Executing Abstract Web Process Flows

Rama Akkiraju[1], Kunal Verma[2], Richard Goodwin[1], Prashant Doshi[3], Juhnyoung Lee[1]

[1]IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532

[2] LSDIS Lab, Department of Computer Science, University of Georgia, Athens, Georgia, GA 30602-7404

[3]Department of Computer Science, University of Illinois, 851 S. Morgan, Chicago, IL 60607

{akkiraju,rgoodwin, jyl}@us.ibm.com

verma@cs.uga.edu

pdoshi@cs.uic.edu

Abstract

Current business process flow representation languages such as BPEL4WS do not accommodate abstract specifications of business activities and dynamic binding of web services at run time. Moreover, dynamic selection of individual web services for a process is often not a stand-alone operation. There may be many inter-service dependencies and domain constraints that need to be considered in selecting legal and appropriate services for realizing an abstract flow. In this paper, we present a prototype workflow engine that accepts abstract BPEL4WS flows augmented with semantic annotations in DAML-S and performs runtime discovery, composition, binding and execution of web services. Building on prior work in this area [Mandel and McIlraith 2003], we provide a way of modeling and accommodating domain constraints and inter-service dependencies within a process flow. The result is a system that allows workflow designers to focus on creating appropriate high-level flows, while providing a robust and adaptable runtime.

1. Introduction

The Business Process Execution Language for Web Services (BPEL4WS) [BPEL 2002] is a language for specifying business processes and business interaction protocols. It superseded XLANG [Thatte 2001] and WSFL [Layman et al. 2001] as standards for Web services flow specification. BPEL4WS provides a representation mechanism for specifying process execution flows, including constructs for representing complex flows, data handling and correlation. In its current form, BPEL4WS requires static binding of services to flows. The process model defined by BPEL4WS is based on the WSDL [Christenson et al. 2001] service description model. WSDL lacks semantic expressivity, which is needed to describe service requirements in an abstract flow. In addition, BPEL4WS does not provide a mechanism for modeling constraints and inter-service dependencies in a process flow.

Some of these limitations are already being addressed by efforts by the Semantic Web community. Efforts in this area originally focused on developing ontology markup languages, such as DAML [DAML 2000], DAML+OIL [DAML+OIL 2001] and OWL [OWL 2002]. More recent efforts have addressed the lack of semantics in the industry backed Web Services standards. The Semantic Web Community has developed a DAML+OIL ontology for Web Services, DAML-S [Ankolekar et al., 2002], and it follow on OWL-S. This family of semantic markup languages and ontologies lay the foundation for Semantic Web Services [McIlraith, Son and Zeng 2001], automatic service discovery, and service composition. However, much work still needs to be done to tie in these foundation technologies with business process integration issues in the context of an industry setting.

In this work, we take a consultant's view of business process flow representation rather than an IT programmer's view. We argue that business process flow specifications should be defined at an abstract task level, leaving the details of specific service bindings for the runtime flow execution engine to manage. To investigate the feasibility of such an approach, we have developed a prototype BPEL4WS runtime that can accept abstract BPEL4WS flows augmented with semantics and perform runtime discovery and binding of Web Services. In this paper, we discuss our implementation and experience with dynamic binding of Web Services. In particular, we focus on our contention that the selection of a web service for a step in a process flow is often not a stand-alone operation. There may be inter-service dependencies [Verma et al., 2004] that would cause a myopic approach to fail to find a valid set of bindings, even though a valid set existed. For example, a process flow in which a document is encrypted using the services of a 512-bit encryption algorithm at one step might need to ensure that there exists a compatible service that can decrypt the document in the subsequent steps. Therefore, representing and accommodating inter-service constraints is crucial to the selection of consistent set of service bindings when executing an abstract flow. To illustrate this, in Section 2 we present two motivating scenarios in which inter-service constraints pose service selection limitations.

Next, in Section 3, we present the architecture of our prototype and discuss how it works with one of the motivating scenarios. The implementation details of the prototype are presented in Section 4. We then review the related work in this area and outline our contributions in Section 5. Finally, we present our conclusions and plans for future work in Section 6.

2. Motivating Scenarios

To demonstrate the need for accommodating inter-service dependencies and constraints, we have chosen two scenarios. Both of them are variations of a purchase order scenario. The first scenario demonstrates domain constraints while the second one illustrates inter-service dependencies.

Suppose that a retailer sends an order for three electronic parts to a distributor: item 1, item 2, and item 3. The distributor has a set of preferred suppliers from whom she orders the parts. Say suppliers A, B and C can supply item 1, suppliers D, E and F can supply item 2 and suppliers G, H and I can supply item 3. Say further that there are some incompatibilities in the technology of suppliers. The incompatible sets might look like: (A, E) (B, F) (E, I) and (C, G) meaning that supplier A's technology is incompatible with that of supplier E's and so on. The job of the distributor is to fulfill the retailer's order while accounting for any technology constraints. A high-level distributor process flow is shown in Figure 1.

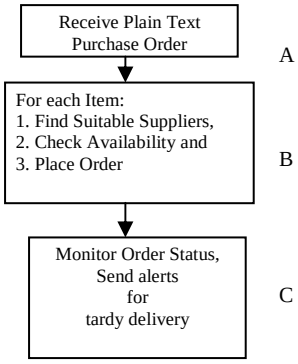

Figure 1: *A schematic illustration of Distributor process*

Step B in figure 1 can be further elaborated as follows: First, the distributor has to find suitable service providers that can supply the requested items from amongst the preferred supplier list. Second, the distributor must check for feasible and compatible suppliers based on technology constraints. Third, the distributor must verify the availability of the requested items from the suppliers and place the purchase orders upon availability confirmation. While the process of placing a purchase order could itself be a complex operation, we treat it as a single operation in this scenario. In our second scenario, we use a more complete version of placing a purchase order. Finally, the distributor must monitor the status of the ordered items on a regular basis to ensure timely delivery and to recover when orders become tardy. Figure 2 shows the detailed distributor process.

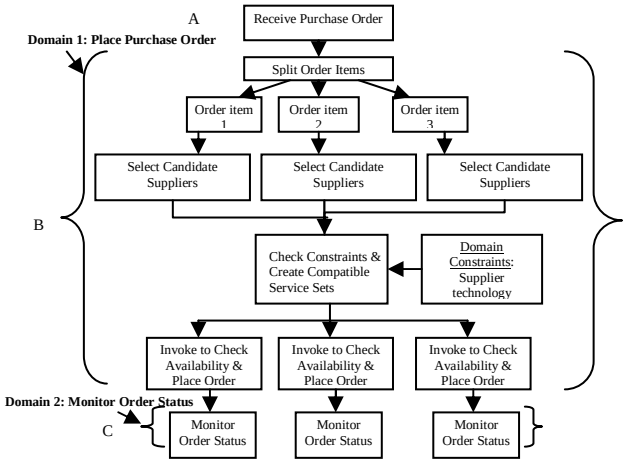

Figure 2: *A schematic illustrating the details in the distributor process*

In figure 2, we group the logical steps of the overall flow into domains. Roughly, these domains would correspond to different administrative domains or sub-domains where different policies and constraints might be enforced. We expect that the policies and constraints within each domain would be specified using WS-policy [Box et. al, 2003], and the domains could be identified from the WS-policy statements. Within each domain, there are constraints that need to be considered while binding services. For example, in domain 1 in figure 2, policies regarding desirable and allowable supplier technology combinations would dictate the selection of services to call for ordering items. In domain 2, recovery policies for tardy orders might impose constraints on recovery actions, including selection of substitutes for tardy orders. Although for this work we are assuming that domains can be clearly delineated, a more complete solution would need to allow for overlapping domains.

Our second scenario involves a retailer sending a purchase order to a distributor and monitoring the order status. Suppose that the distributor's web service has a policy that will only allow it to accept order documents that are signed and encrypted using public-key technology. The business process flow in this scenario involves finding document signing and encrypting services in the binding process. The encryption capabilities of service providers are further refined based on encryption types and key lengths such as 256-bit, 512-bit, and 1024-bit etc. The following inter service

dependencies are evident in this flow: (1) a plain text order document has to be signed and encrypted before processing (2) a document should be encrypted only after it is signed, (3) a key must be obtained before a signed document can be encrypted. This process is shown in figure 3. The high-level flow is shown in the boxes that run top-to-bottom. The details of preparing a purchase order for submission are shown in the cloud to the right. The steps in the cloud are meant to be discovered automatically during execution.

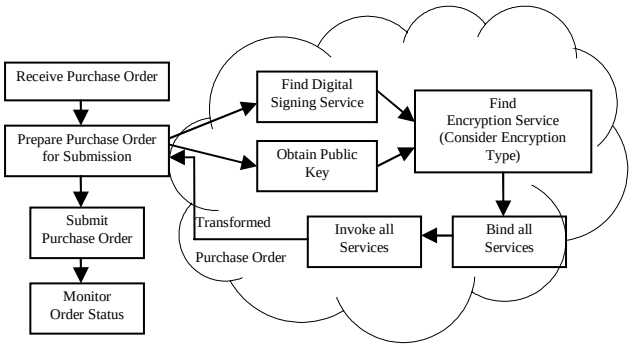

Figure 3: *Document encryption scenario process flow*

This scenario illustrates the need for accommodating inter-service dependencies during dynamic binding of Web process flows.

3. Our Solution Approach

In this section, we explain the details of our solution approach by referencing the electronic parts purchase order scenario. The key components of our architecture are shown in Figure 4. They are: a Generic Web Service Proxy, A semantic UDDI module, a planning module, a dynamic binding module and a web service invoker.

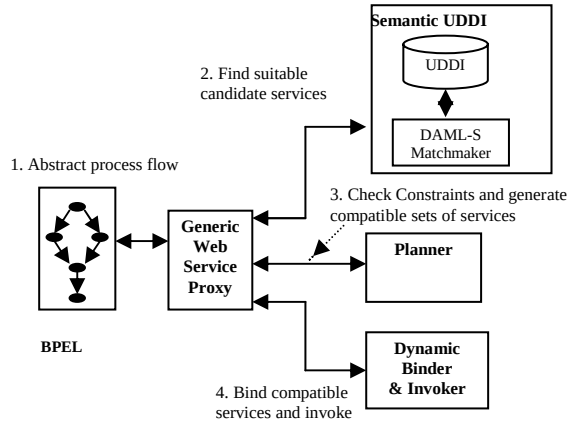

Figure 4: *Interaction flow between abstract process flow and our dynamic service binder.*

We used the following technologies in developing our prototype: (1) DAML-based of semantic markup languages for ontology (DAML+OIL), and service capability representations (DAML-S) (2) A semantic UDDI server [Akkiraju et. al 2003) for finding suitable services (3) a DAML-S matching engine [Doshi et. al 2003] for matching service semantics (4) a semantic network based ontology management system, SNoBASE [Lee et al 2003] that offers a DQL-based [DQL 2003] Java API for querying ontologies represented in DAML+OIL/OWL (5) IBM's ABLE [Bigus et al 2001] engine for inferencing (6) IBM's Planner4J planning engine [Srivastava 2004] (7) BPEL4WS for representing the process flows (8) BPWS4J, IBM's BPEL execution engine [BPWS4J 2002] and (9) WebSphere Application Server [IBM 2003]: IBM's Java application server for deploying and executing Web Services and BPEL4WS flows.

To create and execute an abstract flow, we represent the context of the process flow along with any domain constraints in DAML+OIL. In the electronic parts scenario, we represent the relationships between electronic items such as network adapters, power cords, batteries, their corresponding technologies such as network type, voltage input/output specs, Lithium-Ion (Li-Ion) battery vs. Nickel Cadmium battery (Ni-Cad). etc. Then, we instantiate distributor's preferred suppliers and capture their technology constraints in the ontology. For example, the following DAML+OIL statements capture facts such as "Type1B is an instance of Battery". "Type1B works with Type2 Power Cords", "Type1B works with Type 3 Power cords" and "Type 1B works with Type 2 Network adapters".

```
<rdf:Description
  rdf:about="<ontologyPath>#Type1B">
  <rdf:type>
    <daml:Class
      rdf:about="<ontologyPath>#Battery" />
  </rdf:type>

  <ns0:worksWith
    rdf:resource="<ontologyPath>#Type2PCh" />
  <ns0:worksWith
   rdf:resource="<ontologyPath>#Type3PCh" />
  <ns0:worksWith
   rdf:resource="<ontologyPath>#Type2NWA" />
</rdf:Description>
```

Figure 5: *Sample compatibility constraints.*

Once the domain is defined, we encode the supplier services for item availability check, purchase order receivers as Web Services in DAML-S. We deploy these DAML-S descriptions as external description in UDDI via t-Models [UDDI 2002]. These descriptions are later used in the selection of suitable services for a given set of requirements. The corresponding WSDL descriptions of these services are used for invoking the actual Web Services.

Once the domain ontologies, service semantics and the corresponding WSDL files are all created, we use them to create abstract BPEL4WS flows to represent business processes. An abstract BPEL4WS flow is divided in to a set of domains, corresponding to different policy domains. For the electronic parts purchase order process example, we define a high-level BPEL4WS document with two steps one for each domain: (1) finding suitable partners, checking item availability and placing orders (2) monitoring the status of purchase orders. A sample abstract BPEL4WS excerpt for electronics parts example with these two steps is shown below. In the first step, the retailer's purchase order request is received by the distributor's order processing Web Service. It then uses a *while* construct to loop through each order item and to source it from preferred suppliers. The order details are passed in the form of an XML domain document (which is explained later in this section).

```
<sequence>
  <invoke partnerLink="Distributor"
          portType="orderHandler"
          operation="orderHandler"
          inputVariable="PurchaseOrderInputs"
          outputVariable="OrderDetails" / >
  <flow>
    <while condition=
        "bpws:getVariableData('supplierCounter')
         <
         bpws:getVariableData(OrderDetails,
                              numSuppliers)">
      <sequence>
        <invoke
          partnerLink="GenericWebServiceProxy"
          portType="domainHandler"
          operation="invokeProxy"
          inputVariable="PurchaseOrderInputs"
        outputVariable="OrderConfirmationOutput"
          />
         <assign>
           <copy>
             <from expression=
         "bpws:getVariableData('supplierCounter'
               +'1') "/>
             <to variable="supplierCounter"/>
           </copy>
         </assign>
      </sequence>
    </while>
   </flow>
  ……
   <invoke
     partnerLink="GenericWebServiceProxy"
     portType="domainHandler"
     operation="invokeProxy"
     inputVariable="OrderConfirmationInput"
     outputVariable="OrderStatusOutput"   / >
 </sequence>
```

Figure 6: *Order item processing flow.*

This notion of domains tells our system that activities within this domain might have interdependencies and that service selection and binding should be done as an atomic operation. For instance, the technology of one service provider might be incompatible with that of another even though the capabilities of both of them match with those of requirements. We use scoping as a way of defining a manageable search space for finding compatible services. Since humans possess the inherent capability to group related things in a given problem domain, we rely on users to tell us the boundaries of domains via abstract flow definitions. In essence, these abstract flows hide the details of activities within a domain. The Generic Web Service Proxy is a Web Service defined via a WSDL document that can be statically bound to a node in the BPEL4WS flow. We use this proxy to defer specifying the execution details of the activities within a domain. We then deploy this high-level BPEL4WS document in BPWS4J, IBM's BPEL4WS execution engine-BPWS4J.

The Generic Web Service Proxy module takes the following as inputs: the semantic descriptions of the service requirements represented in DAML-S, domain constraints or service dependency constraints represented in OWL, the location of public or private UDDI registries to find suitable matches. We use the approach specified in [Sivashanmugan et al. 2004] to augment this BPEL4WS to carry the semantic descriptions of service requirements instead of the services themselves. A sample XML that captures these requirements is specified below.

```
<Partners domain = 0>
  <Partner id = 1>
    <SemSpecsURI>
  http://local/damls/RequestElectronicParts.daml
    </SemSpecsURI>
    <ConstraintsURI>
  http://local/ontologies/electronic_parts.daml
    </ConstraintsURI>
    <UDDISpecs>
      <RequestTModel>
        "Specify UUID for the request TModel"
      </RequestTModel>
      <CategoryName>
        'Electronic Components and Supplies'
      </CategoryName>
      <CategoryValue> 32.11.17.00.00
      </CategoryValue>
    </UDDISpecs>
  </Partner>
</Partners>
```

Figure 7: *Sample service requirements.*

During the execution of the high-level BPWS4J flow, at each node, the Generic Web Service Proxy gets invoked. At a high-level the Generic Web Service Proxy discovers suitable services, automatically binds feasible sets and invokes them and returns control to the upper BPEL4WS flow. BPWS4J engine then proceeds with the execution of the remaining steps of the flow.

4. Implementation Details

In this section, we describe the implementation details of our prototype. The generic web service proxy is used to find relevant sets of services that can fulfill the specified high-level requirement, and invoke those services to

obtain appropriate outputs and effects. It achieves this by coordinating the service discovery and binding activities by using the services of semantic UDDI, planner and dynamic binder and invoker modules.

4.1 Semantic Service Discovery

Service discovery is achieved via the semantic UDDI and DAML-S matchmaker modules. We have used the approach described in [Akkiraju et al 2003] to extend the service discovery capabilities of UDDI. In summary, service providers, and requesters annotate their service capabilities, and requirements using external descriptions published in UDDI as DAML-S description t-Models. When a requester invokes the find_tModel() method a service discovery proxy intercepts these requests and performs semantic matching. Specifically, the service discovery proxy retrieves a set of candidate services that are described in DAML-S and those that are advertised under related industry categories. In the electronic parts scenario, all supplier service description tModels are registered under a UNSPSC category 'Electronic Components and Supplies'. The proxy selects these services and then invokes a DAML-S semantic matching engine to refine the selection by matching the semantic descriptions of the capabilities of services with semantic descriptions of the requirements.

4.2 Planner

We treat the binding of abstract services at run time as a run time planning problem. The description of the abstract service is used as a description of the goal to be accomplished and the web services advertised in the UDDI server as the set of available operators. Domain constraints and preferences are used to limit and guide the generation of a plan. To accommodate inter-service dependencies and constraints, we treat the binding of abstract services for each domain as a single planning problem. The abstract plan forms the description of the planning problem to be solved.

The interaction between the planning module and semantic UDDI server is currently restricted to a simple protocol. Initially, the planner simply requests services that can be bound to each abstract action in the abstract plan. If such services are found, the planning task degenerates into a selection a feasible set of service bindings that meet all the constraints. If no such set can be found, the planner requests all the services with the corresponding UNSPSC code and treats them as operators available for solving the planning problem. While this is a reasonable first approach, there is no guarantee that the required services will be registered under a single UNSPSC code. This protocol may also download many more service descriptions then are required, resulting in significant performance degradation. An alternative approach would be to integrate the planner with the UDDI server, so that the server could directly return service compositions in addition to references to existing services. We've explored this approach, but one issue that we need to address is that the current UDDI specification does not allow the server to return service compositions.

In our implementation, we make use of the Planner4J planning infrastructure. This planning framework includes a number of planning algorithms and uses PDDL [Fox and Long 2002] to describe planning problems and plans. As a result, our current implementation needs to translate from the DAML-S representation of web services into the PDDL representation. We also translate the domain descriptions, constraints and goal into PDDL in order to invoke the planner. The resulting plan is then translated into a BPEL4WS flow. In some cases, the resulting plan might consist of a single web service call. As stated above, this case is handled by the UDDI semantic matcher which would find the appropriate service. In other cases where the planning module is required, the flow might include a set of series of services that need to be invoked.

4.3 Dynamic Binder and Invoker

The dynamic binder module takes the generated flows and services and binds them to the corresponding abstract nodes in the BPEL4WS flow and invokes them.

Our prototype is developed in Java and uses IBM's WebSphere Application Server deployment environment. The running time of the prototype includes the time taken to load the relevant ontologies (done once and retained in memory) and to inference the relationships. Since the size of the ontologies in our sample domains is relatively small (on the order of dozens of concepts), SNOBASE keeps all the ontological concepts and instances in memory for fast access. For larger ontologies this may not be feasible.

5. Related Work

In our view, processing high-level descriptions of process flows and generating executable flows from it consists of three steps. First, we need to augment the BPEL4WS language with semantics for representing process and service semantics. Second, we need to dynamically discover services that match given high-level descriptions using semantic matching of Web Services. Finally, we need to accommodate inter service dependencies and domain constraints in selecting suitable services to bind for a given process. Some work has already been done in all these aspects.

Sivashanmugam et al. developed a template-based approach to capturing the semantic requirements of

process services in the METEOR-S Web Service Composition Framework. [Sivashanmugam et al. 2004]. The semantic information about services in the templates can be used to dynamically discover suitable services and generate executable BPEL4WS documents. Paolucci et al [Paolucci et al. 2002b], and Akkiraju et al., [Akkiraju et al 2003] present mechanisms for dynamically discovering Web Services using semantic extensions to UDDI registry. An approach is presented by Sirin et al. for semi automatically generating process compositions using semantic capabilities of Web services [Sirin et al. 2003]. Mandel et al [Mandel and McIlraith 2002] present an approach to combine DAML-S and BPEL4WS for achieving dynamic binding. They also account for user defined constraints in service selection. A significant difference between this work and our approach is that we capture the domain of related services within the BPEL4WS flow and use this information to bind all services that are related or belong to a local domain at once to accommodate their domain constraints and service dependencies. The result is a set of bindings that are legal and feasible in the operating domain.

6. Conclusion and Future work

In this paper, we argued that business process flow specifications should be defined at abstract task level leaving the details of specific service bindings and execution flows for the runtime flow execution engine. To demonstrate the feasibility of this approach, we have presented a prototype BPEL4WS engine that supports dynamic binding of Web services in business process flows while considering inter-service and domain dependencies and constraints. Our work leverages the advances in semantic web technologies, to augment the flexibility to the current industry standards. We have illustrated our work by describing our prototype in the context of two scenarios: purchase order scenario in electronic parts domain and a secure document purchase order scenario. In these examples, we present an approach to handle dependencies between services in a process.

The limited size of the examples used in this paper doesn't allow us to fully explore the problems associated with scaling up to realistic process flows. It is our experience that real world business process flows tend to be much more complex with many more interdependencies. Scaling up to handle such challenges requires further explorations.

In this work, our intention was to demonstrate the feasibility of applying semantics to create abstract flows for which we can create a suitable runtime flow execution engine. For this, we have chosen to work within current industry standards, such as BPEL4WS as much as possible. As a follow on to our current work, we are exploring service execution monitoring and recovery of process flows that are dynamically composed using probabilistic models.

7. References

[1] Akkiraju R., Goodwin R., Doshi P., and Roeder S. 2003. A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI In the *workshop proceedings of Eighteenth International Joint Conference on Artificial Intelligence. Information Integration on the Web*. WEB-1 pg: 87-92

[2] Ankolekar A., Burstein M., Hobbs J. J., *et al*. 2001. DAML-S: Semantic Markup for Web Services. In *Proceedings of the International Semantic Web Working Symposium (SWWS).*

[3] Berners-Lee T., Hendler J., Lassila O. 2001. The Semantic Web. *Scientific American*.

[4] Bigus J., and Schlosnagle D. 2001. Agent Building and Learning Environment Project: ABLE. http://www.research.ibm.com/able/

[5] BPEL Technical Committee. 2002. Business Process Execution Language: BPEL. IBM Developer Works Article. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[6] Box,D., Curbera, F., Hondo,M., Kale, C., Langworthy, D., Nadalin, A., Nagaratnam,N., Nottingham, M., von Riegen, C., Shewchuk, J., Web 2003 Services Policy Framework (WSPolicy) *http://www.ibm.com/developerworks/library/ws-policy*

[7] Christenson E., Curbera F., Meredith G., and Weerawarana S. 2001. Web Services Description Language (WSDL). www.w3.org/TR/wsdl

[8] DAML Technical Committee. 2000. DARPA Agent Markup Language- DAML. http://www.daml.org

[9] DAML+OIL Technical Committee. 2001. DAML+OIL. http://www.daml.org/2001/03/daml+oil-index

[10] Doshi P., Goodwin R., and Akkiraju R. 2003. Parameterized Semantic Matching for Workflow Composition. Draft Paper in Preparation.

[11] DQL Technical Committee 2003. DAML Query Language (DQL) http://www.daml.org/dql

[12] Fox M., and Long, D., PDDL2.1: An Extension to PDDL for Expressing Temporal Domains, The AIPS-02 Planning Competition Committee, 2002. http://www.dur.ac.uk/d.p.long/competition.html.

[13] IBM 2002. The IBM Business Process Execution Language for Web Services Java™ Run Time

(BPWS4J). http://www.alphaworks.ibm.com/tech/bpws4j

[14] IBM 2003. IBM Websphere Application Server http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/appserv

[15] Layman F., Curberra F., Roller D., and Schmidt M. Web Services Flow Language: WSFL. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[16] Lee J., Goodwin R. T., Akkiraju R., Doshi P., Ye Y. SNoBASE: A Semantic Network-based Ontology Ontology Management. http://alphaWorks.ibm.com/tech/snobase.

[17] McIIraith S., Son T., and Zeng H. 2001. Mobilizing the Semantic Web with DAML-Enabled Web Services. *Semantic Web Workshop*.

[18] Mandel, D., McIIraith S., 2003 Adapting PBEL4WS for the semantic web: The bottom up approach to web service interoperation *Second International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, 2003.

[19] OWL Technical Committee. 2002. Web Ontology Language (OWL). http://www.w3.org/TR/2002/WD-owl-ref-20021112/

[20] Paolucci M., Kawamura T., Payne T. R., and Sycara K. Semantic Matching of Web Services Capabilities. *The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June, 2002.

[21] Paolucci M., Kawamura T., Payne T. R., and Sycara K. 2002. Importing the Semantic Web in UDDI. *In Web Services, E-Business and Semantic Web Workshop*.

[22] Sirin E., Hendler J., and Parsia B. 2003. Semi-automatic composition of web services using semantic descriptions. Web Services: *Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, April 2003.

[23] Sivashanmugam K., Miller J., Sheth A., Verma K. , International Journal of E-commerce (to appear, 2003).

[24] Srivastava, B., A Software Framework for Applying Planning Techniques. IBM Technical Report RI 04001 March 2004.

[25] Thatte S. 2001. XLANG: Web Services for Business Process Design. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[26] UDDI Technical Committee. 2002. Universal Description, Discovery and Integration (UDDI). http://www.oasis-open.org/committees/uddi-spec/

[27] Weerawarana S., Curbera F. 2002. Business Process with BPEL4WS: Understanding BPEL4WS. http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/

[28] Denker G., Kagal L., Finin T., Paolucci M., and Sycara K., Security for DAML-S Web Services: Annotation and Matchmaking, Proceedings of the Second international Semantic Web Conference, 2003, pp 335-350.

[29] Verma K., Akkiraju R., Goodwin R., Doshi P., Lee J., On Accommodating Inter Service Dependencies in Web Process Flow Composition, AAAI Spring Symposium, 2004, pp. 37-43.