

Integrating Agent Services into BPEL4WS Defined Workflows

Paul A. Buhler¹ and José M. Vidal²

¹ College of Charleston, Dept. of Computer Science,
66 George Street, Charleston, SC 29424, USA
buhlerp@cofc.edu

² University of South Carolina, Computer Science and Engineering
Columbia, SC 29208, USA
vidal@sc.edu

Abstract. In the future, Web services and software agents will coexist in the Business Process Management solution space. In our vision, agents will have symbiotic relationships with Web services, harnessing them as externally defined agent behaviors. In this paper we detail the development of a demonstration system that transparently integrates agent services into BPEL4WS defined workflows. The development of the demonstration system illustrates the power of compositional approaches to system creation. It also serves to reinforce the importance of open standards, since the integration of the separate components is dependent upon the interoperability that standards provide. This work is an important first step toward fully integrated agent-based workflow management systems.

1 Introduction

Business processes, which are rooted in the concept of workflows, are essentially social networks; as such, they lend themselves to analysis via agent-based simulation. Recently, several large corporations, such as Procter & Gamble, Southwest Airlines, Merck, and Ford Motor Company have touted the benefits of agent-based simulations which have identified ways to optimize their operations. In these simulations, software agents represent the individual components of the system. The agent's behaviors are modeled after their real-world counterparts. After validating the accuracy of the simulation, by comparing its performance to the real-world system, individual agent's behavior rules can be modified to assess the impact of the change on the system. Procter & Gamble claims that agent-based simulations have been used to identify optimizations of its supply chain, which are saving the company US \$300 million annually [1].

Currently, two trends are changing the way businesses interact with their environments. The first of these trends is the incorporation of real-time data into business processes. Corporate leaders believe that having the ability to adapt their processes in

near real-time will provide a competitive edge; however, the introduction of environmental dynamics may simply destabilize business processes because the sociality of the business process is not typically recognized. The second trend is the dynamic realignment of business partners enabled by advances in information technology. The need for adaptive processes is being driven by the demands of e-commerce in both B2B and B2C spaces.

Initial B2B automation activities were centered on Electronic Data Interchange (EDI) initiatives. More recent work in the B2B space has focused on the development and deployment of ebXML (electronic business XML). With both EDI and ebXML the collaborating business partners predefine the terms of their electronic interaction. As discussed by Jenz, these technologies enforce regulated B2B interaction and as such, they create closed communities of business partners. [2]. In comparison, views toward virtual organizations require flexible, on-the-fly alignment of business partners; in other words, adaptive workflow capabilities. These loose collaborations of business partners operate in open, non-regulated B2B/B2C scenarios where pre-negotiated collaboration agreements are a hindrance in these environments [2].

Business process management software is gaining momentum due to the emergence of a de facto standard for describing a business process as compositions of Web services. This standard is named BPEL4WS, which is an acronym for Business Process Execution Language for Web services [3]. In our earlier works [4, 5],[6],[7] we have explored the relationship between Web services, Multiagent Systems (MAS), and workflows. Our vision is to create adaptive workflow capability through decentralized workflow enactment mechanisms that combine Web service and agent technologies.

The applicability of MAS to workflow enactment has previously been noted [8]; however, it is only recently that the notion of using passive Web services as externally defined behaviors of proactive agents has become palatable. Besides differentiating Web services and agents based upon a measure of proactivity, there are several other important distinctions worth noting. Some of the distinguishing characteristics provided by Huhns are: Web services know only about themselves, they do not possess any meta-level awareness; Web services are not designed to utilize or understand ontologies; and Web services are not capable of autonomous action, intentional communication, or deliberately cooperative behavior [9]. In contrast, agents possess all of these capabilities.

This paper discusses the first step toward moving agents out of the simulation environment and injecting them into the workflow itself. To facilitate this discussion, an example BPEL4WS process is introduced. This sample process serves as a running example throughout the rest of the paper. Next, the major components of the software infrastructure, which allows the integration of the agents into the workflow, are described. The paper proceeds to discuss the end-to-end demonstration of the workflow engine calling an agent service. The paper concludes with a summary of its contribution and a discussion of future directions for this work.

2 An Example BPEL4WS Workflow

BPEL4WS is a Web service composition language; as such it allows the specification of a collection of Web services and the coordination of their interaction. When thought of from a business process perspective, BPEL4WS can be said to be a process description language suitable for defining workflows where the activities to be performed consist of Web service invocations. A thorough explanation of BPEL4WS is beyond the scope of this paper; however, the following sample is provided to help the uninitiated develop an intuition about BPEL4WS. Please note that namespace and variable information has been removed from the example in an effort to simplify its presentation.

```
<process
  name="stockLookupProcess">
  <partners>
    <partner name="requestor"/>
    <partner name="stockQuoteProvider"/>
    <partner name="currencyExchangeProvider"/>
    <partner name="simpleFloatMultProvider"/>
  </partners>
  <variables>
    ...
  </variables>
  <sequence name="main">
    <receive name="request"/>
    <flow name="getQuoteandExchangeRate">
      <invoke name="getStockQuote"
        operation="getQuote"
        inputVariable="stockQuoteRequest"
        outputVariable="stockQuoteResponse">
      </invoke>
      <invoke name="getExchangeRate"
        operation="getRate"
        inputVariable="currencyExchangeRateRequest"
        outputVariable="currencyExchangeRateResponse">
      </invoke>
    </flow>
    <invoke name="multiplyFloat"
      operation="multiply"
      inputVariable="simpleFloatMultRequest"
      outputVariable="simpleFloatMultResponse">
    </invoke>
    <reply name="response"
      operation="requestLookup"
      variable="response">
    </reply>
  </sequence>
</process>
```

The basic functionality of this workflow is to provide a stock quote that is converted into the requestor's local currency. If invoked with the arguments "IBM" and "Switzerland" the stock quote for IBM would be converted into Swiss Francs before being returned. This workflow has four participating partners: the service requestor,

who requests the execution of the workflow; the stock quote provider, a Web service that provides delayed stock quotes; the currency exchange rate provider, a Web service that provides exchange rates between countries; and the simple floating point multiplication provider, a Web service that multiplies two numbers and returns their product. Figure 1, provides a graphical view of the structure of the workflow in Use Case Maps (UCM) notation [10]. UCM is intuitive; the line represents the thread of control, which passes through the partners of the workflow. The workflow process starts at the end of the line designated with a ball, which corresponds with the `<sequence>` tag in the process definition. The process ends at the end capped by a line, this aligns with the `</sequence>` tag. Tracing this line from start to finish provides an accurate account of the temporal ordering of the workflow's activities. Notably, the line splits and joins in the middle of the process, this corresponds to the `<flow>`, `</flow>` tags respectively. In BPEL4WS activities found inside a flow block are executed concurrently.

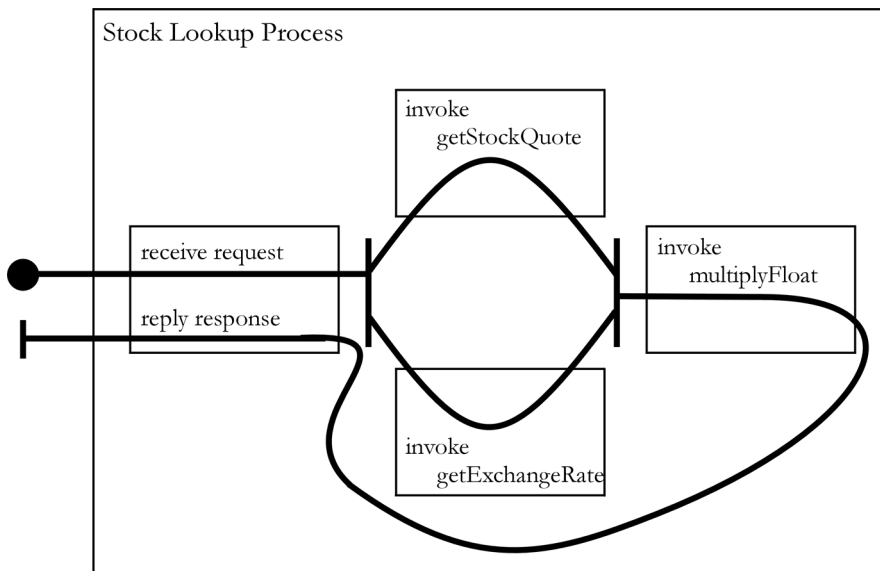


Fig. 1. A Use Case Maps model of the example BPEL4WS, which depicts the ordering of the activities in the workflow.

3 Infrastructure

In order to demonstrate the integration of an agent into a workflow, a platform to carry out this research was needed. The major components of the platform are the BPWS4J Editor and Engine 2.0 [11], the Web service Agent Gateway (WSAG) 1.0 [12], and JADE (Java Agent DEvelopment framework) 3.1 [13]. In addition to these primary components, the following are also used: J2SE 1.4.2, Eclipse 2.1, Tomcat

4.1.29, Axis 1.1, and webMethods Glue Standard Edition 4.1.2. Each of the major components will be briefly discussed in subsequent sections.

3.1 BPWS4J

BPWS4J is the common name for the IBM Business Process Execution Language for Web services Java Run Time. BPWS4J provides two essential functions to the infrastructure. The first contribution is the BPWS4J Editor, which is a plug-in for the Eclipse environment. The Editor allows for the graphical creation of a BPEL4WS defined workflow. Having the capability to diagrammatically define the workflow is helpful because it is unwieldy to write programs in XML. The second essential function is the BPWS4J Engine, which provides an enactment service for BPEL4WS workflows. The engine consumes a workflow specification and deploys it as a Web service. When the Web service is invoked, the underlying workflow executes.

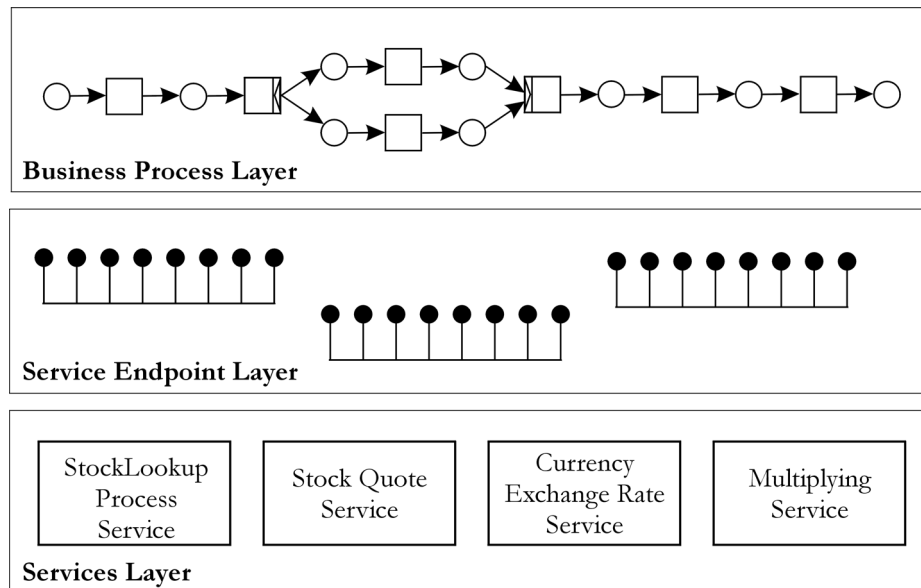


Fig. 2. BPWS4J interacts with the constituent Web services through a layered model.

The BPWS4J's Engine coordinates the activities that occur in the Business Process Layer (see Figure 2). BPWS4J enacts the business process as encoded in the BPEL4WS file. BPWS4J locates and binds to the service endpoints based upon information found in the user supplied WSDL files. In Figure 2, the middle layer contains the service endpoints and the services are found in the lowest layer. The Web services are insulated from their respective endpoints by Web service deployment middleware. Although the business process is separated from the Web services, they are relatively tightly coupled due to the dependence on explicit operation and message syntax embedded in the BPEL4WS workflow description.

The communication between the layers identified in Figure 2 occurs in a bi-directional and synchronous manner. Layered models with synchronous communication channels are characteristic of distributed applications [14]. BPWS4J coordinates the execution of workflow as a distributed application; thus it can be concluded that BPWS4J is not an application integration platform. Although the difference between distributed applications and application integration is subtle, it is fundamental to our desire to develop decentralized agent-based workflow enactment mechanisms.

Agents can be viewed as independent applications that provide services to one another through loosely coupled, asynchronous message exchange. Agents are able to take advantage of the non-blocking nature of their messaging by overlapping other processing with their communicative acts. The agent uses its autonomy to determine what work to perform; however, we can envision an agent searching for ways to optimize the workflow in which it is engaged. This might occur through finding other service partners that provide better quality of service, or learning from its interaction histories with existing partners so as to maximize the utility of their future interactions.

3.2 WSAG

The WSAG is a partial implementation of the Agentcities Web services Working Group's Technical Recommendation on the integration of Web services into the Agentcities/openNet platform [15]. Gateway agents reside in the WSAG and are responsible for managing conversations with target agents. The WSAG provides the capability to pass a Web service invocation request through a gateway agent to a target agent. The target agent services the request and responds back through the gateway to the Web service client. Thus the WSAG functions as a translator between SOAP message traffic and ACL (Agent Communication Language) based communicative acts (see Figure 3).



Fig. 3. The WSAG enables messages to flow between Web services and agents. ACL messages are exchanged asynchronously, whereas SOAP message exchange occurs synchronously. Copyright © 2002-2003 Agentcities Task Force (ACTF). All Rights Reserved.

The WSAG software consists of two related components: one, a set of tools which greatly assist in the generation of gateway agents; and two, a gateway configuration and deployment application that runs within a Tomcat servlet container. To develop a gateway agent the developer must first write a Java interface, which defines the parameters required by the target agent service. The code generation tools use the Java interface definition to generate the gateway agent. The developer is only responsible for writing the Java code that moves data to/from the content area of the ACL mes-

sages used to communicate with the target agent. Once the gateway agent is created, the management functions of the WSAG software are used to deploy the gateway agent and associate it with the target agent.

3.3 JADE

JADE is a popular FIPA compliant, Java-based agent development platform. FIPA (Foundation for Intelligent Physical Agents) is a consortium that produces standards to enhance the interoperability of heterogeneous agents [16]. The WSAG uses JADE for the implementation of the gateway agents. The target agents in the demonstration system were also constructed with JADE; however, any FIPA compliant agent toolkit would work.

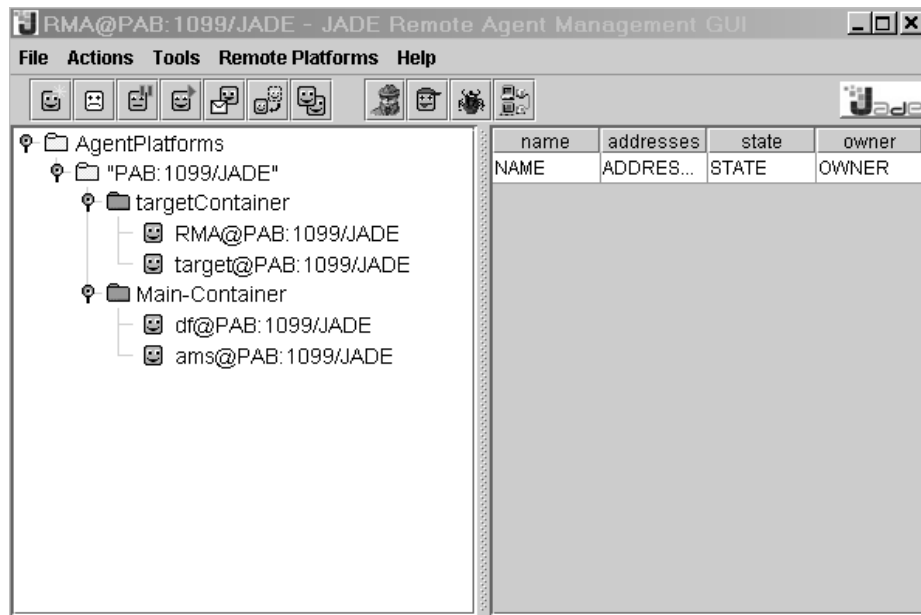


Fig. 4. Jade's Remote Agent Management utility provides utilities for interacting with agents and managing the agent platform.

JADE implements the FIPA reference model for agent platforms. Figure 4 depicts JADE's Remote Agent Management utility. The figure shows that one agent platform with two containers is executing. The main container supplies basic services to the agent platform. The Directory Facilitator (DF) provides yellow page services to the agents running on the platform. The DF also provides the mechanism for agents to advertise their services in the agent directory. The Agent Management System (AMS) provides services to the agent platform that allow the creation, deletion and migration of agents.

4 End-To-End Demonstration

An end-to-end demonstration was performed utilizing the software detailed in the previous section. This demonstration shows the feasibility of injecting an agent into a workflow executed by the BPWS4J Engine. The concept is to use the WSAG to slide an agent between the enactment mechanism and a destination Web service. The example workflow introduced in Section 2, was used for the demonstration. In the demonstration, the BPWS4J engine passes control to the target agent, which in turn calls the Stock Quote Web service. The target agent can be seen running in the target container in Figure 4.

Since the workflow calls the target agent instead of the Web service directly, an opportunity is created to do something intelligent. One can imagine many ways in which this could be useful. Perhaps the Stock Quote Web service guarantees the quote it provides is current within the past 15 minutes and it charges a micropayment for service access. The agent might check a local cache of interaction histories with the Web service to see if the requested symbol has been quoted within the past few minutes. If so, the agent may choose to return the cached quote value instead of calling the service, thus saving the cost associated with the invocation. If the agent were made aware of the stock exchange schedule, it could use its cache for quote requests that occur after the closing bell, and on weekends and holidays. Of course many other possibilities exist when semantic service matching is considered.

4.1 The Software Development Process

It should be clear the demonstration system was assembled with a development process that was primarily compositional, as opposed to creational. Compositional software development methodologies are by necessity different than those used for bespoke software development. The act of composition requires an iterative process that contains the following ordered steps: identification, selection, installation, integration and evaluation.

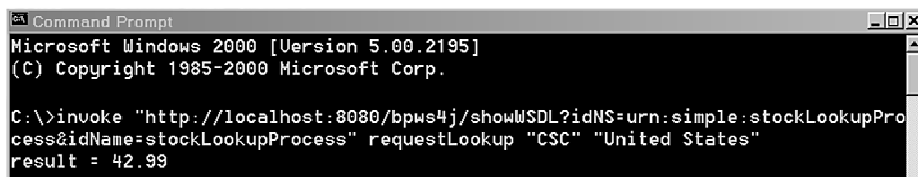
In the identification stage, the marketplace is scanned for relevant components. The most promising candidates are selected based upon criteria that not only include a measure of requirements fit, but also cost, support, and supplier stability. The selected components are then installed and reconciled with the system dependencies of the other components within the solution space. After the components are installed, they must be integrated via configuration and customization. Finally, once the system is assembled, it can be evaluated.

It is important to realize that the only part of the system owned by the project is its architecture [17]. The architecture needs to account for the future evolution of the system. This evolutionary dynamic cannot be ignored as new products continuously enter and leave the solution space. Functionally, the architecture of the demonstration systems relies upon workflow design and enactment tools, Web service/agent integration technology, agent construction tools, and miscellaneous supporting infrastructure. Although the overall system is integrated, each of these architectural components can be evolved separately given that their integration points remain consistent.

Open standards play an important role in the stability of a composed software system. The demonstration system depends upon components that adhere to workflow specification standards, Web service standards, and agent standards. BPEL4WS provides the de facto standard for workflow specification, when each of the activities in the workflow can be accessed as a Web service. The core Web service standards of WSDL, SOAP, and HTTP ensure Web service interoperability. In the agent space, the FIPA standards define the basic services that need to be supplied by compliant agent platforms. Adherence to the FIPA standards enables agents from heterogeneous sources to assemble in open systems.

4.2 Putting the Pieces Together

For demonstration purposes, it is important to establish that the existing workflow is executing successfully, before altering the system to incorporate an agent service. When the BPWS4J engine has deployed the workflow, it can be tested by invoking the workflow's Web service and verifying the response. The webMethods Glue toolkit provides a convenient command-line Web service invocation utility that eliminates the need to code a Web service client. Figure 5 shows how this utility can be used to invoke the workflow executing in the BPWS4J engine.



```
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>invoke "http://localhost:8080/bpws4j/showWSDL?idNS=urn:simple:stockLookupProcess&idName=stockLookupProcess" requestLookup "CSC" "United States"
result = 42.99
```

Fig. 5. Calling the workflow with the invoke command-line utility from webMethods Glue.

Once the workflow has been verified, select a Web service from the workflow to replace with an agent service. In the demonstration system, the delayed stock quote Web service was selected for this purpose. At this point, a gateway agent for the WSAG needs to be constructed with the same interface as the replaced Web service. The WSAG automates much of this task after a Java interface has been written by the developer. The following code sample defines a Java interface for a target agent service that mimics the stock quote service that is being replaced.

```
package stockQuoteAgent;

public interface StockQuote {
    Float getQuote( String symbol );
}
```

As indicated in Section 3.2, the WSAG gateway agent development tools use this interface definition to construct both the gateway agent and its Web service interface. Although the defined StockQuote interface is consistent with the service being replaced, the WSAG tools are unable to generate a WSDL file that is compatible with

the BPEL4WS workflow definition. When incompatible WSDL definitions are found by the BPWS4J engine, the deployment of the workflow fails. To illustrate this point, a snippet of the tool generated WSDL file follows:

```
<wsdl:message name="getQuoteResponse">
  <wsdl:part name="getQuoteReturn" type="xsd:float"/>
</wsdl:message>
<wsdl:message name="getQuoteRequest">
  <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="StockQuote">
  <wsdl:operation name="getQuote"
    parameterOrder="in0">
    <wsdl:input message="impl:getQuoteRequest"
      name="getQuoteRequest"/>
    <wsdl:output message="impl:getQuoteResponse"
      name="getQuoteResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

This portion of the WSDL file defines the abstract interface to the Web service. It states that the Web service provides an operation named `getQuote` that receives a `getQuoteRequest` message that contains one variable named `in0` whose type is `xsd:string`. The output of the `getQuote` operation is a `getQuoteResponse` message that contains one variable named `getQuoteReturn` whose type is `xsd:float`. Although semantically the same as the `getQuote` service being replaced, the names of the messages and their parameters do not align with the syntax of the replaced service.

Although the BPEL4WS description is decoupled from the services, the underlying services associated with the workflow must have the exact interface definition as the abstract Web services the BPEL4WS was written against. To resolve the deployment issue the incompatibility causes, the WSDL file generated by the WSAG had to be manually edited to reconcile the differences. For comparative purposes, the pertinent section of the new WSDL description follows:

```
<wsdl:message name="getQuoteResponse1">
  <wsdl:part name="Result" type="xsd:float"/>
</wsdl:message>
<wsdl:message name="getQuoteRequest1">
  <wsdl:part name="symbol" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="StockQuote">
  <wsdl:operation name="getQuote"
    parameterOrder="symbol">
    <wsdl:input message="impl:getQuoteRequest1"
      name="getQuoteRequest1"/>
    <wsdl:output message="impl:getQuoteResponse1"
      name="getQuoteResponse1"/>
  </wsdl:operation>
</wsdl:portType>
```

Once the WSDL for the gateway agent is conformant with the WSDL of the service being replaced, the gateway agent can be installed in the WSAG. During the

installation process, the address for the target agent is supplied. Figure 6, shows a partial screen capture of a WSAG user interface. An examination of Figure 6 shows how the gateway agent is configured to communicate with the target agent. The Remote Agent Management utility, shown in Figure 4, can be used to obtain the address of the target agent. Whenever the Web service interface to the gateway agent is invoked, the SOAP message is effectively translated to an ACL message and sent to the target agent.

List of Deployed Web Services

• **stockQuoteAgent** | [WSDL interface](#) | [UNDEPLOY](#) | [REGISTER](#) into UDDI

Type	stockQuoteAgent.StockQuote
Target Agent	target@PAB:1099/JADE
Target Platform	http://PAB:7778/acc

Fig. 6. The user interface to the Deployed Web services function of the WSAG. The figure shows the association between the gateway agent and the target agent.

After the gateway agent's Web service interface has been deployed, the workflow needs to be updated to invoke it. This is accomplished by redeploying the workflow through an administrative interface of the BPWS4J Engine. During the deployment process, the path to the WSDL files for each service partner is supplied. Figure 7, shows a screen capture of the BPWS4J user interface that is used in this step. Instead of supplying the WSDL to the original stock quote service, the WSDL file for the gateway agent's Web service interface is entered. After this change is made, whenever the workflow needs to call the stock quote service, control will be passed to the gateway agent which in turn communicates with the target agent.

Deploy a Process: Partner Identification

Please enter the name of the WSDL file which corresponds to each of the following partners in the business process.

stockQuoteProvider

simpleFloatMultiProvider

currencyExchangeProvider

Fig. 7. One of the user interfaces presented during the deployment of a workflow with the BPWS4J Engine. Note that the path to the WSDL file for the stockQuoteProvider points to the stockQuoteAgent running in the WSAG.

Currently the demonstration systems is only designed to illustrated the incorporation of an agent service into a BPEL4WS workflow. The target agent possesses no intelligence and simply calls the underlying delayed stock quote service that was previously invoked by BPWS4J. The target agent is a JADE agent and is therefore written in Java. The following Java snippet is self documenting and shows the relevant processing:

```

private void stockQuoteRequestBehavior( ACLMessage requestMsg )
{
    final String wsdlName = "urn:xmethods-delayed-quotes.wsdl";
    final String operation = "getQuote";
    String args[] = new String[1];

    // copy the stock symbol from the ACL msg to args[0]
    args[0] = requestMsg.getContent();

    // invoke the stock quote Web service
    String wsResponse = invokeWebService( wsdlName,
        operation, args );

    // create an ACL message to return the response
    ACLMessage responseMsg = new ACLMessage( ACLMessage.INFORM );

    // set this agent's ID in the sender portion of the msg
    responseMsg.setSender( getAID() );

    // set the receiver to be the address of the sender
    responseMsg.addReceiver( requestMsg.getSender() );

    // place the returned stock quote into the content area
    responseMsg.setContent( wsResponse );

    // send the message back to the requestor
    send( responseMsg );
}

```

The message sent from the target agent, returns to the gateway agent. The gateway agent extracts the stock quote from the ACL content area, and returns it back to the BPWS4J Engine. From the Engine's perspective, the changeover from a Web service to an agent service is transparent.

5 Conclusion

Many lessons were learned during the construction of the demonstration system and much work lay ahead. Specifically, a next generation WSAG is being planned. Greater adaptability can be achieved in the next generation WSAG if the coupling between the gateway and target agents were loosened. One approach to looser coupling would require target agents to register with the agent platform's directory facilitator their ability to handle certain Web service requests. When the gateway agent receives a request it could use the directory facilitator to locate target agents capable of providing the desired service. Additionally, it seems logical to design a content language for Web service/agent interaction. Simply placing the invocation parameters into the ACL content area is insufficient for operation in open agent environments. Perhaps something as straightforward as using SOAP as a content language would fulfill this requirement. An added benefit to using SOAP directly is that namespace information would be preserved. As Web services transition from rpc/literal to

doc/literal invocation styles, the namespaces will likely be useful in associating semantic meaning to the message content.

Recently, one other application of the developed platform has been recognized. The Semantic Discovery Service (SDS) described in [18] would benefit from deployment as an agent service rather than as a Web service. As an agent service, the SDS could register itself with the agent platform's directory facilitator enabling its use by other agents. Use of the WSAG also allows the SDS to be the target agent for many gateway agents, each with its own WSDL definition. The gateway agents would be responsible for providing an OWL-S description of the desired service. This semantic description could be passed in the content area of the ACL message along with the invocation parameters. This arrangement allows the use of the SDS to be decoupled from the BPEL4WS process definition, which would no longer have to be modified to support the SDS. Integrating the SDS in this way allows it to be truly agnostic for it would no longer require that it be packaged in a Web service wrapper for each use.

The work presented in this paper is but a first step toward fully integrated agent-based workflow management systems. Although this step may appear small, it represents a great stride forward since it establishes a research platform upon which further Web service/agent integration activities can be performed. The development of the demonstration system illustrates the power of compositional approaches to system creation. It also serves to reinforce the importance of open standards, since the integration of the separate pieces is dependent upon the interoperability that standards provide.

6 Acknowledgements

This work is supported by the U.S. National Science Foundation under grant IIS 0092593 (CAREER award).

References

- [1] Anthes, G. Agents of Change. *Computerworld*:26-27, January 27, 2003.
- [2] WebServices.Org. The 'big boys' unite forces - What does it mean for you?, <http://www.webservices.org/index.php/article/articleview/633/1/24/>.
- [3] XML Cover Pages. Business Process Execution Language for Web services (BPEL4WS), <http://xml.coverpages.org/bpel4ws.html>.
- [4] Buhler, P. and Vidal, J.M. Semantic Web services as Agent Behaviors. in Burg, B., Dale, J., Finin, T., Nakashima, H., Padgham, L., Sierra, C. and Willmott, S. eds. *Agent-cities: Challenges in Open Agent Environments*, Springer-Verlag, Berlin, 2003, 25-31.
- [5] Buhler, P. and Vidal, J.M. Adaptive workflow = Web services + agents. In *Proceedings of the First International Conference on Web services*, 2003.
- [6] Buhler, P. and Vidal, J.M. Towards adaptive workflow enactment using multiagent systems. *Information Technology and Management Journal: Special Issue on Universal Enterprise Integration*, to appear, 2004.

- [7] Vidal, J.M., Buhler, P. and Stahl, C. Multiagent Systems with Workflows. *Internet Computing*, 8(1):76-82, 2004.
- [8] Singh, M.P. and Huhns, M.N. Multiagent Systems for Workflow. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 8:105-117, 1999.
- [9] Huhns, M.N. Agents as Web services. *Internet Computing*, 6(4):93-95, 2002.
- [10] Buhr, R.J.A. and Casselman, R.S. *Use case maps for object-oriented systems*. Prentice Hall, 1996.
- [11] IBM. BPWS4J, <http://www.alphaworks.ibm.com/tech/bpws4j>.
- [12] Whitestein Information Technology Group AG. Web services Agent Integration Project, <http://wsai.sourceforge.net/index.html>.
- [13] Telecom Italia Lab. JADE (Java Agent DEvelopment Framework), <http://sharon.cselt.it/projects/jade/>.
- [14] Hohpe, G. and Woolf, B. *Enterprise integration patterns : designing, building, and deploying messaging solutions*. Addison-Wesley, Boston, 2003.
- [15] Agentcities Web services Working Group. Integrating Web services into Agentcities Technical Recommendation, <http://www.agentcities.org/rec/00006/>.
- [16] The Foundation for Intelligent Physical Agents, www.fipa.org.
- [17] Brownsword, L., Oberndorf, T. and Sledge, C.A. Developing New Processes for COTS-Based Systems. *IEEE Software*, 17(4):48-55, 2000.
- [18] Mandell, D.J. and McIlraith, S.A. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web service Interoperation. In *Proceedings of the Second International Semantic Web Conference*, 227-241, 2003.