Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation

Daniel J. Mandell and Sheila A. McIlraith

Dept. Computer Science, Knowledge Systems Laboratory, Stanford University
Stanford, CA, 94305-9020, USA
{dmandell,sam}@ksl.stanford.edu

Abstract. Towards the ultimate goal of seamless interaction among networked programs and devices, industry has developed orchestration and process modeling languages such as XLANG, WSFL, and recently BPEL4WS. Unfortunately, these efforts leave us 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. 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 binding of Web services together with interoperation through semantic translation. We discuss the value of semantically enriched service interoperation and demonstrate how our framework accounts for user-defined constraints while gaining potentially successful execution pathways in a practically motivated example. Finally, we provide an analysis of the forward-looking limitations of frameworks like BPEL4WS, and suggest how such specifications might embrace semantic technology at a fundamental level to work towards fully automated Web service interoperation.

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, BEA, SAP, and Siebel [7]. 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 OWL-S (previously DAML-S), an OWL (previously DAML+OIL) ontology for Web services [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 [6], 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. In Section 5, we outline directions for Web service interoperation frameworks to achieve tighter integration between infrastructure and Semantic Web technologies. We conclude by highlighting the distinct features brought to Web service interoperation through richly expressive Semantic Web languages and well-defined semantics, and suggesting how interoperation frameworks might incorporate semantic markup and reasoning 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 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:

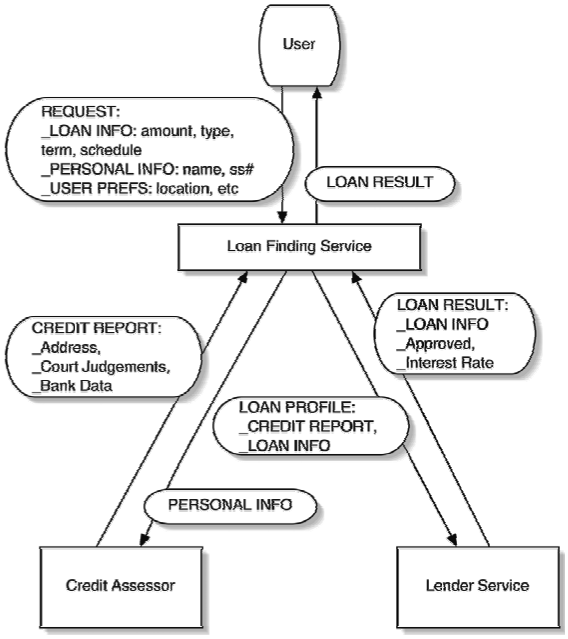

Figure 1. An interaction model for the loan example domain.

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 (*Business Process Execution Language for Web Services*).

3.1 BPEL4WS and BPWS4J

The BPEL4WS specification co-authored by IBM, Microsoft, BEA, SAP, and Siebel Systems 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 such as *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 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; *variables*, which contain 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 BPEL4WS 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 Critical Analysis of BPEL4WS Automation

BPWS4J enables automated Web service execution, and BPEL4WS opens the way for automated service discovery by leaving service partners unbound at design time. Both the engine and the specification, however, have shortcomings that limit their ability to provide a foundation for seamless interoperability.

3.3.1 Limitations in BPWS4J

Although a mechanism for dynamic partner assignment is outlined in the BPEL4WS specification, the version of BPWS4J available at the time of writing omits the service reference assignment feature, so a dynamically discovered service could not be bound to a partner role within BPWS4J. BPWS4J, then, enables automated Web service execution, but not automated Web service discovery.

Without automated discovery, the service provider is responsible for choosing service partners *a priori* and preconcerting them into an effective unit. 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. Additionally, discovering and integrating the service partners manually places greater responsibility and maintenance demands on the service provider than in the automated case.

3.3.2 Limitations of BPEL4WS

More interesting than engine-specific limitations are those inherent in the form and content of the BPEL4WS specification. Descriptions of executable and abstract processes in BPEL4WS are not declarative; they are not encoded in a manner that facilitates symbolic manipulation. As such, they are not well suited to many of the automated reasoning tasks envisioned by Semantic Web services [23].

For example, the task of binding service partners to physical ports can, theoretically, occur at runtime in BPEL4WS. Nevertheless, the description of those service partners is done via WSDL portType definitions. Effective dynamic service binding cannot be performed by solely matching WSDL messaging interfaces. Service partners should be selected based on functional, nonfunctional and behavioral descriptions of what a service does, and how it does it. Further, we argue that such descriptions must be encoded in an ontology language (e.g., [26]).

Restricting service descriptions to the expressivity of strictly syntactic WSDL interfaces limits 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. Likewise, service partners that are syntactically identical, but semantically incompatible, because different messages are described using the same name, will cause binding of functionally incompatible services.

At the heart of the problem is BPEL4WS's reliance on describing services using XML and XMLSchema. XML provides a rudimentary content language, but lacks the constructs to describe complex relationships between Web resources. While XMLSchema augments XML with a data model and enables datatyping, the semantics of XML is underspecified. Further, XMLSchema is not sufficiently expressive to create and relate rich datatypes. In contrast, RDF and RDF(S) provide a rudimentary ontology language. Not only do they provide a data model for XML, but they also enable the representation of classes, properties, domain and range, and sub-class plus super-class hierarchies. Still further expressive is OWL, and its predecessor, DAML+OIL, which also include a well-defined semantics and the ability to define complex relationships between properties of objects in an ontology. By describing services in DAML+OIL or OWL, not only do we get a more expressive language for describing service partners, but also tools for automatically reasoning about those services [16]. With the absence of automated reasoning in BPEL4WS, it is the BPEL4WS author’s responsibility to manually construct a process model that follows the operational semantics of its service partners.

4 Automated Service Discovery, Customization, and Semantic Translation

In this section we present work that extends BPWS4J with customized, dynamic binding of service partners and semantic translation to address shortcomings presented in Section 3. To enable these features, we need to consider 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 Semantic Web 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[1] 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 collect DAML-S service profiles into a repository and exploit their semantics to query for partners based on descriptions of the partners' desired properties.

[1] The DAML-S ontology has recently been translated to OWL, and renamed OWL-S [8].

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 Web services.

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.

Further information about the SDS and demonstration code is available online at (http://ksl.stanford.edu/sds).

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 other properties.

In our approach to automated service customization, user constraints are encoded as DAML-S sentences in requests to BPWS4J. Because the BPEL4WS author expects each service partner to exhibit particular functional behavior, the BPEL4WS process may also add constraints on the functional classes of service partners. These constraints are applied to the `functionalClass` property of service profiles in the KB. `functionalClass` properties, in turn, point to an ontology of service functions contained in the KB. In the absence of a computer interpretable operational semantics, referencing ontological representations of functional classes facilitates manual service composition by allowing the service partner and BPEL4WS to agree upon expected behavior.

Once the SDS receives a request for a partner service invocation, the SDS wraps the request's DAML-S restrictions inside a DQL query and sends them to the DQL server. The DQL server invokes the JTP DAML+OIL reasoner to compute the set of DAML-S service 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).

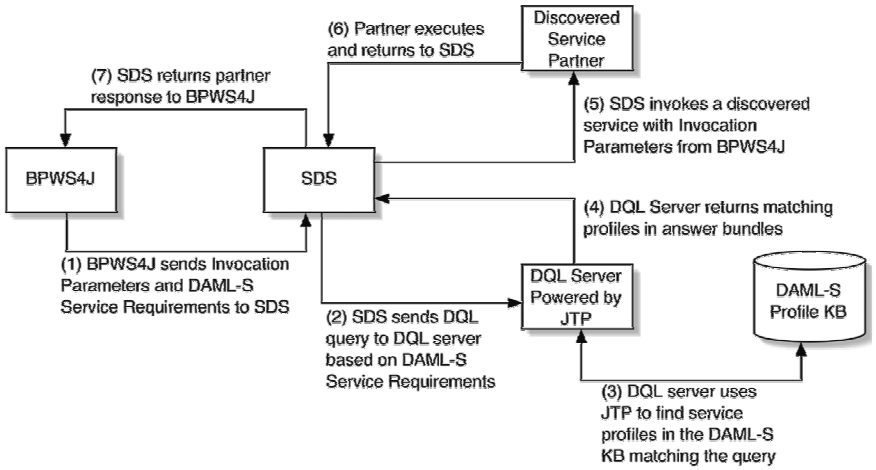

Figure 2. Interaction flow between BPWS4J, SDS, DQL server, & discovered service partners.

4.4 Automated Semantic Translation

A key feature of semantically enriched data structures is their translatability within the context of automated reasoning. In goal-directed reasoning, an automated reasoner can exploit the semantic equality of syntactically distinct classes of objects to increase the number of potentially successful execution pathways. Within the context of Web services, *semantic translation* refers to redefining well-defined data types in terms of their relationships to one another via implicit or explicit translational axioms. Semantic translation increases Web service interoperability by facilitating automatic translation of 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 translational axioms are encoded as 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 constructs a successful 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.

Also note that we only account for inputs and outputs in translation because the translator services are of the same functional class. The translation services we describe are merely implementations of translational axioms and have no preconditions or side effects.

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`. The behavior of the service is enforced by BPWS4J requiring that the partner be of functional class `creditAssesor`, which is defined in an ontology as described in Section 4.3. 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 of the required functional class using automated customization, but if the only such available lender 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 of functional class `semanticTranslator` 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 customized, dynamic service binding 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 services' preconditions and effects, aside from those accounted for implicitly by their functional classes. 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 operational semantics 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 – for example, an operational semantics defined by a DAML-S *ServiceModel* – 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. To so do, frameworks will need to shift from an *interface-oriented* perspective on Web services to one that is *functionally-oriented*.

5 Future Directions for Web Service Interoperation Systems

In this Section we briefly outline some directions for Web service interoperation frameworks to facilitate fuller integration between infrastructure and Semantic Web technologies. We consider automating service discovery and then service composition in turn, grounded in our critique and extension of BPEL4WS.

5.1 Adapting BPEL4WS for Automated Service Discovery, Customization, and Semantic Integration

As discussed in Section 3.3.2, the central shortcoming in BPEL4WS giving rise to the need for the Semantic Discovery Service is its reliance on XML and XMLSchema for describing Web service partners (for now we set aside shortcomings in the current implementation of the BPWS4J engine). BPEL4WS could subsume the capabilities provided by the SDS by relaxing its dependence on XML in favor of higher-level descriptions in a language like DAML-S. DAML-S service profiles can be used to query for service partners, as in the SDS, and the DAML-S service grounding can be

used to connect the high-level description to communication level protocols like WSDL (see [8] for details), saving the final binding to portTypes until runtime. BPEL4WS engines could easily query repositories for services using their built-in communication-level functionality.

5.2 Adapting BPEL4WS for Automated Service Composition

Automating service composition with frameworks like BPEL4WS requires a more substantial evolution, shifting its perspective as an execution framework for predefined process models to that of an automated reasoner over abstract goals. Execution plans devised to attain such goals must be adjustable as execution state changes.

Several working systems developed by researchers in the Semantic Web community perform this function [22,24,20]. These frameworks share two key components necessary to automate Web service composition. The first is a declarative representation of the capabilities of each Web service in a semantically well-defined language that is computer interpretable, as employed by the SDS or the modified version of BPEL4WS suggested in Section 5.1. Further, this information must abstractly describe the service's function as well as its form. While BPEL4WS's predecessors, XLANG and WSFL, had a well-defined semantics for describing the execution of a workflow, to date BPEL4WS has not been shown to maintain this property. Adopting the DAML-S *ServiceModel* would shift the perspective of the interoperation framework towards being functionally-oriented on service partners. Partners could then be chosen based on functional and operational constraints in addition to their communication-level properties. In the case of the loan finding example, the *ServiceModel* would formally declare how the service functions, the potential side effects, and the necessary preconditions so that automated reasoning machinery could appropriately manipulate it to determine a composition. The runtime execution of the model would be determined by an automated reasoner within the BPEL4WS engine, allowing far greater flexibility in successfully completing the user's request. In a loan finding scenario, for example, there are clearly many more ways to get a loan than the "assessor-lender" model. The composition system could propose, for example, an advance on the user's credit card, government subsidized assistance, or other solutions that meet the user's objective but may be more financially desirable or practical.

As implied above, the second necessary component to automated Web service composition is appropriate computational machinery to manipulate service descriptions to produce a composition. All of the composition work from the Semantic Web mentioned above employs such machinery operating on some model with a formal executional semantics. For example, Karmasim [24] uses reachability analysis over Petri nets. McIlraith and Son's work [22] uses Prolog and a subset of first-order logic. McDermott's work [20] likewise uses a logic-based automated reasoning system. All of these approaches, while powerful, reveal the need for further research, particularly in integration with industrial efforts. Importantly, however, this work highlights the spectrum of approaches interoperation frameworks could employ to increase their power and flexibility.

6 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.

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. We gratefully acknowledge the financial support of the US Defense Advanced Research Projects Agency DARPA Agent Markup Language (DAML) research grant #F30602-00-2-0579-P00001 and CALO research grant #NBCHD030010. Finally, we also gratefully acknowledge financial support from the Stanford Networking Research Center.

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 and OWL-S . 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. *Proceedings of the International Semantic Web Conference (ISWC)*, pp. 348-363, 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., Sheth, 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. Frank, G. "A General Interface for Interaction of Special-Purpose Reasoners within a Modular Reasoning System", Proceedings of the 1999 AAAI Fall Symposium on Question Answering Systems, pp. 57-62, 1999.
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*, pp. 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)*, pp. 482-493, 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. *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. *Proceedings of the 1st International Semantic Web Conference (ISWC)*, pp. 333-347, 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.