Enacting BPEL4WS Specified Workflows with Multiagent Systems

Paul A. Buhler
Computer Science Dept
College of Charleston
Charleston, SC 29424 USA
pbuhler@cs.cofc.edu

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

ABSTRACT

This paper describes our development of a distributed, functionally equivalent agent-based workflow enactment mechanism from a BPEL4WS specification. This work demonstrates that BPEL4WS can be viewed as a description of the social order of a collection of agents, where the agents serve as proactive proxies for the underlying passive Web services. Although the Semantic Web initiative is working toward semantically rich descriptions of Web services, which can be reasoned about by agents, the current state-of-the-art does not yet allow for collections of agents representing semantic Web services to organize themselves to enact workflows. Therefore, this work is critically important as it serves as a bridge from existing, static views of workflow enactment to future, agent-based, dynamic workflow engines.

1. INTRODUCTION

This paper details the design and development of an open, distributed, agent-based workflow enactment mechanism utilizing BPEL4WS [2] as the specification of the Multiagent System (MAS). The impact of this work is broad, as it cuts a swath across many existing and emerging technologies; for example, Business Process Management Systems, Web services, Internet Agents, application integration, and XML-based coordination mediums.

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. [18]. 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 [18].

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. In our earlier works [13],[12],[24],[11] 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, for example [23]; 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 deliberatively cooperative behavior [17]. In contrast, agents possess all of these capabilities.

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.

This paper will first detail a sample BPEL4WS workflow that will serve as a running example throughout the remainder of the paper. Next, a discussion of the architecture and design of the distributed enactment mechanism is presented. This is followed by an examination of the hybrid coordination model used. The discussion proceeds with detail about the design of the workflow agents. The paper provides information on how the enactment mechanism is configured, including an examination of the configuration data that is consumed by the workflow agents. Finally, the paper concludes with a discussion of lessons learned, insights gained, and future work.

2. A SAMPLE BPEL4WS WORKFLOW

BPEL4WS is an XML-based defacto standard that allows the specification of a workflow where the activities are defined by Web service invocations. BPEL4WS has been submitted to OASIS for standardization and in the future will be known as WS-BPEL. A complete description of BPEL4WS is beyond the scope of this paper; however, the following discussion should provide enough background to enable understanding of the sample workflow.

BPEL4WS files specify the coordination of control and data between service partners that represent underlying Web services. Control constructs such as sequence and split-join are represented by XML tags that delineate control blocks. For example, the actions found between a <flow>, </flow> tags are to be executed concurrently. BPEL4WS defers to the underlying WSDL for the specification of the data that is exchanged by the service partners. The messages exchanged with a Web service are designated by variables within the BPEL4WS file. Assignment and copy operations between variables allows data to be manipulated and passed between Web services.

Often initial research efforts are directed toward solving "toy" problems. The example workflow described below serves this purpose. Abstractly, the workflow consumes two parameters, a stock symbol and a country name. The result of the workflow is a quote for the stock localized into the currency of the given country. For example, providing 'CSC' and 'Switzerland' will return the price for a single share of Computer Sciences Corporation stock in Swiss Francs.

The example workflow encoded in BPEL4WS follows. A few items to note, bold-face text is used to designate the control constructs and workflow activities, the remaining text describes the data-centric coordination of messages exchanged between the partners and their Web services. The BPEL4WS has been simplified by removing attributes that do not help clarify the example.

```
<process>
  <partners>
    <partner name="requestor"/>
    <partner name="stockQuoteProvider"/>
    <partner name="currencyExchangeProvider"/>
    <partner name="simpleFloatMultProvider"/>
  </partners>
  <variables>
    <variable name="request"/>
    <variable name="response"/>
    <variable name="stockQuoteProviderRequest"/>
    <variable name="stockQuoteProviderResponse"/>
    <variable name="currencyExchangeProviderRequest"/>
    <variable name="currencyExchangeProvidrResponse"/>
    <variable name="simpleFloatMultProviderRequest"/>
    <variable name="simpleFloatMultProviderResponse"/>
  </variables>
  <sequence>
    <receive name="request"
       partner="requestor"
       operation="requestLookup"
       variable="request"
       createInstance="yes">
    </receive>
    <assign>
      <copy>
        <from variable="request" part="symbol"/>
        <to variable="stockQuoteProviderRequest"
           part="symbol"/>
      </copy>
      <copy>
        <from expression="'usa'"/>
        <to variable="currencyExchangeProviderRequest"
           part="country1"/>
      </copy>
      <copy>
        <from variable="request" part="country"/>
        <to variable="currencyExchangeProviderRequest"
           part="country2"/>
      </copy>
    </assign>
    <flow>
      <invoke name="getStockQuote"
         partner="stockQuoteProvider"
         operation="getQuote"
         inputVariable=
            "stockQuoteProviderRequest"
        outputVariable=
            "stockQuoteProviderResponse">
      </invoke>
      <invoke name="getExchangeRate"
         partner="currencyExchangeProvider"
         operation="getRate"
         inputVariable=
            "currencyExchangeProviderRequest"
         outputVariable=
            "currencyExchangeProviderResponse">
      </invoke>
    </flow>
    <assign>
      <copy>
        <from variable="stockQuoteProviderResponse"
           part="Result"/>
        <to variable="simpleFloatMultProviderRequest"
           part="f1"/>
      </copy>
      <copy>
        <from variable=
           "currencyExchangeProviderResponse"
           part="Result"/>
        <to variable="simpleFloatMultProviderRequest"
           part="f2"/>
      </copy>
    </assign>
    <invoke name="multiplyFloat"
       partner="simpleFloatMultProvider"
       operation="multiply"
       inputVariable=
          "simpleFloatMultProviderRequest"
       outputVariable=
          "simpleFloatMultProviderResponse">
    </invoke>
    <assign>
      <copy>
        <from variable="simpleFloatMultProviderResponse"
           part="multiplyReturn"/>
        <to variable="response" part="Result"/>
```

```
      </copy>
    </assign>
    <reply name="response"
        partner="requestor"
        operation="requestLookup"
        variable="response">
    </reply>
  </sequence>
</process>
```

Internally, the workflow definition coordinates the interaction of the four workflow partners named: requestor, stockQuoteProvider, currencyExchangeProvider, and simpleFloatMultProvider. Figure 1, provides a graphical view of the structure of the workflow in Use Case Maps (UCM) notation [14].

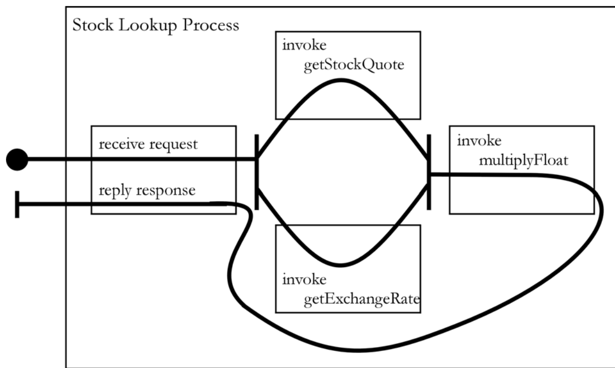


Figure 1. A UCM diagram for the example workflow

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. 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.

3. ARCHITECTURE AND DESIGN

Web services and the BPEL4WS have created a resurgence of interest in workflow technologies and process-oriented views of software systems. Traditionally, workflow engines have been based upon the static enactment of workflows under centralized control. This classic approach is at odds with current trends towards real-time enterprises, which closely monitor changing marketplace conditions and events. The ultimate goal is to have this data fed back into the business processes, increasing process responsiveness by allowing adaptive changes to occur. To achieve this type of workflow agility, new enactment mechanisms are required.

Distributed systems possess three dimensions of distribution: computation, control, and data. With BPEL4WS, the Web services are the computational activities, and the control and data dimensions specify the coordination required to manage the process. The BPWS4J Engine is a BPEL4WS enactment engine available from IBM's AlphaWorks site [1]. BPWS4J provides central coordination of the workflow, while the computation is potentially distributed across the Internet. In BPWS4J, each workflow instance has its own thread of control with simulated parallelism, thus the engine enacts the workflow as a distributed application [15]. Distributed applications typically posses a single thread of control and use synchronous communications to transfer control from one component to the next.

Our perspective is that the application integration paradigm provides a more appropriate model of Internet based workflow enactment, particularly when inter-organizational workflows are considered. Application integration considers the components to be independently executing applications that are integrated via the asynchronous exchange of data and control. Since Web services are passive entities that don't execute until called, we wrap them in proactive agents that possess their own thread of control. The agents are then integrated to enact the workflow. The agents are coordinated with a shared data space and the asynchronous exchange of messages. This architecture is flexible and loosely coupled.

Our goal is to create an open architecture, built atop open standards, for increased interoperability. The primary Web service standards of SOAP, WSDL and UDDI allow for language and platform neutral Web service invocation. In the agent space, the FIPA standards [3] 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. Additionally, we chose to use open source or freely available software whenever possible.

Another design goal worth mentioning was the desire to preserve the compositional completeness property inherent to BPEL4WS. In this context, compositional completeness means that the composition of Web services is itself published and accessed as a Web service that can participate in other compositions [22]. Since complex workflows are often viewed as a hierarchy of workflows, the compositional completeness property allows agent-based workflows to be incorporated via BPEL4WS into other workflow definitions.

Based upon our architectural desires and design constraints, the following software components were used in the creation of the distributed enactment mechanism: BPWS4J Editor for the graphical creation of BPEL4WS specified workflows; webMethod's Glue [4] as a high level Web service invocation toolkit; JADE [5] as a FIPA compliant agent development environment; the Web Service Agent Gateway (WSAG) [6] as a bridge between synchronous Web service calls and asynchronous agent messaging; and Xindice [7] a networked, native XML database used as a coordination medium.

4. HYBRID COORDINATION MODEL

As previously discussed, the domain of coordination encompasses issues of both data and control. The distributed workflow enactment mechanism utilizes a hybrid coordination model, which means that it combines separate data-centered and control-centered coordination mechanisms [16]. The data is managed via a shared, network addressable XML repository, while the control of the workflow activities is driven by asynchronous message exchange between the agents. The message exchange pattern for the control messages is derived from a Colored Petri Net (CPN) model of the workflow.

4.1 Xindice as a Coordination Medium

Xindice facilitates the storage, retrieval, and sharing of XML data. Xindice is a networked native XML database that complies with the XML:DB API specification. Xindice stores XML documents in logical groupings called collections. Data is retrieved from a collection via the evaluation of an XPath [8] query that is

evaluated against the documents in a collection. Xindice's features make it an ideal choice as a coordination medium.

Tuple spaces are often the coordination medium of choice for agent-based systems. Tuple spaces allow processes to communicate across space and time, e.g. a process running on one machine can write information to a shared tuple space which is to be read by another process, running on a different machine the day after tomorrow. Tuple spaces provide a form of associative memory. Associative memory is accessed by content, not by address. By way of analogy, SQL is used to retrieve records from a RDBMS that match criteria specified in the 'where' clause of the query. In the same way, a query against a tuple space retrieves records that match criteria specified in a template. With Xindice, XPath can be viewed as a template mechanism that can retrieve specific elements, attributes, or even collections of nodes from an XML document.

An example will provide some insight into how Xindice and XPath are used as a coordination medium for the sharing of data across the distributed workflow agents. In our workflow example, the stockQuoteProvider partner interacts with a stock quote Web service. This interaction occurs with XML-based SOAP messages, which are intercepted and stored in Xindice. A sample of a captured SOAP Response message appears below.

```
<soap:Envelope>
   xmlns:soap="http://schemas.xmlsoap…
   xmlns:xsi="http://www.w3.org/2001/…
   xmlns:xsd="http://www.w3.org/2001/…
   xmlns:soapenc="http://schemas.xmls…
   soap:encodingStyle="http://schema… >
   <soap:Body>
     <n:getQuoteResponse
      xmlns:n="urn:xmethods-delayed-quotes">
       <Result xsi:type="xsd:float">
          40.35
       </Result>
     </n:getQuoteResponse>
   </soap:Body>
</soap:Envelope>
```

Downstream in the workflow, the returned stock quote needs to be multiplied against the currency exchange rate to localize the price. For this to occur, the quoted price needs to be extracted from the XML document presented above. The following XPath query retrieves the quote as a string, which can then be converted into its numeric equivalent.

```
string(//n:getQuoteResponse/Result)
```

Requests for the execution of the workflow generate unique collections within the Xindice repository. This allows for the clean separation of data between individual workflow cases. Additionally, it assures efficient XPath queries since the number of documents in a given collection remains small.

4.2 CPNs as a Flow Control Mechanism

Petri Nets (PNs) have been used for workflow control since the mid 1990's [9]. PNs, also known as place-transition nets, provide a deceptively simple, yet rigorous, way to model finite state machines. PNs are represented as directed graphs with two types of nodes, places and transitions, which are graphically represented as circles and squares respectively. The state of execution is maintained by tokens that reside in the place nodes of a PN. A transition is enabled if each of its input places is marked by a token. When a transition is enabled it fires, removing a token from each of the input places and depositing a token in each of the output places. From a workflow perspective, the activities of the process occur at the transition nodes in the net. Figure 2 presents the example workflow in PN form, where the transitions correspond with the following activites: A – receive request, B – invoke getStockQuote, C – invoke getExchangeRate, D – invoke multiplyFloat and E – reply response.

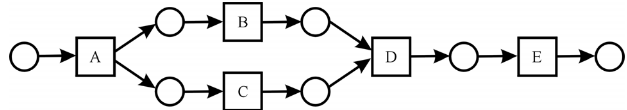


Figure 2. A PN Model for the example workflow

A comparison of the UCM diagram in Figure 1 with the PN model in Figure 2 reveals that they are equivalent.

CPNs are an extension of basic PNs and include the notion that the tokens carry data. The different colored tokens equate to different data types. The demonstration system utilizes two different colored tokens. The first is used for messaging between the WSAG and the agent-based enactment mechanism. The second is used to communicate control information between the agents as they process a workflow instance. The following is a sample message sent by the WSAG:

```
WSAG:stockLookupProcess:requestor|request:csc:
   Switzerland
```

The message has a signature indicating that it is being sent by the WSAG. Next, the message identifies the name of the workflow, followed by the partner name the message is intended for. The vertical bar separates the message header from the payload. The payload of the message indicates that a request is being made for a quote for CSC stock localized into Swiss currency.

An example of a control message exchanged between two agents during workflow enactment follows:

```
DWfA:stockLookupProcess:simpleFloatMultProvider
   :1080665330511:currencyExchangeProvider
```

This message carries the Distributed Workflow Agent (DWfA) signature, identifies the workflow name, and the partner name the message is intended for. The numeric value is a unique ID that is assigned to each workflow instance. This ID is also used to identify the appropriate collection in the Xindice database. The final piece of information is the name of the partner role that sent the message; in this example the message is from the currencyExchangeProvider. Given the PN shown in Figure 2, it should be apparent that before the simpleFloatMultProvider can invoke the multiplication Web service, it would need to receive messages from both the currencyExchangeProvider and the stockQuoteProvider for the same workflow instance.

It is not hard to imagine using a PN within a centralized workflow enactment mechanism to control the execution order of the workflow activities. However, an interesting question arises regarding the use of a PN for distributed workflow enactment. This question is how is it possible to separate the net into pieces

that can be distributed while retaining equivalent behavior. The answer is illustrated in Figure 3, which depicts the refinement of a place between two transitions with a simple PN consisting of two places and one transition. After the refinement of the net, Transition T1 now writes a token to place P2.1, which enables the subsequent transition. This transition writes its output token to place P2.2, which may reside across a network. Place P2.2 in turn enables transition T2.

More concretely, the transitions in the PN model are agents and the transition labeled DF/MTS represents FIPA compliant Directory Facilitator (DF) and Message Transport Service (MTS) components. When an agent in the workflow completes its task, it utilizes the DF to locate the address of the agent that has registered itself as playing the next partner role that needs to receive control. The agent generates an ACL Request message, loads the content area with DWfA signed data, and sends the message to the address returned by the DF. The MTS in turn facilitates the message delivery. Thus the distribution of the CPN is effectively managed by the DF acting as a middle-agent [19]. Figure 4 presents a UML sequence diagram, which illustrates the message exchange pattern for an instance of the example workflow.

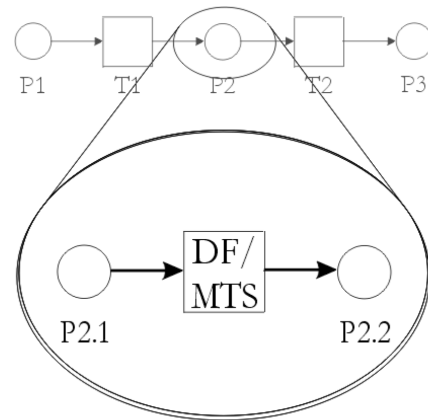
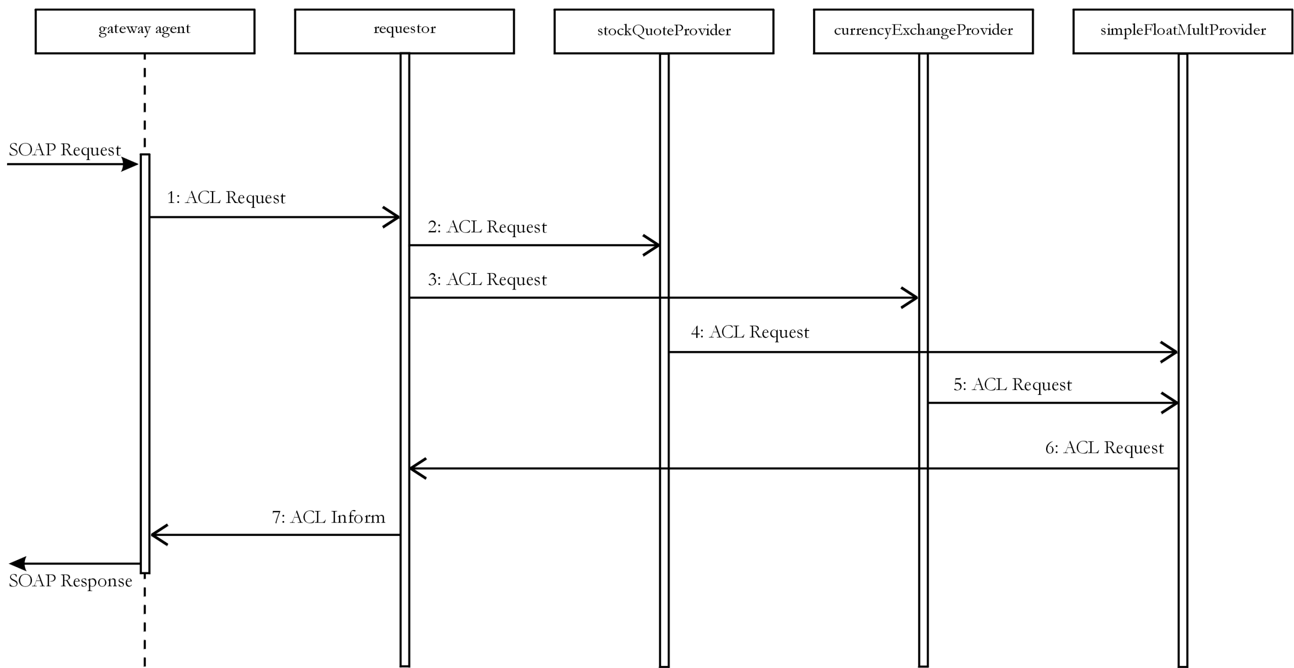


Figure 3. Refinement of P2 with a subnet




```
1: WSAG:stockLookupProcess:requestor|request:csc:usa
2: DWfA:stockLookupProcess:stockQuoteProvider:1081186215373:requestor
3: DWfA:stockLookupProcess:currencyExchangeProvider:1081186215373:requestor
4: DWfA:stockLookupProcess:simpleFloatMultProvider:1081186215373:stockQuoteProvider
5: DWfA:stockLookupProcess:simpleFloatMultProvider:1081186215373:currencyExchangeProvider
6: DWfA:stockLookupProcess:requestor:1081186215373:simpleFloatMultProvider
7: 42.3
```

Figure 4. A UML sequence diagram showing the message exchange pattern derived from the CPN model, and sample data.

5. AGENT DESIGN

There are two types of agents that enact the workflow: target agents and distributed workflow agents. A target agent interfaces the distributed workflow agents to the WSAG. The distributed workflow agents are the proactive proxies for the passive Web services they represent. Both types of agents are implemented with JADE and are thus FIPA compliant.

One of the design goals for the distributed workflow enactment mechanism was to have the ability to externally configure the agents at run time. Thus the agents are generic and are differentiated through an instantiation of their behavioral characteristics, which are defined by the partner roles in the BPEL4WS file. Section 6.2 provides a detailed discussion of the external configuration data and its use.

5.1 Target Agents

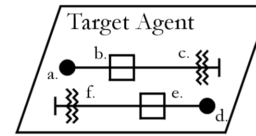
Figure 5 illustrates the structure of a target agent in UCM notation. The agent is represented with a parallelogram, which indicates it is an active component in the system. Target agents receive messages from both the WSAG and other distributed workflow agents; the two distinct execution paths in Figure 5 denote this. The boxes found on the execution path simply designate that some processing is occurring, while the two squiggly lines note a "layer fold" in UCM notation. A layer fold is an abstraction that indicates that some complexity is hidden or collapsed along the path. In this case, the layer fold is used to indicate the interaction of the target agent with the middle-agents.

5.2 Distributed Workflow Agents

Figure 6 reflects the implementation of the distributed workflow agents. The only new UCM notation is the dashed rounded rectangle, which is a placeholder symbol for a passive component. The distributed workflow agents share the same code base; they are simply instantiated with different workflow partner information. This is consistent with the fact that the primary distinction between these agents is the Web service they represent.

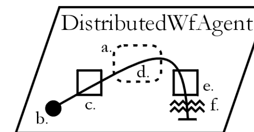
6. SYSTEM CONFIGURATION

The architecture for the distributed enactment mechanism relies upon many different components that must be properly configured. Figure 7, provides a high-level diagram that shows the interaction between the major components. Note that the solid lines tipped with arrows indicate synchronous message exchange, while the dashed variation designates asynchronous messaging. The following sections will describe the configuration of the components shown in Figure 7.



a. workflow enactment request from the WSAG
b. create collection for the case and store the request message in Xindice
c. utilize DF to locate outgoing partner roles and MTS to deliver ACL Request message(s)
d. receipt of DWfA message indicating case completion
e. retrieve workflow response from Xindice
f. utilize MTS to send response to the gateway agent

Figure 5. UCM diagram of a Target Agent



a. slot to hold run-time assignment of Web service
b. request for Web service invocation
c. case managment and Xindice access for building Web service request message
d. dynamic bind and invocation of the Web service
e. Xindice storage of SOAP request/response pair
f.. utilize DF to locate outgoing partner roles and MTS to deliver ACL Request message(s)

Figure 6. UCM diagram of a Distributed Workflow Agent

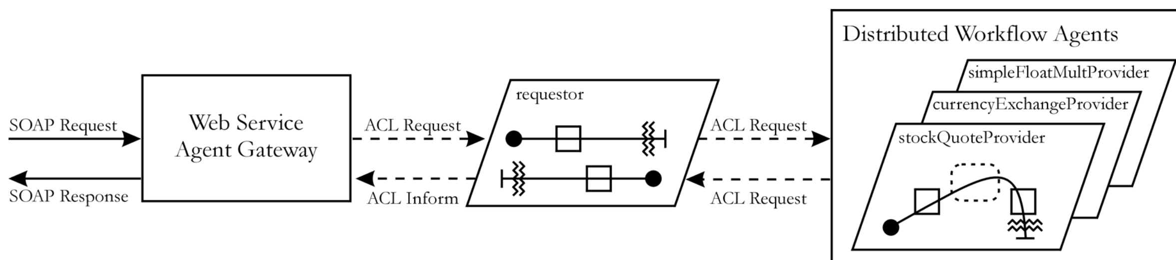


Figure 7. Illustration of the interaction between the components of the distributed enactment mechanism

6.1 Configuring the WSAG

The WSAG provides a Web service interface for services provided by a target agent. In our example, the target agent plays the requestor partner role. As defined in the BPEL4WS, the requestor receives requests from end users and responds with a reply after the workflow runs.

Use of the WSAG requires that a gateway agent is generated and deployed. It is critical that the interface for the gateway agent aligns with the workflow's SOAP request and response message structure. The gateway agent's interface is specified with a Java interface. The WSAG provides tools that facilitate the generation of gateway agents. These tools consume the Java interface and produce a skeletal gateway agent. The skeletal code is then edited to comply with the messaging interface of the target agent. The gateway agent is then compiled and packaged for deployment. For the example workflow, the following Java interface was used to generate the gateway agent.

```
package stockLookupProcess;

public interface StockLookupProcess
{
  Float request( String symbol,
          String country );
}
```

Once the gateway agent is built and installed, it needs to be deployed. The deployment step publishes a WSDL interface for the gateway agent, and associates the gateway agent with the target agent. The WSAG management console provides the means to accomplish this task. Figure 8 shows the configuration of the stockLookupProcess gateway agent. When the WSAG receives a SOAP request for the stockLookupProcess, the gateway sends an ACL request to the specified target agent running on the target agent platform. When the workflow is complete the target agent sends an ACL Inform back to the gateway agent, which in turn sends a SOAP response to the workflow consumer.

Type	stockLookupProcess.StockLookupProcess
Target Agent	requestor@PAB:1099/JADE
Target Platform	http://PAB:7778/acc

Figure 8. Configuration of the Gateway Agent

6.2 Configuring the Workflow Agents

The workflow agents in the system share a single configuration file, expressed in XML, that is conveniently stored in Xindice. The configuration data is derived from the BPEL4WS file and the underlying WSDL files for the individual Web services. Currently, the configuration data is manually generated; however, we believe that much of this process can be automated.

A portion of the configuration data for the example workflow process is provided and discussed below.

```
<configData workflow="stockLookupProcess">

<messages>
 <message name="request">
   <part name="symbol" type="xsd:string"/>
   <part name="country" type="xsd:string"/>
 </message>
 <message name="response">
   <part name="Result" type="xsd:float">
    q:string(//agent[@role='simpleFloatMultProvider']
    /response//ns1:multiplyReturn)
   </part>
 </message>
<message name="simpleFloatMultProviderRequest">
   <part name="f1" type="xsd:float">
    q:string(//agent[@role='currencyExchangeProvider']
     /response//Result)
   </part>
   <part name="f2" type="xsd:float">
    q:string(//agent[@role='stockQuoteProvider']/
     response//Result)
   </part>
 </message>
 <message name="simpleFloatMultProviderResponse">
   <part name="multiplyReturn" type="xsd:float"/>
 </message>
</messages>

<partners>
 <partner name="requestor">
   <inputPlaces/>
   <service>
     <wsdl> </wsdl>
     <operation> </operation>
     <messageName>response</messageName>
   </service>
   <outputPlaces>
     <place>stockQuoteProvider</place>
     <place>currencyExchangeProvider</place>
   </outputPlaces>
 </partner>
<partner name="simpleFloatMultProvider">
   <inputPlaces>
     <place>stockQuoteProvider</place>
     <place>currencyExchangeProvider</place>
   </inputPlaces>
   <service>
      <wsdl>
        http://…/axis/SimpleFloatMult.jws?wsdl
      </wsdl>
      <operation>multiply</operation>
      <messageName>
         simpleFloatMultProviderRequest
      </messageName>
   </service>
   <outputPlaces>
      <place>requestor</place>
   </outputPlaces>
 </partner>
</partners>

</configData>
```

The configuration file contains both data-centric and control-centric coordination information relevant to the enactment of the workflow. The data-centric portion is identified with the <messages> tag, while the control-centric section is identified with the <partners> tag.

The <messages> section defines the messages that the individual partners use when interacting with their associated Web service. The message names come directly from the BPEL4WS file, while the message parts are specified in the underlying WSDL files for each Web service. Each message

part has an optional value that is either a constant, designated by "c:", or an XPath query designated by a "q:". The associated XPath queries inform the agent how to obtain the data from Xindice. For example, the target agent sends an ACL Inform message to the gateway agent, the contents of this response message is defined in the configuration file. The response message contains one part named Result, whose type is xsd:float. The associated XPath query specifies how to obtain the data from the SOAP response message stored into Xindice by the simpleFloatMultProvider.

The <partners> section contains the control-centric coordination information relevant to each of the partners in the workflow. The partner names are the same as those specified in the BPEL4WS file. Each partner is bound to a specific Web service, specified by a wsdl, operation, messageName triplet. The messageName corresponds with a message found in the <messages> section of the configuration file.

The agents track each DWfA signed message they receive against the individual workflow cases. When an agent receives a message for a workflow instance from each of the partners specified in the <inputPlaces> section, the agent invokes the Web service. This directly corresponds to the enabling of a transition in a PN, since each of its input places are marked. Next, the intercepted SOAP request/response pair from the Web service interaction is stored in Xindice. The agent then sends a DWfA message to each of the workflow partners found in the <outputPlaces> section. For example, the simpleFloatMultProvider will not call the multiplication Web service until it has received messages from both the stockQuoteProvider and the currencyExchangeProvider. Once these messages are received, the multiplicationWeb service is invoked, the SOAP messages are stored, and the requestor role is notified.

6.2.1 Command Line Parameters

The workflow agents are provided the name of the workflow in which they are participating and the name of the partner role they are performing via command line parameters. As previously mentioned, the distributed workflow agents are each instances of the same Java class. It is the command line parameters that distinguish them. The parameters provide the agent enough information to retrieve partner specific information from the workflow's global configuration file. The agent uses the partner information when registering with the DF, so that other agents can identify it as playing a specific partner role.

For example, the following shows the command line used to establish the stockQuoteProvider agent:

```
java jade.Boot –container
     stockQuoter:DistributedWfAgent
    (stockLookupProcess stockQuoteProvider)
```

The target agent utilizes a different class file; however, it is established in a similar fashion. The command line to establish the target agent is:

```
java jade.Boot
    -container-name Target-Container
    -gui requestor:TargetAgent
    (stockLookupProcess requestor)
```

Figure 9 shows a screen shot of the JADE Remote Agent Management console with the entire complement of workflow agents running.

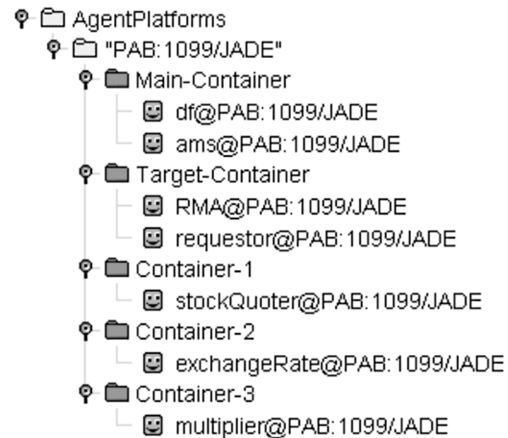


Figure 9. JADE's Remote Agent Management console showing the full complement of workflow agents

7. CONCLUSION AND FUTURE WORK

One of the most important points to make about the distributed workflow enactment mechanism is that it is functional and provides a research platform upon which further refinement and experimentation can be performed. Throughout its development, we have grappled with many issues and found reasonable and scaleable solutions. Although not discussed in this paper, one of the challenges that must be overcome when dynamically binding to Web services in a stubless manner is how to deal with the returned data. It was our conscious decision to forgo the unmarshalling of the SOAP response. Keeping the data in XML format insulates the code from differences between rpc/literal and doc/literal Web service styles. Additionally, it becomes transparent that a native XML database is the coordination medium of choice.

We have learned many valuable lessons and there remains much work to do. Importantly, our demonstration system does not support <switch> and <pick> BPEL4WS constructs. These constructs support selective routing, which can be thought of as the business rules of the workflow process. For example, if the response from the previous Web service was less than 50, pass control to partner A, otherwise use partner B. We believe that we can still maintain code genericity by augmenting the <outputPlaces> section of the configuration file with RuleML. The rules will then be processed as conditional logic scripts in a manner inspired by [21].

The hybrid coordination model has proven its relevance with the demonstration system. If for example a Linda-like tuple space were used to convey control messages, the first agent to consume the message does the work. Our use of the DF and

asynchronous messaging opens up interesting research opportunities regarding task allocation. For example, consider what might happen when a workflow agent utilizes the DF to locate an agent playing the role identified by an outgoing place and it is discovered that multiple agents are returned. The agent might use a reputation mechanism to select one of the partners, or engage in a bidding scenario managed with a contract net protocol, et al. The point is that the individual agents maintain the opportunity to do something intelligent and potentially optimize the execution of the workflow at run-time.

The conversion of BPEL4WS into PN form is another area that requires further study. Currently, we generate PNs based upon the replacement property that exists with workflow nets [10]; however, while sufficient for modeling positive flow control, it is difficult to capture fault and exception handling. Additionally, the fact that BPEL4WS inherits the calculus-based approach of XLANG presents difficulty when being expressed with PN's graph-based constructs. We will further pursue our work with Humboldt University where ongoing work is developing a PN semantic for BPEL4WS. An initial description of this approach can be found in [24].

Other opportunities exist to demonstrate the advantages of agent-based workflow enactment. As more semantic Web services become available, we would like to integrate the Semantic Discovery Service (SDS) [20] as an basic agent service available to the workflow agents. To accomplish this integration, the <partner> description in the configuration file would need to be augmented with a semantic description of the Web service the partner represents. At run-time, the workflow agent can use its autonomy to locate other potential Web service partners with the aid of the SDS. This integration would allow the agents to heal the workflow in the event that their primary Web service becomes unresponsive. Likewise, various Web services would likely provide different QOS levels, which would provide opportunities to explore self-optimizing algorithms.

Finally, the work described in this paper opens up a new avenue of research regarding Agent-Oriented Software Engineering (AOSE). We have demonstrated that it is possible to take a BPEL4WS file that was created in graphical workflow design tool, and use it to instantiate a MAS. This opens the possibility that a more general MAS design methodology and toolset can be formalized from advancements occurring in the Business Process Management space. This is a natural fit because a workflow essentially represents the sociality of the business process, that is, the relationships between the workflow participants, the necessary conversations they have while processing the work, and the work product itself. It is worth exploring if an end-to-end AOSE process can be formalized consisting of the Gaia Agent-Oriented Analysis and Design methodology [25], graphical workflow design tools which emit BPEL4WS, and the distributed workflow enactment mechanism described in this paper.

8. ACKNOWLEDGEMENTS

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

9. REFERENCES

[1] IBM. BPWS4J, http://www.alphaworks.ibm.com/tech/bpws4j.

[2] XML Cover Pages. Business Process Execution Language for Web Services (BPEL4WS), http://xml.coverpages.org/bpel4ws.html.

[3] The Foundation for Intelligent Physical Agents, www.fipa.org.

[4] webMethods, Inc. Glue Overview, http://www.webmethods.com/solutions/wM_Glue/

[5] Telecom Italia Lab. JADE (Java Agent DEvelopment Framework), http://sharon.cselt.it/projects/jade/.

[6] Whitestein Information Technology Group AG. Web services Agent Integration Project, http://wsai.sourceforge.net/index.html.

[7] The Apache XML Project. Xindice Homepage, http://xml.apache.org/xindice.

[8] World Wide Web Consortium. XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath.

[9] Aalst, W.v.d. The Application of Petri Nets to Workflow Managment. *Journal of Circuits, Systems, and Computers*, *8*(1):21-66, 1998.

[10] Aalst, W.v.d. and Hee, K.M.v. *Workflow management : models, methods, and systems*. MIT Press, Cambridge, Mass., 2002.

[11] Buhler, P. and Vidal, J.M., Integrating Agent Services into BPEL4WS Defined Workflows, USC CSE TR-2004-003, http://jmvidal.cse.sc.edu/papers/buhlertr04a.pdf.

[12] Buhler, P., Vidal, J.M. and Verhagen, H. Adaptive workflow = web services + agents. In *Proceedings of the First International Conference on Web Services*, 131-137, 2003.

[13] Buhler, P.A. and Vidal, J.M. Towards the Synthesis of Web Services and Agent Behaviors. In *Proceedings of the Agentcities: Challenges in Open Agent Environments Workshop*, 25-31, 2002.

[14] Buhr, R.J.A. and Casselman, R.S. *Use case maps for object-oriented systems*. Prentice Hall, 1996.

[15] Curbera, F. and Khalaf, R. Implementing BPEL4WS: The Architecture of a BPEL4WS Implementation. In *Proceedings of the Grid Workflow Workshop at GGF-10*, 2004.

[16] DeLoach, S.A. Analysis and Design of Multiagent Systems Using Hybrid Coordination Media. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, 2002.

[17] Huhns, M.N. Agents as Web Services. *Internet Computing*, *6*(4):93-95, 2002.

[18] WebServices.Org. The 'big boys' unite forces -

What does it mean for you?, http://www.webservices.org/index.php/article/articleview/633/1/24/.

[19] Klusch, M. and Sycara, K. Brokering and Matchmaking for Coordination of Agent Societies: A Survey. in Omicini, A., Zambonelli, F., Klusch, M. and Tolksdorf, R. eds. *Coordination of Internet agents : models, technologies, and applications*, Springer, Berlin ; New York, 2001, 197-224.

[20] 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.

[21] Nadelson, M. Stay Flexible with Logic Scripts. *JavaPro*, *7*(9), 2003.

[22] Schneider, J.-G., Lumpe, M. and Nierstrasz, O. Agent Coordination via Scripting Languages. in Omicini, A., Zambonelli, F., Klusch, M. and Tolksdorf, R. eds. *Coordination of Internet Agents : Models, Technologies, and Applications*, Springer-Verlag, New York, NY, 2001, 153-175.

[23] 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.

[24] Vidal, J.M., Buhler, P. and Stahl, C. Multiagent Systems with Workflows. *Internet Computing*, *8*(1):76-82, 2004.

[25] Wooldridge, M., Jennings, N.R. and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, *3*:285-312, 2000.