse and avoid redundancy, the semantic mapping management component can be extended to detect semantic equivalence of two syntactically different generated concepts and keep one of them only. In [14], the authors focus on this topic and outline some approaches to achieve this. The current version of WS2JADE has not yet implemented that specific feature but future versions will include it.

Table 2. Communication Mismatch

FIPA agent communication	W3 Web Service communication
ACL/IIOP+HTTP	SOAP/HTTP
Asynchronous	Synchronous/Asynchornous
Stateful	Stateless

Another area is the interaction pattern translations. Web Service Choreography Description Language (WS-CDL) [36] has been under development for some time. WS-CDL is considered as a layer above WSDL in the Web service technology layer hierarchy. It describes a set of rules to explain how different partners may act in a conversation. W3C recommends it as a necessary complement to BPEL and programming languages like Java that only describe one endpoint, and not the whole system of interaction.

Table 3. Description Mismatch

FIPA Agent Service	W3 Web Service
Name – Name of the service	Names of services, port types, operations, etc.
Type – Type of the service	Type – Container of data type
Protocols – List of supported protocols	Message – Abstract, typed definition of data
Ontologies – List of supported ontologies	Operation – Abstract description of action
Languages – List of supported content languages	Port Type – Abstract set of operations
Ownership – The owner of the service	Binding – Protocol & data format specs for a port type
	Port – Single endpoint as combination of a binding and a port type.

In the WS2JADE approach, we plan to convert BPEL4WS and WS-CDL (however not at this stage of WS-CDL development) into agent Unified Modeling Language (AUML [5]) for the overall protocol representation in UML template. AUML is an extension of UML language for agents and has been used as a standard language to describe FIPA interaction protocols. The interaction translation will keep generated AUML documents in its protocol specification repository which can be looked up by client agents (or the client agents' designers) before using the service. In this version of WS2JADE we have not implemented the translation of WSCL to AUML. One reason for this is the instability and immaturity in Web Service Choreography as evidenced by the suppression of WSCI (Web Service Choreography Interface) and WSCL (Web Service Choreography Language) [2] by WS-CDL [36] which is still in the first draft version.

4 Management of Web Services

In this section we first discuss existing standards for the management of Web services and how these standards can be implemented. Then we discuss why agents are suited for Web service management and how agent systems, in particular JADE, can fit into emerging management models and take advantage of them by using WS2JADE.

4.1. Web Services Management Frameworks with Web Services

Web services represent vital resources for any business organization and are prevalently used to carry out business processes and transactions between businesses or within an enterprise. When they interact with other Web services, they form a logical network which can be distributed across enterprises. From the business organization's view point, the ability to manage such a logical network, and automate and integrate various internal functions is very critical in order to provide Web service security, usability and reliability. In order to achieve this, there is a need for standards and tools which allow the management of Web services. At present, there are very few standards which address the management of business processes or the underlying application services they rely on. The very nature of Web services makes this task (of managing Web services) all the more challenging. Some of the characteristics of Web services which contribute to this challenge include *distributed nature, extensibility, standardization and discovery* [22].

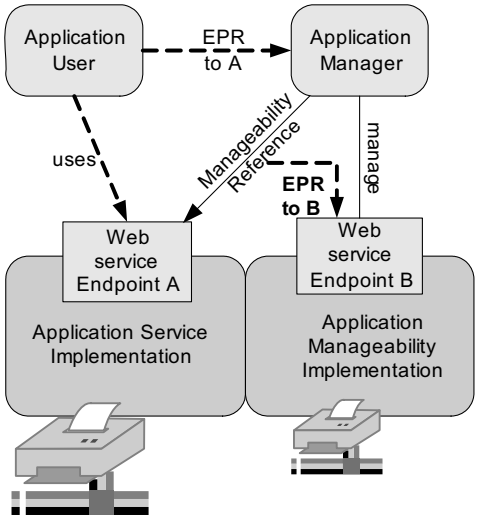

Fig. 5. Management of a Web service or a resource exposed as a Web service using a Web service [22]

Hewlett-Packard has proposed the Web Services Management Framework (WSMF) which is a logical architecture for the management of resources [22]. Extending this work further, with the support from other companies including IBM and DELL, HP has released the Web Services Distributed Management (WSDM) specification which defines how the management of any resource can be accessed via web service protocols – Management Using Web Services (MUWS) [24] and how Web services can be managed using Web services – Management of Web Services (MOWS) [25]. The specification was submitted to Organization for the Advancement of Structured Information

Standards (OASIS) and has been accepted as a standard. Microsoft, in collaboration with various IT companies has released its own SOAP-based protocol for managing systems (including Web services) called Web Services for Management (WS-Management) [1]. WS-Management shares the basic idea with WSDM in identifying a manageable resource and communicating with it.

As shown in Figure 5, each time a resource is exposed as a Web service or a Web service is made available, it also provides a reference to its manageability endpoint. The Application Manager can find this reference to the Application Manageability Implementation, and then performs management actions on the application by exchanging messages with Web service endpoint B (manageability endpoint). In more complex scenarios, the Application Manager can find out the relationships among different application processes from their manageability interfaces and use them as the inputs for a more dynamic management algorithm.

4.2 Enabling Web Service Management with WS2JADE

This part discusses the application of agent technology for Web service management and WS2JADE capability in integrating with emerging Web services based management models. This discussion is a significant departure from previous discussions on how Web services can be accessed and used by agents since management functionalities discussed so far are on the agent service level. In this part, we discuss how agent-based management can be carried out on the Web services level.

A management system, in general, requires some level of automation. In an ideal situation, it should be able to diagnose faults and take appropriate correction actions. A software agent is an autonomous software entity or computer system situated in some environment. Therefore a management system itself or its sub-components shares the fundamental characteristic of an agent. As a result, modelling management systems as agents or agent systems is a natural choice. There have been many management systems with agents. Examples are Sun's Java Management Extension (JMX) [31]-a new feature in version 5.0 of Java 2 platform and McAfee's ePolicy Orchestrator [21] software. In JMX, a given resource is instrumented by Java objects known as Managed Beans, or MBeans. These MBeans are registered and managed by management agents, known as JMX agents. The specification provides a set of services for JMX agents to manage MBeans. In ePolicy Orchestrator, Anti-Virus software on client PCs are monitored by a set of agents for possible virus outbreaks. In these examples, the management environments in which the managed objects (MBeans, Anti-Virus software) reside do not scale up globally. In particular, the environment is limited to Java programming languages for JMX and to a local computer for ePolicy Orchestrator. Consequently, the management agents in these local environments do not need to follow well-defined standards. However, Web services environment can span organizational boundaries, and hence, its management agents, if implemented, should exhibit social capabilities with a common global standard such as FIPA specifications so that coordination can take place at a level. web services management is a broad term which covers different areas such as

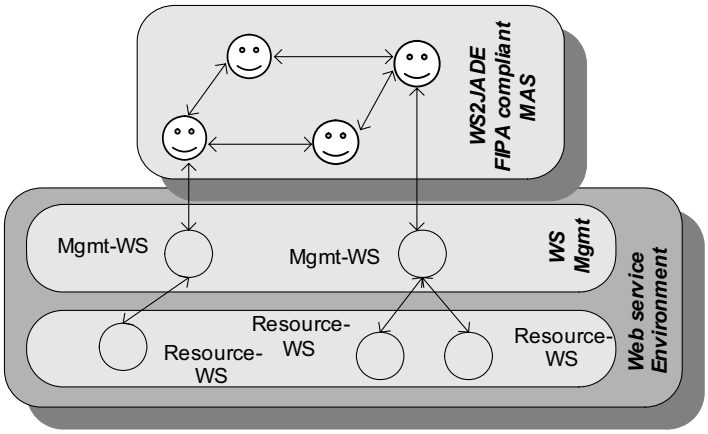

Fig. 6. WS2JADE for WS Management

access mechanism (authentication, authorization, etc), provisioning (SLA management, execution monitoring, etc), and composition (composition structure, conversation relationships, etc). As mentioned before, the MUWS 1.0 specifications define how to represent and access manageability interfaces of resources as Web services. The MOWS 1.0 specification defines how to consider Web services as resources and how to describe and access that manageability interfaces using MUWS. They together provide the Web services managers with one set of protocols and semantic instrumentation to manage Web services based applications and processes across enterprise and organizational boundaries. However, to what extent a manageability interface can be exposed to external managers and how the Web services managers coordinate and manage their Web services efficiently and intelligently are not in the scopes of these specifications. In other words, while these web service management standards define the interfaces needed for managing a web service, they do not specify 'how, 'when' or 'why' a web service should be managed. If we look at the attributes of software agents, we can see that they are often characterized by their 'intelligent and social capabilities' and exhibit autonomous, goal-driven behaviour. Hence multi-agent systems fit into the web service management picture provided they can use the manageability interface. The agents can coordinate, cooperate and negotiate on how much visibility of a management interface should be exposed, and make plans of how to manage Web services. Direct Web service manipulations are supported by management infrastructures which implement WSDM or WS-Management. Figure 6 presents this Web services management model with the support from WS2JADE. In the top layer of WS2JADE and FIPA compliant MAS, available AI techniques can be implemented and distributed among FIPA compliant agents, these agents communicate with WS2JADE agents to get access to the manageability interfaces of manageable Web services in the second layer of Web service Management. They then coordinate and manage the Web services in the bottom layer. Note that these are logical layers. In some implementation, services in the Web service management layer can be the same with services in the bottom layer.

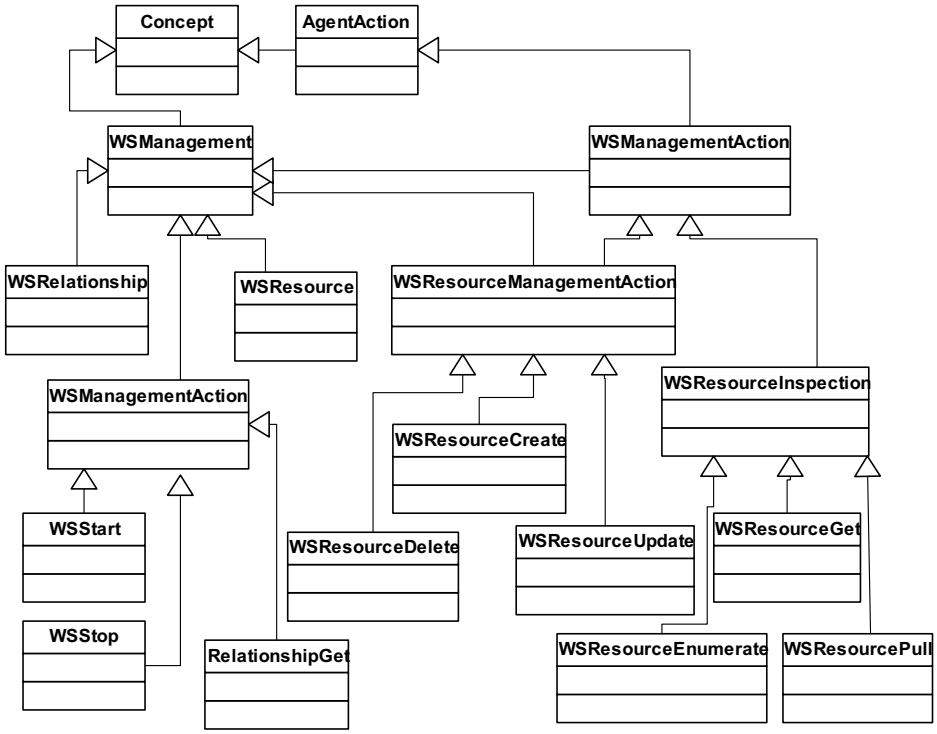

Fig. 7. WS Management Ontology Model

We believe a Web based management of Web services model, such as WSDM or WS-Management, has a great potential to be widely adopted by software vendors in the future. Hence, we have implemented an ontology base for Web services management in WS2JADE. The ontology structure was designed after a careful review of both WSDM and WS-Management specifications to combine common and important management elements in these specifications. Figure 7 partially presents this ontology structure. As illustrated from the figure, WSResource and WSRelationship are important concepts for Web service management. They extend from the WSManagement concept which serves as a root concept of all other content elements. WSResource defines a distinct type of management attributes exposed by a manageable Web service. WSRelationship class represents relationships between two Web services or two Web services resources. At the lowest level, three types of relationships are defined: *includesOf*, *dependsOn* (between two Web services), and *correlatesWith* (between two Web services resources). *"includesOf*" relationship between two Web services means that the first Web service is composite and it has the second Web service in its composition structure. *"dependsOn"* relationship keeps track of relationships hidden by virtualization process. A *"correlatesWith"* relationship indicates a correlation between property values of two Web ser-

vices' resources. WSManagementAction is the root concept where all other management actions subclass from. These management actions include WSResourceCreate, WSResourceDelete, and WSResourceUpdate which correspond to resource management functions of creating, deleting, and setting a resource in WSDM and WS-Management. Also, for resource enumerations, WSResourceEnumerate establishes the resource enumeration context and WSResourcePull iterates over an enumeration result set. Depending on how much control over a service that a manageability interface has, the Web service management ontology can be further extended. FIPA compliant agents and WS2JADE agents use the ontology to communicate on Web service management related information. Whether a direct control of Web service is required, WS2JADE agents translate the control information into management actions available in WSDM or WS-Management. With the help of Web service management ontology and the management model outlined in Figure 6, management reasoning and access for Web service control can be decoupled and handled separately.

5 Demonstration

This section demonstrates the Web service - agent integration capabilities of WS2JADE with simple examples of Web service management including service discovery, composition and deployment with JADE agents. It describes how Web services can be accessed and used by JADE agents and how other agents can use this advantage to build value-added services in composition.

The Find-and-Bind example demonstrates Web services discovery and deployment by JADE agents. A client agent searches for Web services on the UDDI agent in WS2JADE and then accesses and uses these Web services through a newly generated agent. The new agent is created, deployed, and registered in the DF (Directory Facilitator) by WS2JADE at run time. It offers services equivalent to the Web services in terms of functionalities except that they can be used by other agents. Then we demonstrate an example for building a composite service from different Web services with the use of WS2JADE. In our scenario, three agents have a plan on how to compose different Web services to form a valued-added service. It uses WS2JADE to discover Web services and then bind these Web services to the abstract services in the plan. Sample Web services used in this demonstration are Amazon Web Service, PayFriend and GlobalTransports Web service. A client agent can search for and view different items from Amazon, pay for them through PayFried service and order with GlobalTransports Web service.

5.1. Deployment of Web Services as JADE Agent Services

WS2JADE provides a tool for deploying and controlling Web services as agent services. This tool is provided through the combination of the service assignment mapping and the interaction pattern management components. It allows JADE agents to deploy Web services on the fly. The libraries which enable this tool are found in the *cia-*

mas.wsjade.management package while the main class for the graphical user interface is *ciamas.wsjade.management.utils.Admin*. The deployment of Web services with WS2JADE is explained in the remainder of this section.

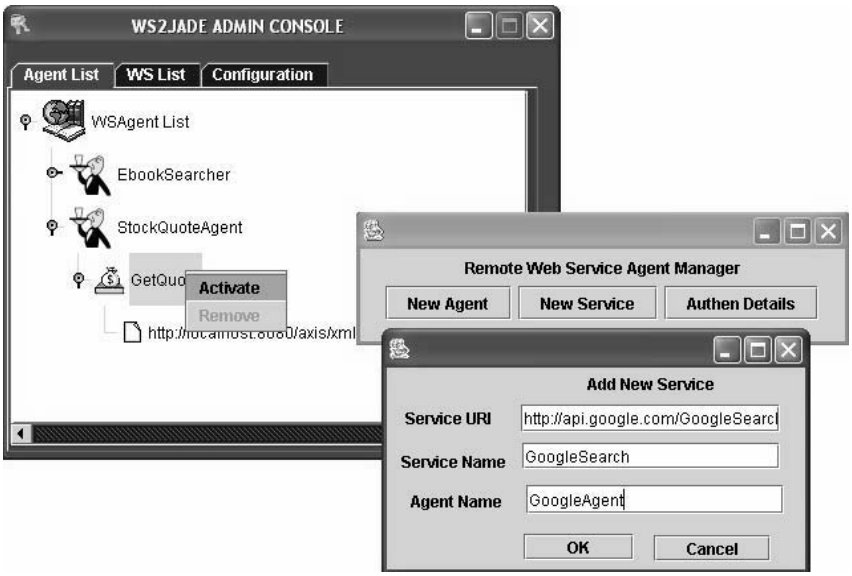

Fig. 8. WS2JADE Admin Console showing service deployment interfaces.

At first, the gateway agent container is started and the WSAG agents are deployed on it. Once the WSAG agents have been created, they are ready to deploy Web services as agent services. After the Web services have been deployed as agent services, they are ready for use by client agents. Each time a Web service is deployed as a JADE agent service, ontology packages are generated and compiled. These ontology packages can then be sent to a public Web server where they are available for download and use by the client agents. As mentioned before, at this stage ontologies are available as Java classes (JADE native ontology) and as OWL files.

Since the WSAG (essentially JADE) agents are multithreaded, they can offer/service multiple Web services at the same time. During the deployment of these Web services, each service is assigned a different agent-service name. The agent-service name can be the same as the Web service. If two WSAG agents deploy the same Web service, they must use different names. The screen shots in Figure 8 show two WS2JADE consoles: one is for local and the other is for remote management. Local management means that WS2JADE's WSAG agents are deployed to run on the local machine. Remote management allows an administrator to activate/deactivate the WSAG agents and control their services remotely. Remote management also requires correct authentications. Authentication details are stored in *.authen* file under WS2JADE root folder. As can be seen from the screenshot, in the left console for the local management, the agentList tab

shows the deployed WSAG agents named *EbookSearcher, StockQouteagent* along with the services they provide namely *Ebook* (hidden) and *GetQoute* services. The WSList tab (its content is invisible in the screenshot) shows the list of deployed Web services along with their WSDL address. The generated ontology for each Web service in OWL format can be viewed by clicking on the agent providing the service. In the right console for the remote management, the Google Web service at http://api.google.com/GoogleSearch.wsdl is deployed as a service of a WSAG agent named GoogleAgent.

5.2 Discovery of Web Services

The Universal Description, Discovery and Integration (UDDI) specifications define a way to publish and discover information about Web services. Actually the UDDI itself is a Web service which provides SOAP interfaces for publishing Web services and querying about Web services. The UDDI inquiry interface in WSDL format is published by UDDI.org and can be found at http://www.uddi.org. The UDDI registry can be used to search for services, service providers or tModels by name or by browsing categorizations. Since the UDDI itself is a Web service, it can be deployed as an agent service with WS2JADE. Once this is done, JADE agents can then query this agent service for services or service providers dynamically. Whenever a new Web service is registered with the UDDI registry, it can be discovered by the WSAG agent upon inquiry by a client JADE agent.

The interaction flow is illustrated in the Figure 9. From the client agent side, it needs to do a search on UDDI agent (step 1) for a wanted Web service (e.g. Google). After getting search results back (step 2) from UDDI agent, the client agent, based on its own reference, determines a service it wants to use and queries the WSManager agent on how to use this Web service. The WSManager agent informs the client the address of the agent which can offer a proxy service of this Web service (step 4). The client now can start to use the service (step 5 and 6). To examine what happen inside WS2JADE, as mentioned before, WS2JADE proxies the MS UDDI (step 1A) through UDDI agent as default to fulfil the client request at step 1. After step 3, the WS Manager creates a new agent and deploys a new agent service that is a proxy of the wanted Web service. The WS Manager also registers this agent on the DF. The address of this agent is returned back to the client agent. WS invocation is again done indirectly through the proxy service of the newly generated agent. The UDDI agent can also be configured to use UDDI servers different than Microsoft's one.

The screenshots in Figure 10 show the interface for a client Agent which can do the search for a service with key words for the service name, business name, or tModel. This WSAG agent has deployed the Microsoft test UDDI which is available at http://test.uddi.microsoft.com and hence, clients can query this UDDI agent for details about different services and service providers. As can be seen, a search for services starting with Google returned 4 results.

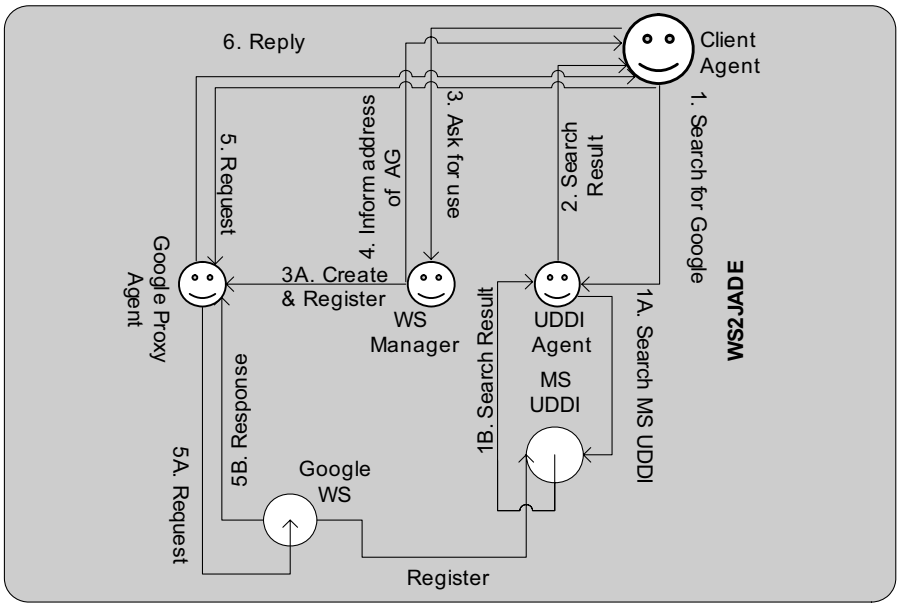

Fig. 9. Find and Bind

The client agent now can use the remote service deployment capability of WS2JADE to request for a wanted Web service to be deployed as an agent service. This can be done by making a request to the Manager agent as shown in figure 8 as explained previously. The Manager agent, after successfully deploying the Web service, informs the client agent the address of the targeted agent and the location of the ontology used to access the service.

5.3 Composition of Web Services

A user can enter keywords that are used in search requests on Amazon and can add items returned in the search results to a remote shopping cart. When they have finished adding items to the shopping cart they can pay for the items and organize delivery. This demonstration is a next step of the previous one. We have a scenario in which three agents: P , A, and G, have a composition plan for offering online item purchase. Such a plan needs to take into account online payment transaction and product delivery. In the plan, P is responsible for client payment. A is responsible for shopping cart. G is responsible for item delivery. The interaction sequence is depicted in Figure 11. First, the client agent searches for the products and add them to its shopping cart (step 1 and 2, repeated). Once the client agent does a checkout, agent A informs it payment details with agent P. The client agent then contacts P to do payment. After the payment is made, agent P informs agent A whether the payment is successful (step 6.1). If it is, agent A sends a message to agent G and asks for item delivery.

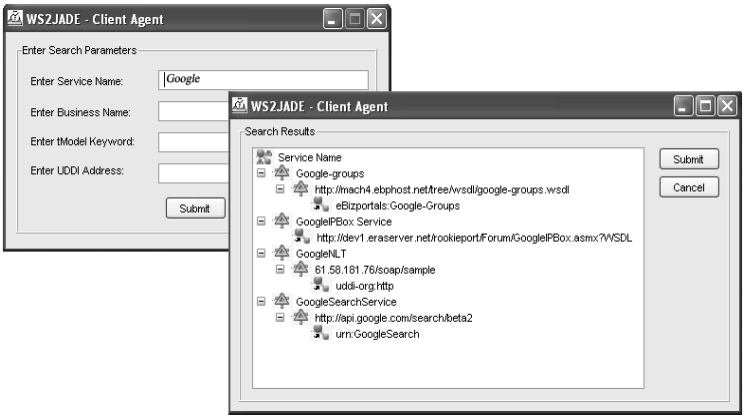

Fig. 10. Screenshot of a client interface querying the UDDI WSAG agent to search for services

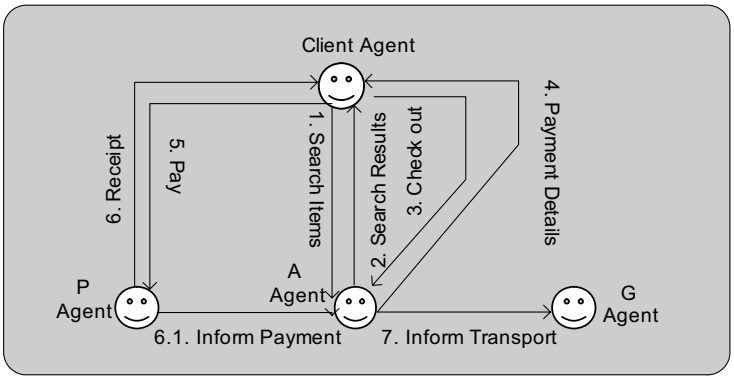

Fig. 11. Composition Plan

At the abstract level, no concrete implementation of services is described in the plan. Because the main purpose of this demonstration is to show how Web services can be invoked and composed by agents, we focus on the interaction sequences only and assume that the plan is encoded in some format that the agents can understand. The services these agents are going to use are Web services. The agents P, A, and G, based on the requirements of their own services, try to find and bind Web services. How this can be done with WS2JADE is explained in the first demonstration. Hence, we skip these steps and assume that after find-and-bind steps, agent P becomes a proxy of PayFriend Web service, agent A becomes a proxy of Amazon Web service, and agent G becomes a proxy of GlobalTransport Web service as depicted in Figure 12. The composite service now can now be executed. Since this is merely a demonstration, transactions are preferably not committed. PayFriend Web service and GlobalTransport Web service are developed by us to emulate essential functionalities in the interfaces of PayPal and

Global Transport Web services, however, without real transactions to any banks. PayFriend and GlobalTransport Web services and Amazon Web service can be found at:

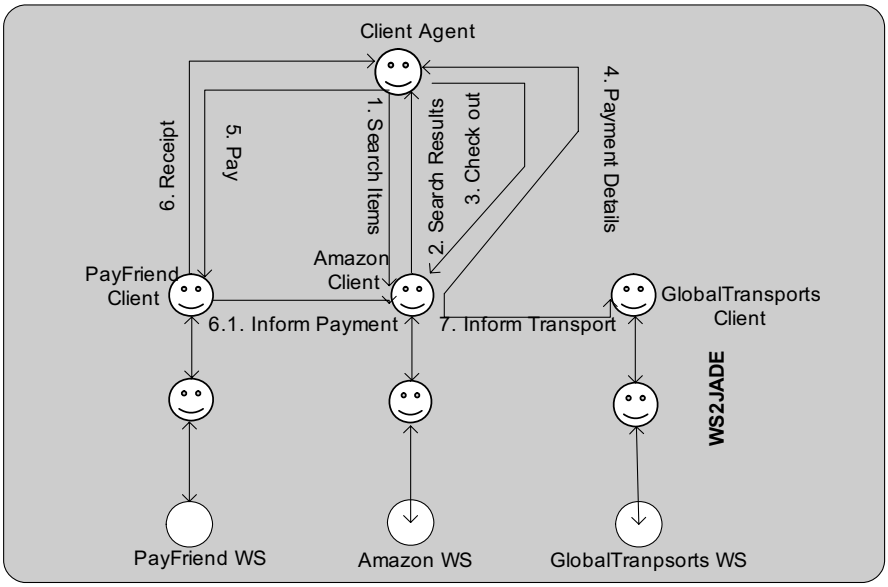

Fig. 12. Composition Service

PayFriend Address:
http://mercury.it.swin.edu.au:8080/axis/services/PayFriend?wsdl
GlobalTransport Address:
http://mercury.it.swin.edu.au:8080/axis/services/GlobalTransports?wsdl
Amazon Web Service Address:
http://soap.amazon.com/schemas2/AmazonWebServices.wsdl

Additional information of PayPal and GlobalTransport Web services are provided in the links below. To use the PayPal service, users need to subscribe to PayPal developer network for a developer key.

PayPal Web service:
https://www.paypal.com/cgi-bin/webscr?cmd=p/pdn/devcentral_landing-outside
Global Transport Web service:
http://transportal.russia.webmatrixhosting.net/default.aspx?static=webservices

The screenshots in Figure 13 show the enactment of a composite Web service. The sequences can be explained as following. A user can enter keywords that are used in search requests on Amazon and can add items returned in search results to a remote

shopping cart. When they have finished adding items to the shopping cart they can pay for the items and organize delivery. The screenshots in Figure 13 and 14 show the result of a composite service invocation by the Client agent. The user clicks 'Search' triggering the Client to send a request to the Amazonagent to search Amazon. The Amazonagent then forwards the request to AmazonProvider who handles the Web service invocation calls. A list of results is then returned to the client via the Amazon agent. A user can then add items to the shopping cart, which triggers another request to the Amazon Web service resulting in the contents of a remote shopping cart being updated with the selected item. Clicking 'Checkout' causes the Client to send a request message to the PayFriend agent. After PayFriend has completed the payment transaction it sends an acknowledgement to the Amazon agent that forwards a request to the GlobalTransports agent who invokes the web service call that organises delivery of the purchased items. A receipt message from the PayFriend agent and delivery acknowledgment message from the GlobalTransports agent are then sent to the Client agent and the GUI updated with dialogs indicating to the user that the transaction is complete.

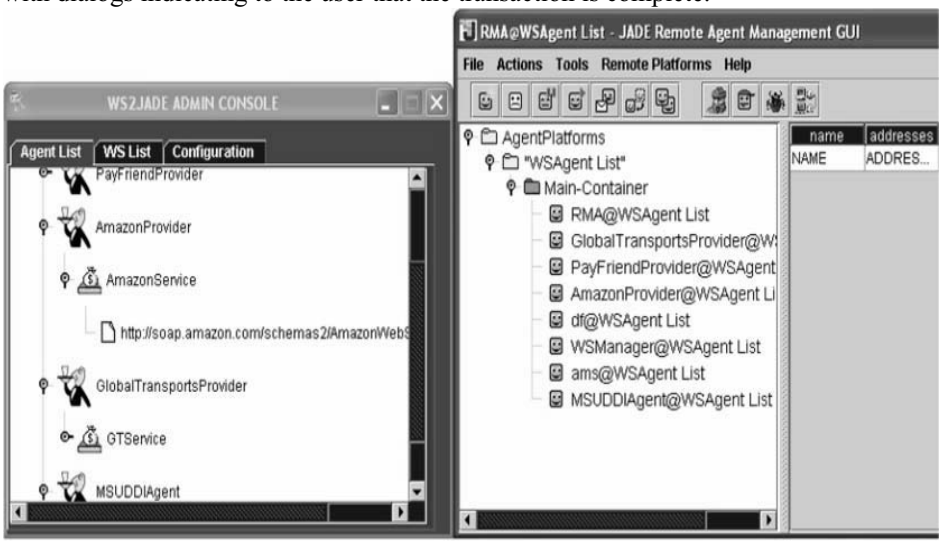

Fig. 13. Screenshot showing a composite service offered by different WSAG agents

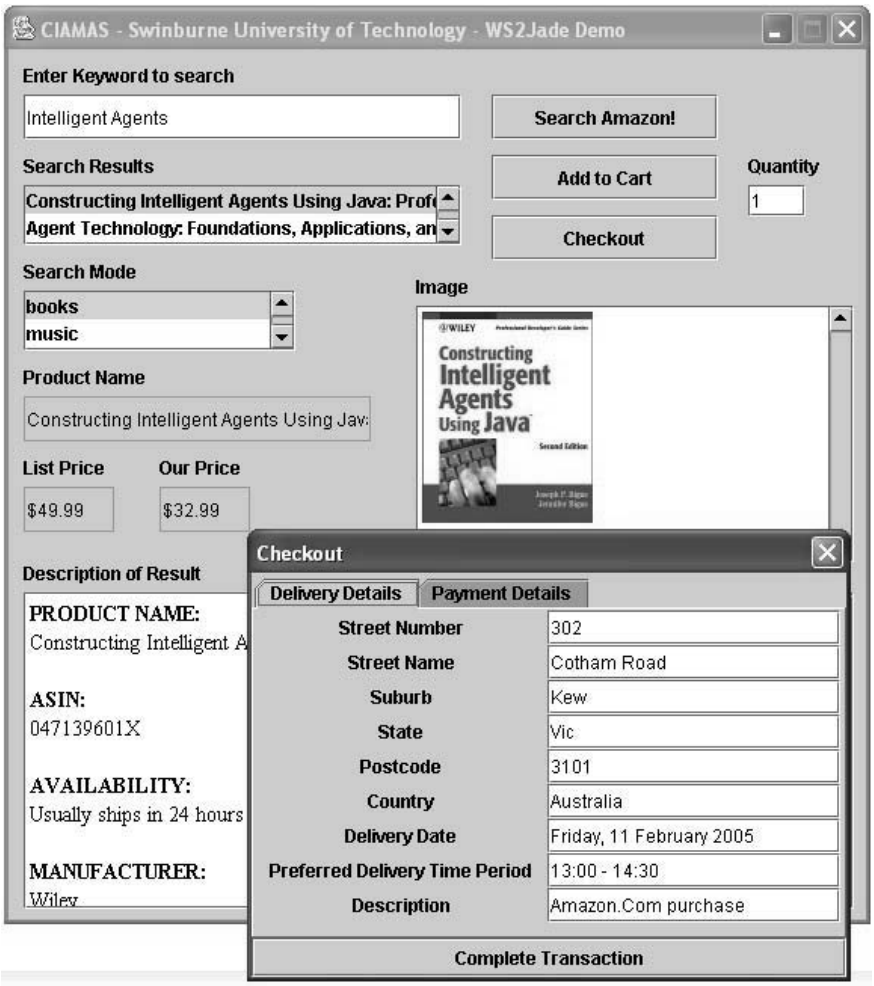

Fig. 14. Screenshot showing the result of a composite service invocation by the Client agent

6. Conclusion

The paper presents a framework and toolkit called WS2JADE for integrating Web services and JADE agents. In contrast to other solutions like WSDL2JADE or WSIGS, WS2JADE provides facilities to deploy and control Web services as agent services at run time for deployment flexibility and active service discovery. WS2JADE also provides a management ontology base for integrating with any future implementations of emerging distributed WS management standards which employ Web services such as

OASIS's WSDM. The future of intelligent agents with autonomous capabilities, which manage and access the widespread Web services infrastructure, is promising. WS2JADE demonstration with simple examples of service discovery, composition and deployment with agents shows how the toolkit achieves first steps in that direction. It demonstrates how Web services can be accessed and used by JADE agents and how other agents can take this advantage to build value-added services.

Since Web services is a volatile area with rapid changes and many specifications in WS-* domain (such as WS-Agreement Specification, WS-Resource Framework) need to be implemented and tested out, WS2JADE has been designed to accommodate future plug-in components. However, there are still software features and design areas that require new solutions, implementations or further improvements. In particular, Web services negotiation, semantic processing, and full integration with WSDM specification will be our next focus for WS2JADE. Web services negotiation will give WS2JADE agents the ability to read, understand contracts, and employ negotiation mechanisms to contract Web services and their compositions according to WS-Agreement specification. Semantic processing capability will improve the ontology management component in structuring ontology trees and possibly merging related ontologies. We are looking forward for a wide adoption of WSDM or WS-Agreement specifications for a global level of WS management with FIPA compliant agent systems. We hope that WS2JADE can be applied in more real-world examples to make contribution into agent-based Web services and business process management in particular and practical applications of multi-agent systems in general.

Acknowledgments

This work is part of the Adaptive Service Agreement and Process Management in Services Grid project CG060081. This 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 A., Arora, et al, Web Services for Management (WS-Management) available at http://www.intel.com/technology/manage/downloads/ws_management.pdf accessed on 6th April 2005.

2 A., Banerji, et al, Web Services Conversation Language (WSCL) 1.0, available at http://www.w3.org/TR/2002/NOTE-wscl10-20020314/ accessed on 25th April 2005

3 agentLink, European Co-ordination Action for agent-based computing, available at http://www.agentlink.org/ accessed on 25th April 2005

4 Agentcities Web Services Working Group, "Integrating Web services into agentCities", Technical Recommendation available at http://www.agentcities.org/rec/00006/ accessed on 25th April 2005

5 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, Springer, pp. 207-221.
6 WS2JADE, Web services to agents http://www.it.swin.edu.au/centres/ciamas, accessed on 24th April 2005.
7 D. Greenwood, M. Calisti, "An Automatic, Bi-Directional Service Integration Gateway", In The 1st International workshop on Web Services Agent-Based Engineering (WSABE' 2004) held in conjunction with The 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems, New York, USA, 2004. Available at www.agentus.com/WSABE2004/program/, accessed on 25th April 2005.
8 D. Greenwood, M. Calisti, "Engineering web service - agent integration", IEEE Systems, Cybernetics and Man Conference, the Hague, Netherlands, Oct, 2004, pp.1918-1925
9 E. Christensen et al., Web Service Description Language (WSDL 1.1), available at http://www.w3.org/TR/wsdl , accessed on 5th April 2005.
10 E.M. Maximilien and M.P. Singh, "Agent-based architecture for autonomic web service selection". In The 1st International Workshop on Web Services and Agent-based Engineering (WSABE'2003) held in conjunction with The 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, Melbourne, Australia, 2003. Available at www.agentus.com/WSABE2003/program/maximilien.pdf, accessed on 25th April 2005.
11 F., Cheng, H., Guo, B., Xu, "Agentification for Web Services", Proc. 28th Annual International Computer Software and Applications Conference (COMPSAC'04) ,Hong Kong, Sept 2004, pp.514-519
12 F., Ishikawa, N., Yoshioka, Y., Tahara, S., Honiden, "Mobile agent System for Web Services Integration in Pervasive Networks", International Workshop on Ubiquitous Computing (IWUC 2004), April, 2004, Porto, Portugal, pp.38-47
13 F., Ishikawa, N., Yoshioka, Y., Tahara, S., Honiden, "Toward Synthesis of Web Services and Mobile agents", In Proc. of the 2st International Workshop on Web Services and agent Based Engineering (WSABE'04), New York, 2004. Available at http://www.agentus.com/WSABE2004/program/.
14 Foundation for Intelligent and Physical agents. Interaction Protocol Specification, http://www.fipa.org/repository/ips.php3
15 H. Ludwig, "Web Services QoS: External SLAs and Internal Policies: Or, How Do We Deliver What We Promise?", Proc. 4th IEEE Int'l Conf. Web Information Systems Eng. Workshops, IEEE CS Press, 2003, pp. 115–120.
16 I., Foster, N.R., Jennings, C., Kesselman, "Brain Meets Brawn: Why Grid and agents Need Each Other", The Third International Joint Conference on Autonomous agents and Multi agent Systems, AAMAS'04, July, 2004, New York, USA, pp.8-15.
17 J.,Beatty, et al., Web Services Dynamic Discovery (WS-Discovery) available at http://msdn.microsoft.com/ws/2004/10/ws-discovery/, accessed on 24th April 2005.
18 J., Cao, et al., Composing Web Services Based on agents and Workflow, M. Li et al., (Eds.), GCC2003, Springer-Velag Berlin Heidelberg, 2004, pp 948-955.
19 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, June, 2003, pp. 626-635.
20 M., Luck, P., McBurney, C., Preist, "Agent Technology: Enabling in Next Generation Computing", Sections 4.5.2, agentLink, 2003, pp. 23-26.
21 McAfee, McAfee® ePolicy Orchestrator®, available at http://www.mcafeesecurity.com/au/products/mcafee/mgmt_solutions/epo.htm, accessed on 24th April 2005.

22 N., Catania et al. An Introduction to WSDM-MOWS and WSDM-MUWS available at http://devresource.hp.com/drc/specifications/wsdm/index.jsp, accessed on 24th April 2005.
23 N. Cavantzas et al. Web Services Choreography Description Language Version 1.0, http://www.w3.org/TR/2004/WD-ws-cdl-10-20041012/, accessed on 24th April 2005.
24 OASIS TC, Web Services Distributed Management: Management Using Web Service (MUWS 1.0) Part 1 & 2, OASIS standard at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm, accessed on 24th April 2005.
25 OASIS TC, Web Service Distributed Management: Management of Web Services (WSDM-MOWS) 1.0, OASIS standard at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm, accessed on 24th April 2005.
26 OpenNet Test-bed Initiatives: http://x-opennet.org/
27 P. Buhler, J.N. Vidal, and H. Verhagen, "Adaptive workflow= web services + agents", In Proc. of the International Conference on Web Services, Las Vegas, U.S.A, July 2003, pp. 131-137.
28 P. Buhler, J.N. Vidal, and H. Verhagen, "Enacting BPEL4WS Specified Workflows with Multi-agent Systems" , In Proc. of the 2st International Workshop on Web Services and agent Based Engineering, New York, 2004. Available at www.agentus.com/WSABE2004/program/, accessed on 24th April 2005.
29 P. V. Biron, K. Permanente, and A. Malhotra, XML Schema Part 2: Data types Second Edition, http://www.w3.org/TR/xmlschema-2/, accessed on 24th April 2005.
30 Sun Microsystems, Inc. Java API for XML-Based RPC (JAX-RPC), available at http://java.sun.com/xml/jaxrpc/index.jsp
31 Sun Microsystems, Inc. Java Management Extension (JMX), available at http://java.sun.com/products/JavaManagement/, accessed on 24th April 2005.
32 Telecom Italia Lab. JADE (Java Agent Development Framework), available at http://sharon.cselt.it/projects/jade/, accessed on 24th April 2005.
33 Whitestein Information Technology Group AG. Web services agent Integration Project available at http://wsai.sourceforge.net/index.html, accessed on 24th April 2005.
34 World Wide Web Consortium, RDF Vocabulary Description Language 1.0: RDF Schema, available at http://www.w3.org/TR/rdf-schema/, accessed on 24th April 2005.
35 World Wide Web Consortium, OWL Web Ontology Language Overview available at http://www.w3.org/TR/owl-features/, accessed on 24th April 2005.
36 World Wide Web Consortium, Web Services Choreography Description Language Version 1.0, available at http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/, accessed on 24th April 2005.
37 X., Gu, Klara N.N., Chang, C., Ward, “QoS-Assured Service Com-position in Managed Service Overlay Networks”, in Proc. of 23rd IEEE International Conference on Distributed Computing Systems (ICDCS 2003), Providence, Rhode Island, May, 2003, p.194.

Information about Software

Software is available on the Internet as

(x) prototype version
( ) full fledged software (freeware), version no.:
( ) full fledged software (for money), version no.:
( ) Demo/trial version
( ) not (yet) available

Internet address*:*

Description of software:
http://www.it.swin.edu.au/centres/ciamas/tiki-index.php?page=ws2jade-proj
Download address:
http://www.it.swin.edu.au/centres/ciamas/tiki-index.php?page=ws2jade

Contact person for question about the software:

Name: Xuan Thang Nguyen
Email: xnguyen@ict.swin.edu.au

Xuan Thang Nguyen
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia.
e-mail: xnguyen@ict.swin.edu.au

Ryszard Kowalczyk
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia.
e-mail: rkowalczyk@ict.swin.edu.au

Mohan Baruwal Chhetri
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia.
e-mail: mchhetri@ict.swin.edu.au

Alasdair Grant
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia.
e-mail: 4103515@swin.edu.au