A Bottom-Up Approach to Automating Web Service Discovery, Customization, and Semantic Translation

Daniel J. Mandell
Stanford University
Knowledge Systems Laboratory
Dept. Computer Science, Stanford University
Stanford, CA 94305-9020, USA
1-650-723-7932
dmandell@ksl.stanford.edu

Sheila A. McIlraith
Stanford University
Knowledge Systems Laboratory
Dept. Computer Science, Stanford University
Stanford, CA 94305-9020, USA
1-650-723-7932
sam@ksl.stanford.edu

ABSTRACT
The ultimate goal for the Web services effort is seamless interoperation among networked devices and programs through the development of distributed computing infrastructure and Web standards. Industry has been fast off the mark with the development of computing infrastructure such as .NET and J2EE, Web service protocols such as SOAP, and a communication-level description language in WSDL. Likewise, orchestration and process modeling languages such as XLANG, WSFL, and most recently BPEL4WS, have been developed to represent Web service interactions through process models. Unfortunately, we are still a long way from seamless interoperation. Researchers in the Semantic Web community have taken up this challenge proposing top-down approaches to achieve aspects of Web Service interoperation based on techniques from artificial intelligence. Unfortunately, many of these efforts have been disconnected from emerging industry standards, particularly in process modeling. In this paper we take a bottom-up approach to integrating Semantic Web technology into Web services. Building on BPEL4WS, we present integrated Semantic Web technology for automating customized, dynamic discovery of Web services together with interoperation through semantic translation. We discuss the value of semantically enriched service interoperation and demonstrate its use in a practically motivated example. Finally, we provide an analysis of the forward-looking limitations of frameworks like BPEL4WS, and suggest that Web service interoperation specifications embrace semantic technology at a fundamental level to work towards fully automated Web service interoperation.

Keywords
Semantic Web, Web Services, BPEL4WS, DAML+OIL, DAML-S, Process Modeling, Automated Service Discovery, Automated Semantic Translation, Ontologies

1. INTRODUCTION
For many, the long-term goal of the Web services effort is seamless interoperation among networked programs and devices. Once achieved, many see Web services as providing the infrastructure for universal plug-and-play and ubiquitous computing [30]. To integrate complex, stateful interactions among services, most of the major industry players have proposed some form of business process integration, orchestration, or choreography model. These include WSCI, BPML, XLANG, WSFL, WSCL, BPSS, the Web Services Architecture, and most recently BPEL4WS from IBM, Microsoft and BEA [27]. Unfortunately, as we discuss in this paper, these standards and their associated computing machinery still place us a long way from seamless interoperability.

In parallel with these industry efforts, the Semantic Web [4] community has been developing languages and computing machinery for making Web content unambiguously interpretable by computer programs, with a view to automation of a diversity of Web tasks. Efforts include the development of expressive languages based on artificial intelligence technology, including RDF [17], RDF(S), DAML+OIL [12,29], and most recently a proposal for the Ontology Web Language (OWL) [10,21,26]. In the area of Web Services, the Semantic Web community has argued that true interoperation requires description of Web Services in an expressive language with a well-defined semantics. To this end, they have developed a DAML+OIL ontology for Web services (DAML-S) [8]. Similarly, researchers have developed automated reasoning machinery to address some of the more difficult tasks necessary for seamless interoperation including a richer form of automated Web Service discovery [25], semantic translation [20], and the ultimate challenge, automated Web Service composition [24,22,14]. Unfortunately, most of these efforts have been top-down approaches building on artificial intelligence and automated reasoning technology, sometimes grounding them in WSDL, but often times avoiding connection with evolving industry standards.

In this paper we take a bottom-up approach to incorporating Semantic Web technology into Web services. We argue that to achieve the long-term goal of seamless interoperability, Web services must embrace many of the representation and reasoning ideas proposed by the Semantic Web community and in particular by the Semantic Web services community. Nevertheless, we acknowledge that Web standards defined by industry efforts will shape the evolution of Semantic Web services. From this viewpoint we take the leading candidate for business process modeling on the Web, BPEL4WS, and its associated computational machinery, BPWS4J, and augment them with Semantic Web technology. In Section 2, we provide a reference example that we use to illustrate our contributions. In Section 3 we introduce BPEL4WS, demonstrate its use on the example, and characterize its level of automation. In Section 4, we extend BPEL4WS with a Semantic Discovery Service and introduce semantic translations to advance the level of interoperability provided by BPEL4WS. We discuss the architecture of our software, demonstrate it with respect to our reference example, and analyze its merits and shortcomings. We conclude by highlighting the distinct features brought to automated Web service interoperation systems through rich semantics, and suggesting that Web service standards developers and interoperation system designers incorporate semantic markup and reasoning into their work to achieve seamless automation of Web services.

2. A MOTIVATING EXAMPLE

To realize the value added by automated interoperation it is helpful to have a real-world example in mind. Consider the task of taking out a loan on the Web. In the absence of automation, the user invests considerable resources visiting numerous sites, determining appropriate service providers, entering personal information and preferences repeatedly, integrating potentially heterogeneous information, and waiting for responses. We would prefer that the user enters information once and receives the expected results from the most appropriate services with minimal additional assistance. One possible interaction model follows:

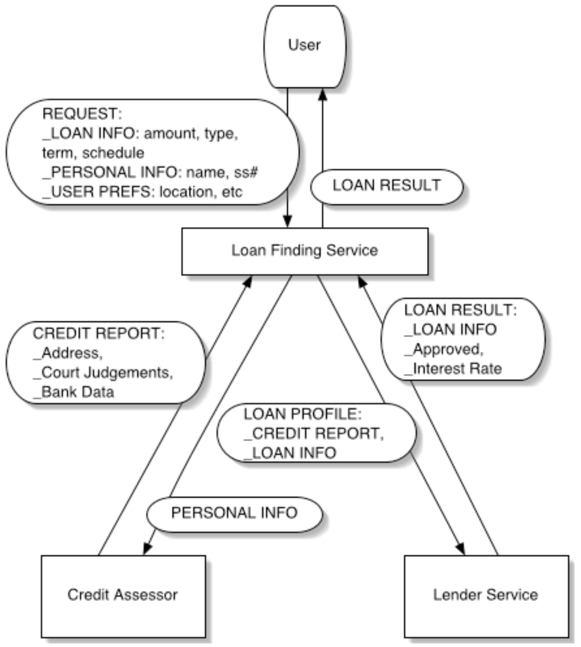

Figure 1. An interaction model for the example domain.

In this scenario, the user sends a single request to a loan finding service containing personal information, the type of loan desired, and some provider preferences. The loan finder distributes its work among two partner services: a credit assessor, which consumes the user's personal information and provides a credit history report, and a lender service, which consumes a credit report and a loan request and returns a rejection or a loan offer and its terms. The loan finder first invokes a credit assessor to generate a credit report for the user, which it then passes to the lender service along with the user's personal information. The lender service generates a result, which the loan finder reports to the user. It is no longer required that the user enter information multiple times, determine which services are appropriate, or standby to bridge output from one service to another as input. These responsibilities have been offloaded to the loan finding service and its *service provider*—the party responsible for the form and function of the loan finding service.

This interaction model is appealing for the user, but switching perspectives to that of the service provider, it remains to show how service partners are selected, ordered, invoked, and integrated.

3. AUTOMATED WEB SERVICE EXECUTION

A number of specifications and software packages are available to automate the execution of hand-written Web service compositions. Among them are BPEL4WS [7], WSCI [2], and BPML [1]. In this section we will focus on the most recently leading player, BPEL4WS.

3.1 BPEL4WS and BPWS4J

The BPEL4WS specification co-authored by IBM, Microsoft and BEA merges ideas from Microsoft's XLANG [28] and IBM's WSFL [18]. It provides a notation for describing interactions of Web services as *business processes*, following in the tradition of *workflow* modeling [31,13]. Workflow in BPEL4WS is directed by traditional control structures like *if, then, else,* and *while-loop*. Services are integrated by treating them as *partners* that fill *roles* in a BPEL4WS *process model*. The communication-level parameters of the partner services are described in accompanying WSDL [6] documents. The process model describes a program that orchestrates the interaction of the service partners. The key components of the process model are: *partners*, which associate a Web service defined in an accompanying WSDL document with a particular role; *containers*, which describe the messages passed between partners and correspond to messages in accompanying WSDL documents; *fault handlers*, which deal with known and unexpected exceptions in the spirit of the *try-catch* programming construct; and *flow*, which lists the *activities* defining the control flow of the process.

The BPWS4J engine [15], released by IBM alongside of BPEL4WS, implements a subset of the features defined in the BPEL4WS specification. The BPWS4J engine consumes a BPEL4WS document and WSDL documents defining the bindings for the BPEL4WS process and its partners. It then establishes a single endpoint for accessing the BPEL4WS process as a Web service.

3.2 BPWS4J and the Loan Example

In order to model the workflow in Figure 1, a service provider writes each of the above elements into a BPEL4WS document. For our current purposes it is worth examining the `<partners>` element.

In a BPEL4WS document modeling the interaction in Figure 1, the loan finding service interacts with three `<partners>` corresponding to the user, the credit assessor service, and the lender service. Note that the loan finding service in Figure 1 is not a partner because it corresponds to the BPEL4WS process itself. The element might be written as follows:

```
<partners>
  <partner name="user"
     serviceLinkType="lns:loanCustomerLinkType"
               partnerRole="user"/>
  <partner name="assessor"
     serviceLinkType="lns:USCreditAssessorLinkType"
               partnerRole="assessor"/>
  <partner name="lender"
     serviceLinkType="lns:loanLenderLinkType"
               partnerRole="lender"/>
</partners>
```

The `serviceLinkType` attribute selects a communication-level agreement between partners. The `partnerRole` attribute identifies which role the partner plays and which the loan finding service plays. Each role is bound elsewhere in the BPEL4WS element space to a WSDL portType.

3.3 Characterization of BPWS4J Automation

Aside from automating Web service execution with BPWS4J, BPEL4WS opens the way for automated service discovery by leaving service partners unbound at design time. Theoretically, the binding of BPEL4WS service partners to physical ports can occur at runtime. WSDL portType definitions, however, are limited to the expressiveness of XML and XMLSchema. The specification does not suggest another formal representation for describing partners outside BPEL4WS, or how representations of services may be queried to locate appropriate partners. Moreover, the version of BPWS4J available at the time of writing omits the service reference assignment feature, so a dynamically discovered service could not be referenced from within BPWS4J. BPWS4J, then, enables automated Web service execution, but not automated discovery, and BPEL4WS lacks inherent exploitation of rich, semantic descriptions of services for broader service interoperation.

Without automatic Web service discovery, the service provider is responsible for choosing service partners *a priori*, and preconcerting the service partners into an effective unit (e.g., by writing a BPELWS document). Because partner services are chosen prior to receiving the user's request, the system cannot customize partner selection for the user's specific needs or preferences. It is possible that the service selects suboptimal service partners, either because the service provider lacks a comprehensive list of potential partners at design time, or because of the difficulty in finding partners whose solution generalizes for all users. In the case of the loan example, it is possible that the user prefers to use an in-state lender because in-state loans offer tax incentives from the user's state government. If the service provider defines the lending partner prior to the user's request, the user's preference is ignored.

Further, restricting service descriptions to the expressivity of XML prevents the integration of service partners that operate on messages that have different syntax but are semantically compatible. For example, perhaps the only appropriate credit assessor for an ex-UK resident provides `UKCreditReports` while the lending service consumes `USCreditReports`. Even if these messages differed only in their representation of dates, an interoperation system that cannot recognize the semantic compatibility of the credit reports could fail to realize a potentially successful integration.

Finally, discovering and integrating the service partners manually places greater responsibility and maintenance time demands on the service provider than in the automated case.

4. AUTOMATED SERVICE DISCOVERY, CUSTOMIZATION, AND SEMANTIC TRANSLATION

In this section we present work that extends BPWS4J with customized, automated service discovery and semantic translation to address the shortcoming presented in Section 3. To enable these features, we need to address three issues:

1. How to formally represent descriptions of potential service partners
2. How to store, query and reason about such descriptions to discover appropriate partners
3. How to integrate discovered partners into the BPWS4J engine

Our approach adopts Semantic Web technologies to address the first issue, and these are described in Section 4.1. In Section 4.2, we present novel work in the form of a *Semantic Discovery Service,* which addresses the last two issues.

4.1 Supporting Technologies

We adopt several key technologies to enable the description of services in a computer interpretable format and the discovery of services with desired properties.

4.1.1 DAML-S

DAML-S is an ontology for describing Web Services based on DAML+OIL. As a DAML+OIL ontology, DAML-S has a well-defined semantics, making it computer-interpretable and unambiguous. It also enables the definition of Web services content vocabulary in terms of objects and complex relationships between them, including class, subclass, and cardinality restrictions.

The DAML-S upper ontology comprises three components:

1. *ServiceProfile* - Relates and builds upon the type of content in UDDI, describing the properties of a service necessary for automatic discovery, such as what the services offers, its inputs and outputs, and its preconditions and effects.
2. *ServiceModel* - Describes a service's process model (the control flow and data-flow involved in using the service). It is designed to enable automated composition and execution of services.
3. *ServiceGrounding* - Connects the process model description to communication-level protocols and message descriptions in WSDL.

In this section, we focus on the ServiceProfile as a declarative descriptor of Web service properties enabling automated, customized service discovery and semantic translation. We will collect DAML-S descriptions of service profiles into a repository and exploit their semantics to query for partners based on a description of the partners' required properties.

4.1.2 DAML Query Language

We adopt the *DAML Query Language* (DQL) [11] as our formal language and protocol for querying repositories of DAML-S service profiles. DQL defines the construction of queries over a repository comprised of DAML+OIL sentences. In our case, the repository is a *knowledge base* (KB) of DAML-S service profiles. DQL queries are handled by a DQL server, which interfaces with an *automated reasoner* operating over the KB. The reasoner determines which profiles satisfy the query restrictions. The DQL server answers the query by returning matching profiles in a series of *answer bundles*.

4.1.3 Java Theorem Prover

We use the *Java Theorem Prover* (JTP) [16] as the DQL server's automated reasoner. JTP is a hybrid reasoning system based on first-order logic model elimination. JTP is a particularly compelling candidate for our work because of its special purpose DAML+OIL reasoner. Since the reasoner is based on the axiomatic semantics of DAML+OIL, performance can be augmented by efficient storage of DAML+OIL sentences as triples and pre-computation of common queries.

4.2 The Semantic Discovery Service

DAML-S provides us with means to formally represent the form and function of Web services, and DQL/JTP provide us with sufficiently powerful machinery to query such descriptions. With these technologies in hand, it remains to integrate semantic service description querying into BPWS4J.

Since the current release of BPWS4J is not immediately extensible, we construct a *Semantic Discovery Service* (SDS) to work within BPWS4J's perspective as an aggregator of existing services.

4.2.1 Form and Function of the SDS

The SDS sits between a BPWS4J process and its potential service partners. Instead of routing requests to previously selected partners, BPWS4J directs them to the SDS through a locally bound Web service interface. In order for the SDS to dynamically discover customized service partners, SDS messages contain (1) the parameters to be sent to a discovered service partner, and (2) the required service partner attributes, including functional and user constraints, expressed in DAML-S sentences. The SDS then locates appropriate service partners and serves as a dynamic proxy between the BPWS4J engine and the discovered partners. With this interface come two important properties of interactions with the SDS:

1. The SDS is agnostic as to the content of the service descriptions and invocation messages it receives.
2. The SDS is stateless, with no knowledge of prior interactions, and no service-specific properties.

These properties grant the SDS portability between any BPWS4J actions and processes.

4.3 Automated Service Customization

Automated service customization, refers to the automatic selection of partners to meet preferences and constraints specific to each user. A user's request might contain preferences for a service's physical location, side-effects, quality of service and security guarantees, and many others.

In our approach to automated service customization, user constraints are encoded as DAML-S sentences in requests to BPWS4J. BPWS4J executes its process model, invoking the SDS with the DAML-S constraints whenever the workflow calls for a partner service. When invoked, the SDS sets to work at discovering a service that meets the constraints encoded in the request. The DAML-S restrictions are wrapped inside a DQL query and sent to the DQL server. The DQL server invokes the JTP DAML+OIL reasoner to compute the set of DAML-S profiles meeting the query criteria. Matching DAML-S profiles are returned to the SDS as answer bundles. The SDS selects a partner from the answer bundles and invokes the partner's endpoint with the message parameters supplied by BPWS4J. The partner does its work and responds to the SDS, which in turn forwards the response to BPWS4J. BPWS4J recovers flow control, and continues executing the process model, invoking the SDS whenever a customized Web service invocation is needed (see Figure 2).

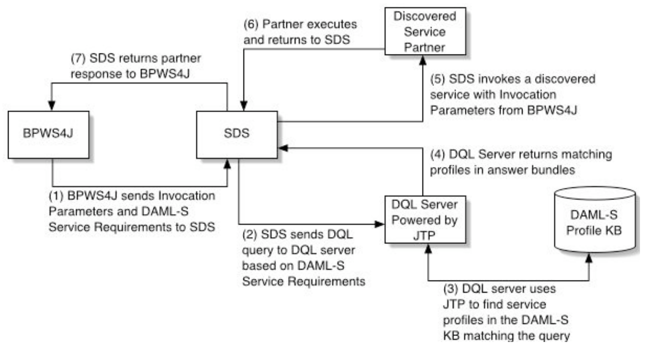
(6) Partner executes and returns to SDS
Discovered Service Partner
(7) SDS returns partner response to BPWS4J
(5) SDS invokes a discovered service with Invocation Parameters from BPWS4J
BPWS4J
SDS
(4) DQL Server returns matching profiles in answer bundles
(1) BPWS4J sends Invocation Parameters and DAML-S Service Requirements to SDS
DQL Server Powered by JTP
DAML-S Profile KB
(2) SDS sends DQL query to DQL server based on DAML-S Service Requirements
(3) DQL server uses JTP to find service profiles in the DAML-S KB matching the query

Figure 2. Interaction flow between BPWS4J, SDS, DQL server, and discovered service partners

4.4 Automated Semantic Translation

A key feature of semantically enriched data structures is their translatability within the context of automated reasoning. An automated reasoner can exploit the semantic equality of syntactically distinct classes to gain a greater reasoning space. In the context of Web services, *semantic translation* refers to redefining well-defined data types in terms of their relationships to one another via translational axioms. Semantic translation increases Web service interoperability in a number of ways, including the ability to automatically translate the inputs and outputs of service partners so they may interact seamlessly.

The SDS provides automated semantic translation for Web service discovery. Our approach uses a recursive *back-chaining* algorithm to determine a sequence of service invocations, or *service chain*, which takes the input supplied by BPWS4J and produces the output desired by BPWS4J. Our translation axioms are encoded into translation programs exposed as Web services. The algorithm invokes the DQL server to discover services that produce the desired outputs. If the SDS does not have a required input, the algorithm searches for a translator service that outputs the required input and adds it to the service chain. The process is recursive and terminates when it successfully constructs a service chain, or the profiles in the KB (or some bounded subset) are exhausted.

The following pseudocode representation of the algorithm returns a service chain, if one exists in the KB, producing the desired output while consuming only the available inputs:

```
Initialization:
weHave = {inputs provided by BPWS4J process};
weWant = {output desired by BPWS4J process};
Step:
findServiceChain (weHave, weWant) {
  svcs = getServicesOutputtingWeWant(weWant);
  foreach service in svcs {
    chain = new chain;
    foreach input in service.inputs {
      if input not in weHave {
         newSvcs = findServiceChain(weHave, service.inputs);
         chain.add(newSvcs);
      }
    }
    if all service.inputs in weHave {
      chain.add(service);
         return chain;
    }
  }
  return null; // no chain found
}
```

Note that this algorithm fits our purposes for a small DAML-S KB, but the worst-case execution time grows exponentially in the number of inputs we allow. To improve performance we could utilize a heuristic that eliminates low scoring services from the `svcs` list based on a scoring function, e.g., the minimal distance between inputs we desire and the service's outputs in a taxonomy tree, as described in [25]. Additionally, we could favor service partners requiring fewer inputs.

4.5 SDS and the Loan Example

We now consider the SDS in the context of the loan finding example from Sections 2 and 3. Assume that the user has recently moved to California, USA from the United Kingdom, so that the only potential credit-reporting agency is based in the UK. This credit assessor produces credit reports of class `UKCreditReport`. BPWS4J must then invoke a service that inputs a `UKCreditReport` and outputs a `LoanResult`. Assume further that the user must get a loan from a US lender and, moreover, wishes to borrow from a California-based lender (to take advantage of in-state tax incentives). The SDS may locate a CA-based lender using automated customization, but if the only such lender available requires a `USCreditReport` as an input the SDS would fail to discover an appropriate lender. The BPWS4J process would report that the request could not be completed.

With semantic translation, the user's request becomes satisfiable. We introduce into the DAML-S KB the profile for a `DateTranslator` service that translates between `USCreditReport` and `UKCreditReport` classes. Assume that this service implements a semantic translation axiom that, for simplicity, properly declares that the credit reports are identical except that the US version represents dates as `MM/DD/YYYY`, while the UK version uses `DD/MM/YYYY`. Since the `DateTranslator` service requires a `UKCreditReport` as an input, and the SDS has one available, the algorithm adds the `DateTranslator` to the chain. The service chain (`assessor→DateTranslator→lender`) now consumes a `UKCreditReport` and produces a `LoanResult` as desired by the BPWS4J process. The SDS executes the service chain and returns the `LoanResult` to BPWS4J.

4.6 Characterization of SDS Automation

We now characterize the level of automation provided by BPWS4J extended with a Semantic Discovery Service within the broader context of Web service interoperation. As in Section 3, the BPWS4J engine provides automated Web service execution given a BPEL4WS process model. In this section, we introduce the Semantic Discovery Service, which extends BPWS4J with automated service discovery and semantic translation. These capabilities enlarge the space of potentially successful executions, and allow the framework to account for user-defined constraints in partner selection.

The SDS does not, however, enable *automated Web service composition*. This notion is regarded in the Semantic Web community as the determination of an execution plan to accomplish an objective given a current state, and adapting that plan as state changes without human intervention. Despite the fact that our implementation does discover and execute a sequence of services to produce a desired output from provided inputs, the fundamental workflow defined in the BPEL4WS document is intentionally unchanged, as we discuss below.

The reasoning performed by the SDS is purely communicative, and ignores the preconditions and effects of the services employed. To reason about preconditions and effects is to reason about what a service does. In the case of BPWS4J, the service provider performs this reasoning manually by defining a BPEL4WS process that utilizes a predefined number of service partners with expected roles and an execution ordering. As such, the service provider imposes a particular decomposition of the process, making it inappropriate for the SDS to perform automated service composition for two reasons. First, recomposing a process without knowing the intended side effects of the original composition (i.e., those intended by the service provider) runs the risk of composing services with unintended side effects. Second, even if the SDS did have a formal description of the expected effects of the service partners – a possibility with DAML-S – recomposing the process into a new workflow reproduces the work of BPWS4J and the service provider, so the SDS would be replacing the very system it is supposed to complement. Enabling automated Web service composition within BPWS4J and similarly featured frameworks requires fundamentally redefining their roles and capabilities towards reasoning about abstract objectives and services described in languages with well-defined semantics.

5. CONCLUSION

Seamless interoperability among networked programs and devices is critical for Web services to provide an infrastructure for the vision of ubiquitous computing. We argued here that industry has taken us a step in this direction with computing machinery for automated service execution, while the Semantic Web community has developed powerful representation and reasoning technology but has remained largely disconnected from the industrial effort. In acknowledgement of the fact that Web service technology will continue to evolve from emerging industry standards, we developed software from the bottom-up that extends industrial machinery with Semantic Web technology to enable automated service discovery, customization, and semantic translation. By integrating our technology, the industrial system gained the following capabilities:

1. Automatic, runtime binding of service partners
2. Selection between multiple service partners based on user-defined preferences and constraints
3. Integration of service partners with syntactically distinct but semantically translatable service descriptions

We further argued that these capabilities approach the limit of automated service interoperation with current industrial machinery. Extending manual composition frameworks with automated composition machinery supplants provider-defined workflows with potentially undesirable recompositions. Achieving automated Web service composition requires a fundamental shift in industrial frameworks from executing predefined process models to computing and adapting execution plans from abstract objectives. In particular, in order for industry to achieve this shift, it is critical that:

1. Web service providers publish unambiguous, computer-interpretable declarations of Web service form and function, at a level of detail commensurate with the task, and in a language with a well-defined semantics
2. Web service interoperation frameworks embed automated reasoning technology into their systems and specifications that is capable of reasoning about semantic descriptions of Web services.

With the Semantic Web grounded in firm industrial support, we can begin to attain the manifold benefits of fully integrated, Web-wide distributed computation.

6. ACKNOWLEDGEMENTS

We would like to acknowledge Jessica Jenkins for her help in writing DAML-S queries and profiles and Rob McCool for his help in integrating DQL with JTP. We also thank Francisco Curbera and Bill Nagy for helpful conversations about BPWS4J, and Srini Narayanan for his useful thoughts throughout this work. Finally we gratefully acknowledge the financial support of the US Defense Advanced Research Projects Agency DARPA Agent Markup Language (DAML) Program #F30602-00-2-0579-P00001.

7. REFERENCES

[1] Arkin., A. Business Process Modeling Language. http://www.bpmi.org/bpml.esp.

[2] Arkin, A., Askary, S., Fordin, S., Jekeli,, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S. Web Service Choreography Interface. http://wwws.sun.com/software/xml/developers/wsci/.

[3] Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y., Januszewski, K., Lee, S., McKee, B., Munter, J., von Riegen, C. UDDI Version 3.0, 2002. http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm

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

[5] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., Winer, D. Simple Object Assess Protocol (SOAP) 1.1. W3C Technical report, 2000. http://www.w3.org/TR/SOAP/

[6] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web Services Definition Language. Technical report, W3C, 2001. http://www.w3c.org/TR/wsdl.

[7] Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., Weerawarana, S. Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/library/ws-bpel/

[8] DAML Services Coalition. DAML-S versions 0.5 , 0.6 and 0.7. http://www.daml.org/services/

[9] DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara). DAML-S: Web Service Description for the Semantic Web. To appear in the *Proceedings of the International Semantic Web Conference (ISWC),* July, 2002.

[10] Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., and Stein, L. OWL Web Ontology Language 1.0 Reference. http://www.w3.org/TR/2002/WD-owl-ref-20020729/

[11] Fikes, R., Hayes, P., Horrocks, I. DAML Query Language, Abstract Specification, 2002. http://www.daml.org/2002/08/dql/dql

[12] Fikes, R. and McGuinness, D. An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL, *Manuscript*. March, 2001. http://www.daml.org/2001/03/axiomatic-semantics.html

[13] Georgakopoulos, D., Hornick, M., Shet, A. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119-153, 1995.

[14] Hendler, J. and McGuinness, D. The DARPA Agent Markup Language. *IEEE Intelligent Systems, Tends and Controversies*, pp. 6-7, November/December 2000.

[15] IBM. BPWS4J. http://www.alphaWorks.ibm.com/tech/bpws4j

[16] Java Theorem Prover. http://www.ksl.Stanford.EDU/software/jtp/.

[17] Lassila, O., Swick, R. Resource Description Framework (RDF) Model and Syntax Specification.W3C Recommendation, 22 February, 1999. http://www.w3.org/TR/REC-rdf-syntax.

[18] Leymann, F. Web Services Flow Language. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[19] McDermott, Drew. "Estimated-Regression Planning for Interactions with Web Services", *Proceedings of the AI Planning Systems Conference (AIPS'02)*, June 2002.

[20] McDermott, D., Burstein, M., and Smith, D. Overcoming ontology mismatches in transactions with self-describing agents. In *Proc. Semantic Web Working Symposium*, pages 285-302, 2001

[21] McGuinness, D. and van Harmelen, F. Feature Synopsis for OWL Lite and OWL. http://www.w3.org/TR/2002/WD-owl-features-20020729/

[22] McIlraith, S. and Son, T. Adapting Golog for Composition of Semantic Web Services. *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, April, 2002.

[23] McIlraith, S., Son, T.C. and Zeng, H. Semantic Web Services. *IEEE Intelligent Systems. Special Issue on the Semantic Web.* 16(2):46-53, March/April, 2001. Copyright IEEE, 2001.

[24] Narayanan, S. and McIlraith, S. Simulation, Verification and Automated Composition of Web Services. To appear in the *Proceedings of the Eleventh International World Wide Web Conference (WWW-11)*, May, 2002.

[25] Paolucci, M., Kawamura, T., Payne, T., Sycara, K. Semantic Matching of Web Services Capabilities. To appear in *Proceedings of the 1st International Semantic Web Conference (ISWC)*, 2002.

[26] Patel-Schneider, P., Horrocks I., and van Harmelan, F. OWL Web Ontology Language 1.0 Abstract Syntax. http://www.w3.org/TR/2002/WD-owl-absyn-20020729/

[27] Proposal for Web Services Choreography Working Group Charter. W3C Architecture Domain. http://www.w3.org/2002/ws/arch/2/09/chor-proposal.html

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

[29] van Harmelen, F. and Horrocks, I. FAQs on OIL: the Ontology Inference Layer. *IEEE Intelligent Systems, Trends and Controversies*, pp. 3-6, November/December, 2000.

[30] Weiser, M. "Some Computer Science Problems in Ubiquitous Computing," *Communications of the ACM,* July 1993

[31] Workflow Management Coalition. The Workflow Reference Model. Document Number TC00-1003, Workflow Management Coalition Office, Avenue Marcel Thirty 204, 1200 Brussels, Belgium, 1994.