On Accommodating Inter Service Dependencies in Web Process Flow Composition

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

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

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

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

verma@cs.uga.edu

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

pdoshi@cs.uic.edu

Abstract

Current business process flow representation languages such as BPEL4WS are prescriptive and operate at the execution level. They do not accommodate abstract specifications of business activities and dynamic binding of Web Services at run time. Moreover, dynamic selection of Web services for a process is, often, not a stand-alone operation. There may be many inter-service dependencies and domain constraints that need consideration in selecting legal and meaningful services for realizing an abstract flow. In this paper, we present a prototype for dynamic binding of Web Services for abstract specifications of business integration flows using a constraint-based semantic-discovery mechanism. Building on prior work in this area (Mandel and McIlraith 2002), we provide a way of modeling and accommodating scoped constraints and inter-service dependencies within a process flow while dynamically binding services. The result is a system that allows people 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 to specify business processes and business interaction protocols. It superseded XLANG (Thatte 2001) and WSFL (Layman et al. 2001) as a standard for Web services flow specification. BPEL4WS provides a representation mechanism for process execution flows consisting of a number of constructs for representing complex flows, data handling and correlation. Unfortunately, in its current specification, BPEL4WS operates at the execution layer. That is, BPEL4WS requires static binding of services to the 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 crucial to capturing service capabilities at abstract levels. Also, BPEL4WS does not specify how to model constraint scopes and inter-service dependencies in a process flow. These limitations hinder the promise of software interoperability.

Some of these limitations are already being addressed in parallel efforts by the Semantic Web community. Recently, this community has developed ontology markup languages such as DAML (DAML 2000), DAML+OIL (DAML+OIL 2001) and OWL (OWL 2002). To address the lack of semantics in the industry backed Web Services standards, the Semantic Web Community developed a DAML+OIL ontology for Web Services known as DAML-S (Ankolekar et al. 2002). This DAML family of semantic markup languages together lays 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 industry setting.

In our 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 abstract task levels leaving the details of specific service bindings and execution flows for the system to discover either automatically or semi-automatically. To investigate the realizability of this claim, we have developed a prototype that can enhance the business process flow representation of BPEL4WS with semantics to facilitate runtime discovery of Web Services. In this paper, we discuss our experiences with dynamic binding of Web Services in process flows. In particular, we contend that the selection of Web services for a step in a process flow is, often, not a stand-alone operation, as there may be dependencies on previously chosen services for the process. 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 it chooses a compatible service that can decrypt the document in the subsequent steps. Therefore, representing and accommodating context-based

constraints is crucial to the selection of legal and meaningful services in fleshing out the abstract flows. To illustrate this, in Section 2 we present two motivating scenarios in which constraints pose service selection limitations. The first scenario presents Web Service description-based constraints while the second scenario poses domain constraints. Next, we present the architecture of our prototype, and discuss how it works with one of the reference scenarios in Section 3. 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.

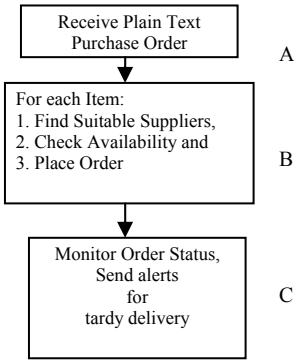

Figure 1: A schematic illustration of Distributor process

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 retailer's order while accounting for any technology constraints. A high-level distributor process flow is shown in Figure 1.

Step B in figure 1 can be further elaborated using the following steps. First, distributor has to find suitable service providers that can supply the requested items from amongst her preferred supplier list. Second, the distributor should check for feasible and compatible suppliers based on technology constraints. Third, the distributor must verify the availability of requested items from the suppliers and place purchase orders upon availability confirmation (This process of placing a purchase order could be a complex operation in itself. It could involve discovering some additional services such as document signing and encryption if the distributor requires such pre processing. This forms the focus of our second scenario). Finally, the distributor must monitor the status of the order items on a regular basis to monitor timeliness of delivery and act on tardy orders.

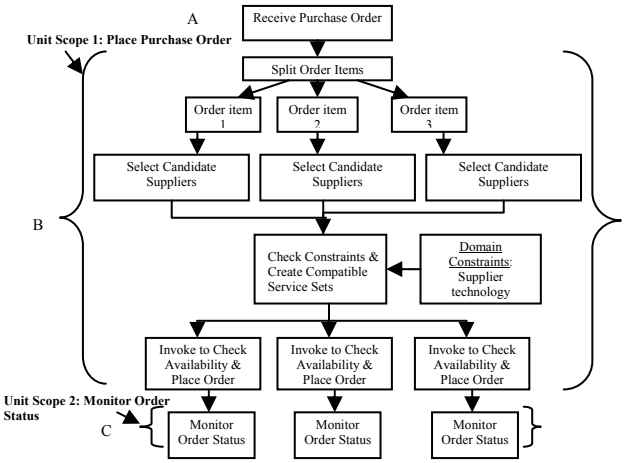

Figure 2: A schematic illustrating the details in the distributor process

In figure 2, we group the logical steps of the overall flow into units of scope. Within each unit of scope, there are domain constraints that need to be considered while binding services. For example, in scope 1, supplier technology constraints dictate the selection of services for order items. In scope 2, the delivery times of each order item might pose constraints in selecting substitutes for tardy orders. Although it is simplistic to assume that domain constraints can be separated out cleanly into units of scope, this scenario, nevertheless, brings forth the issues in dynamic binding of services.

Our second scenario involves a retailer sending a purchase order to a distributor and monitoring the order status. Suppose that the distributor requires the order documents to be 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 elaborated 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.

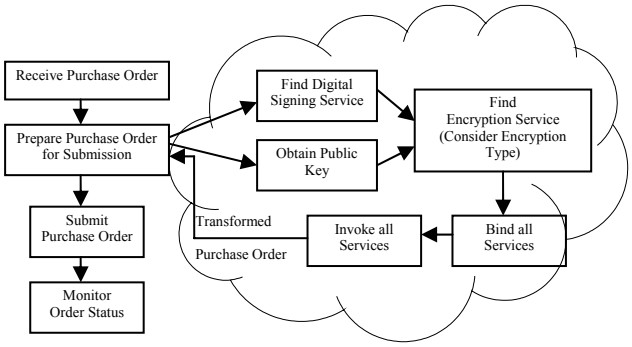

Figure 3: Document encryption scenario process flow

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 Constraint Checker, a Dynamic Binder and Invoker.

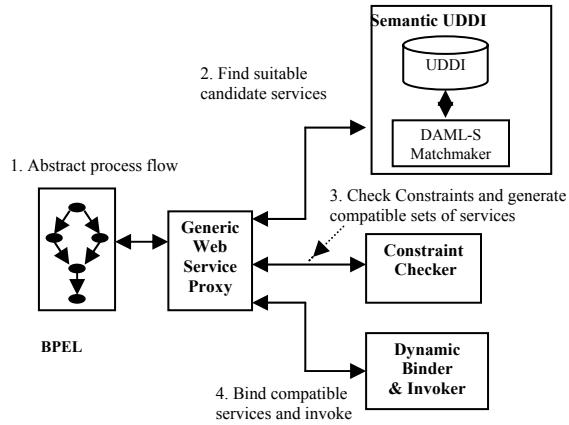

Figure 4: Interaction flow between abstract process flow and our dynamic service binder.

We used the following technologies for 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, Goodwin and Akkiraju 2003) for matching service semantics (4) a semantic network based ontology management system known as SNoBASE (Lee et al 2003) that offers 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) BPEL4WS for representing the process flows (7) BPWS4J– IBM's BPEL execution engine (BPWS4J 2002) and (8) WebSphere Application Server (IBM 2003): IBM's Java application server for deploying and executing Web Services and BPEL4WS flows.

First, 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>
```

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, users can then create abstract BPEL4WS flows to represent business processes. An abstract BPEL4WS flow is divided in to a set of unit scopes. For the electronic parts purchase order process example, we define a high-level BPEL4WS document with two steps one for each unit of scope: (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 scope 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="scopeHandler"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="scopeHandler"operation="invokeProxy"
      inputVariable="OrderConfirmationInput"
      outputVariable="OrderStatusOutput"   / >
</sequence>
```

This notion of unit of scope tells our system that activities within this scope 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 scopes via abstract flow definitions. In essence, these abstract flows hide the details of activities within a scope. We bind a generic Web Service to each unit scope thus defined in the high-level BPEL4WS process flow document. 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 unit of scope. 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. 2003) to augment this BPEL4WS to carry the semantic descriptions of service requirements instead of the services themselves. A sample XML scope document that captures these requirements is specified below.

```
<Partners scope = 0>
  <Partner id = 1>
    <SemSpecsURI>
      http://localhost/damls/RequestElectronicParts.daml
    </SemSpecsURI>
    <ConstraintsURI>
      http://localhost/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>
```

During the execution of the high-level BPWS4J flow, at each node (alternately unit scope), 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 job of The Generic Web Service Proxy is to find relevant sets of services that can fulfill the specified high-level requirement, and invoke those services to obtain appropriate outputs. It achieves this by coordinating the service discovery and binding activities by using the services of semantic UDDI, constraint checker and dynamic binder and invoker modules.

4.1 Semantic Service Discovery

This is Service discovery is achieved via 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 using semantics. In summary, service providers, and requesters annotate their service capabilities, and requirements respectively as external descriptions published in UDDI as DAML-S description t-Models. When a requester invokes a find_tModel() method (in our case this is initiated by the Generic Web Service Proxy module to find suitable bindings for a given requirement at a given node in the flow), 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. For example, in the electronic parts scenario, all the supplier service description tModels that are registered under a UNSPSC category known as 'Electronic Components and Supplies' are retrieved. The proxy then invokes a DAML-S semantic matching engine to perform matching between the capabilities of services that are retrieved in the previous step and requirements of those that are specified at a given node. Our DAML-S matching engine is capable of finding simple services as well as compositions of sets of services that together match the given requirements. In the electronic parts scenario, all suppliers that can supply the requested parts whose item availability service capabilities match that of the request specified by the distributor get returned by the matchmaker. These matching sets of services are then passed back to the Generic Web Service Proxy which invokes the Constraint Checker module to select compatible sets that meet the specified constraints.

4.2 Constraint Checker

The Constraint Checker module takes the set of suitable services selected from the previous step for each node in the flow, the domain or service constraints and creates feasible/compatible sets of services ready for binding[2]. It uses SNoBASE- the semantic network based ontology management system to infer the technological compatibility of suppliers' parts. For example, in the electronic parts scenario, if a suitable battery supplier is identified, then the subsequent matching power cord supplier and network adapter sets can be obtained by running a query that looks as follows (It is to be noted that the statements are 'and'ed by default. <ontologyPath> refers to the location of the ontology file on host system. For example, in this case it refers to: http://localhost/ontologies/electronic_parts.daml file).

(Type1B rdf:Type <ontologyPath>/#Battery)
(Type1B ns0:worksWith ?X)
(?X rdf:type <ontologyPath>/#PowerCord)
(?X ns0:worksWith ?Y)
(?Y rdf:type <ontologyPath>/#NetworkAdapter)
(?P rdf:type <ontologyPath>/#PowerCordSupplier)
(?P ns0:supplies ?X)
(?N rdf:type <ontologyPath>/#NetworkAdapterSupplier)
(?N ns0:supplies ?Y)

We provide a generic interface to perform the constraint checking. Many heuristic approaches can be implemented for generating feasible sets of services. We have implemented a greedy look-ahead algorithm in our current implementation for generating these compatible sets. Various selection criteria such as cost, time, quality etc. can be employed in choosing a compatible set from the possibly many that get generated. The chosen compatible set of services is then passed to the dynamic binder and invoker module.

4.3 Dynamic Binder and Invoker

The dynamic binder module selects appropriate services from the specified compatible sets of services and binds them to the corresponding nodes in the BPEL4WS flow and invokes them. Selection of appropriate services can be achieved by considering the cost, quality and other ratings of service providers. We have implemented a simple scheme where the first compatible set gets chosen for binding. The output of the invoked unit of scope is then passed back to the high-level BPEL4WS flow.

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 (of 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. We are currently investigating efficient caching mechanisms for SNOBASE system.

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

[2] Depending on the efficiency requirements, one might consider first generating a set of compatible set of services (and their providers) that meet the constraints and then perform semantic matching on these. In our prototype system we have chosen to first match interfaces and then pass the matching services through constraint checker.

domain constraints in selecting suitable services to bind for a given process. Some work has already been done in all these aspects.

A template-based approach to capture the semantic requirements of individual services in processes was discussed in the METEOR-S Web Service Composition Framework (Sivashanmugam et al. 2003). The semantic information about services in the templates was used to dynamically discover suitable services using METEOR-S Web Service Discovery Infrastructure (Verma et al. 2004) and generate executable BPEL4WS documents. (Paolucci et al. 2002) and (Akkiraju et al. 2003) present mechanisms for dynamically discovering Web Services using semantic extensions to UDDI registry. An approach for semi automatically generating process compositions using semantic capabilities of Web services is presented in (Sirin et al. 2003). A domain ontology for DAML-S security was discussed in (Denker et al. 2003).

In a related work, (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 scope of related services within the BPEL4WS flow and use this information to bind all services that are related or belong to a local scope 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 levels leaving the details of specific service bindings and execution flows for the system to discover either automatically or semi-automatically. To support this argument, we have presented an approach to achieve dynamic binding of Web services in business process flow composition while considering inter-service dependencies and constraints. Our work leverages the advances in semantic web technologies, to augment the flexibility to the current industry standards. We have shown the usefulness of our work by implementing our prototype in the context of two scenarios: purchase order scenario in electronic parts domain and a secure document purchase order scenario. We contend that this paper adds to current work in this area, by presenting an approach to handle dependencies between services in a process.

In this work, by clearly separating the scopes of services we have simplified the problem in many ways. The real world business process flows tend to be much more complex with many interdependencies and no clear unit of scopes. This calls for further explorations in the area of accommodating complex process dependencies. Some might even question BPEL4WS's expressiveness in representing complex flows. In this work, our intention was to demonstrate the value of applying semantics to describe business processes at higher level abstractions. For this, we have chosen to work with the current industry process flow representation language-BPEL4WS. 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

Akkiraju R., Goodwin R., Doshi P., and Roeder S. 2003. In the workshop proceedings of *Eighteenth International Joint Conference on Artificial Intelligence*. Information Integration on the Web. WEB-1 pg: 87-92

Ankolekar A., Burstein M., Hobbs J., Lasilla O., Martin D., McDermott D., McIlraith S., Narayanan S., Paolucci M., Payne T., Sycara K., 2002 DAML-S: Web Service Description for the Semantic Web, *Proceedings of the International Semantic Web Conference (ISWC),* pp. 348-363

Berners-Lee T., Hendler J., Lassila O. 2001. The Semantic Web. *Scientific American*. 284(5), pp. 34–43.

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

BPEL Technical Committee. 2002. Business Process Execution Language: BPEL. IBM Developer Works http://www106.ibm.com/developerworks/webservices/library/ws-bpel/

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

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

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

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

Doshi P., Goodwin R., and Akkiraju R. 2003. Parameterized Semantic Matching for Workflow Composition. Forthcoming

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

IBM 2002. The IBM Business Process Execution Language for Web Services Java™ Run Time (BPWS4J). http://www.alphaworks.ibm.com/tech/bpws4j

IBM 2003. IBM Websphere Application Server http://www3.ibm.com/software/info1/websphere/index.jsp?tab=products/appserv

Leymann F., Curberra F., Roller D., and Schmidt M. 2001, Web Services Flow Language: WSFL. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Lee J., Goodwin R. T., Akkiraju R., Doshi P., Ye Y. 2003 SNoBASE: A Semantic Network-based Ontology Ontology Management. http://alphaWorks.ibm.com/tech/snobase.

McIlraith, S., Son, T., and Zeng, H., 2001, Mobilizing the Semantic Web with DAML-Enabled Web Services. *In Proceedings of Autonomous Agents – Ontologies in Agent Systems (OAS 2001) workshop (Montreal, Canada),* pp. 1-11.

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

Paolucci M., Kawamura T., Payne T, and Sycara K., 2002 Semantic Matching of Web Services Capabilities. *The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), pp. 333-347.

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.

Sivashanmugam K., Miller J., Sheth A., Verma K. 2003. *Technical Report 03-008*, LSDIS Lab, Computer Science Dept., University of Georgia.

Thatte S. 2001. XLANG: Web Services for Business Process Design. http://www.gotdotnet.com/team/xml_wsspecs/xlangc/default.htm

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

Verma K., Sivashanmugam K., Sheth A., Patil A., Oundhakar S., and Miller J., 2004, METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, *Journal of Information Technology and Management* (to appear, 2004)

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